Aalto University
School of Electrical Engineering
Degree Programme in Automation and Systems Technology

Mikko Harju

Automated Construction of Dynamic Bayesian Networks in Simulation Metamodeling

Master's Thesis
Espoo, April 16, 2013

Supervisor: Prof. Raimo P. Hämäläinen
Instructor: D.Sc. (Tech.) Kai Virtanen

Aalto University
School of Electrical Engineering
Degree Programme in Automation and Systems Technology

ABSTRACT OF
MASTER'S THESIS

Author: Mikko Harju

Title:
Automated Construction of Dynamic Bayesian Networks in Simulation Metamodeling

Date: April 16, 2013 **Pages:** 6 + 71

Professorship: Systems and operations research **Code:** Mat-2

Supervisor: Prof. Raimo P. Hämäläinen

Instructor: D.Sc. (Tech.) Kai Virtanen

This thesis introduces an automated approach for constructing dynamic Bayesian networks (DBNs) in simulation metamodeling. DBN metamodels permit studies dealing with simulation data produced by discrete event simulation (DES) models. The new approach allows easier and faster construction of such metamodels without requiring detailed knowledge of the methodology of Bayesian networks. Deficiencies in previously created DBN metamodels are thus readily corrected by creating new refined models. This increases the overall accuracy and usability of DBN metamodels.

DES is an event based form of stochastic simulation that enables the study of the time evolution of the variables of the underlying system. Simulation metamodels are used to investigate the properties of simulation models by describing their behavior in the form of input-output mappings. In DBN metamodels, a DBN represents the joint probability distribution of the time-dependent variables of a DES model. The utilization of DBNs in metamodeling, unlike the use of input-output mappings, therefore enables investigations involving time-dependent variables. Unconditional and conditional time evolutions, i.e., the evolution over time of marginal or conditional probability distributions, can be studied. This allows for various forms of what-if analysis.

The automated approach to the construction of DBN metamodels presented in this thesis includes design of experiment, preprocessing of the simulation data, selection of the variable specific time instants for the DBN, creation of the DBN, and validation of the DBN. In addition, this thesis introduces the concept of multiple time scales in DBNs which allows for more accurate DBNs without increasing their size. An implementation of the approach, a tool for constructing DBN metamodels, is also presented. Constructing DBN metamodels with the tool verifies the practicality of the automated approach. The use of the approach and the tool is illustrated by two example simulation studies dealing with air combat and the operation of an air base.

Keywords: automation, discrete event simulation, dynamic Bayesian networks, simulation, simulation metamodeling

Language: English

Aalto-yliopisto
Sähkötekniikan korkeakoulu
Automaatio- ja systeemitekniikan koulutusohjelma

DIPLOMITYÖN
TIIVISTELMÄ

Tekijä:	Mikko Harju		
Työn nimi:	Dynaamisten Bayes-verkkojen automatisoitu konstruointi simulaatiometamallinnuksessa		
Päiväys:	16. huhtikuuta 2013	**Sivumäärä:**	6 + 71
Professuuri:	Systeemi- ja operaatiotutkimus	**Koodi:**	Mat-2
Valvoja:	Prof. Raimo P. Hämäläinen		
Ohjaaja:	TkT Kai Virtanen		

Tässä työssä esitellään uusi lähestymistapa dynaamisten Bayes-verkkojen (dynamic Bayesian networks, DBNs) automatisoituun konstruointiin simulaatiometamallinnuksessa. DBN-metamallien avulla tutkitaan diskreetillä tapahtumasimuloinnilla (discrete event simulation, DES) luotua simulointidataa. Lähestymistavan avulla kyetään konstruoimaan DBN-metamalleja helposti ja nopeasti tuntematta Bayes-verkkojen toimintaa lähemmin. Aiemmin konstruoitujen DBN-metamallien mahdollisia puutteita voidaan korjata vaivattomasti luomalla uusia paranneltuja metamalleja. Tämä menettely parantaa DBN-metamallien tarkkuutta ja käytettävyyttä.

DES on stokastinen simulointimuoto, joka mahdollistaa mallin muuttujien arvojen aikakehityksen tarkastelun. Simulointimetamalleilla tutkitaan simulointimallien ominaisuuksia kuvaamalla niiden sisäänmenojen ja ulostulojen välistä yhteyttä. DBN-metamalleissa DBN kuvaa DES-mallin aikariippuvien muuttujien yhteisjakauman, minkä avulla voidaan tarkastella muuttujien reuna- ja ehdollisten todennäköisyysjakaumien aikakehitystä. Tämä mahdollistaa erilaiset mitä-jos -analyysit, joita ei voida toteuttaa pelkillä sisäänmeno-ulostulokuvauksilla.

Tässä työssä esiteltävä lähestymistapa DBN-metamallien automatisoituun konstruointiin koostuu koesuunnittelusta, simulointidatan esikäsittelystä, muuttujakohtaisten ajanhetkien valinnasta DBN:ää varten, DBN:n luomisesta sekä metamallin validoinnista. Automatisoidun mallintamislähestymistavan lisäksi tässä työssä esitellään DBN:ien muuttujakohtaiset aikaskaalat, joiden avulla kyetään konstruoimaan tarkempia DBN:iä kasvattamatta niiden kokoa. Esitettyyn lähestymistapaan perustuen kehitetään DBN-metamallien konstruointityökalu. Työssä havainnollistetaan esitetyn lähestymistavan ja konstruointityökalun käyttökelpoisuutta kahdella esimerkkitapauksella, jotka liittyvät ilmataistelua ja ilmatukikohdan toimintaa kuvaaviin simulointimalleihin.

Asiasanat:	automatisointi, diskreetti tapahtumasimulointi, dynaamiset Bayes-verkot, simulaatiometamallinnus, simulointi
Kieli:	Englanti

Contents

Preface

This thesis builds on the doctoral dissertation of Dr. Jirka Poropudas and his research in the field of simulation metamodeling. Jirka is the person responsible for presenting the topic of utilizing Bayesian networks in metamodeling to me. Without him, this thesis would never have been started.

I would like to thank the instructor of this thesis, Dr. Kai Virtanen, for all his advice regarding academic writing. Mr. Jouni Pousi's assistance on various technical issues related to the thesis is also highly appreciated. I also want to thank the supervisor of this thesis, Prof. Raimo P. Hämäläinen, who in his capacity as the director of the Systems Analysis Laboratory research group has created an environment where junior scientists are encouraged to take the first steps on the road from student to researcher.

Finally, I wish to thank the staff at the Systems Analysis Laboratory for providing an enjoyable workplace and my parents for patiently supporting my academic pursuits.

Espoo, April 16, 2013

Mikko Harju

1 Introduction

Conducting experiments is a natural way to investigate the properties of various systems. Rather than conducting experiments on the actual system, a model of it can be created for experimenting on. Simulation models (see, e.g., [28]) are used to achieve this, as are physical models in some circumstances. This thesis focuses on discrete event simulation (DES, see, e.g., [2]) which is an event based form of simulation. DES is a popular simulation methodology for modeling and analyzing a wide range of systems. For further information on DES analysis, see, e.g., [28]. An example application of DES analysis related to decision making in aircraft maintenance is provided in [31].

DES models are becoming increasingly complex. The analysis of such models can be computationally demanding if appropriate methodologies are not used. Simulation metamodels (see, e.g., [3, 8, 13]) can assist in simulation studies by describing the behavior of DES models without considering their inner logic and dynamics. A metamodel expresses properties of the output of a simulation model as a function of its input. Utilizing dynamic Bayesian networks (DBNs, see, e.g., [32]) in metamodeling is a new approach [36, 38, 39]. Such metamodels consider the joint probability distribution of the variables in a DES model [39], including state variables. What-if analysis that is not possible with mere input-output mappings can therefore be conducted with the help of DBN metamodels [34, 38, 39]. This includes analysis involving state variables at specific time instants and multiple output variables.

The construction of a metamodel that describes the behavior of a simulation model accurately enough is not always a simple process. By automating aspects of it and only leaving certain key decisions to be made manually, the process is simplified significantly. Automation has previously been applied in simulation analysis to, e.g., select the number of simulation replications [20] and estimate the warm-up length of models [21]. Automation in the construction of metamodels has also been considered on a general level [7].

This thesis describes a new approach to the automated construction of DBN metamodels representing DES models. Issues related to the construction of such models are the primary focus of the thesis. New means are adopted to improve certain aspects of the metamodel construction in order to make it more suitable for automation. In addition, multiple time scales, i.e., different degrees of temporal detail for different variables, are included. This is a new feature in DBN metamodels. An implementation of the automated construction of DBN metamodels is also presented in the form of a tool for constructing DBN metamodels (referred to as the DBN metamodeling tool in this thesis). The application of the approach is illustrated with two example

studies.

A DBN metamodel takes into account all the variables of a DES model. The variables are categorized into three types: input variables, state variables, and output variables [48]. DES models are utilized by employing the Monte Carlo method (see, e.g., [43]), i.e., repeating the same simulation many times but with different random numbers. Different values are also used for the input variables in order to investigate their impact on the output variables. The observed values of the output variables are then analyzed. This is a time consuming process that can be bypassed with the help of metamodels [13, 14, 34]. While a simulation model attempts to imitate the behavior of the underlying system, metamodels are used to express the relation of the inputs and outputs directly. Metamodels do not generate individual observations of the output variables – the behavior of the outputs is instead expressed as a function of the inputs.

For any given values for the inputs of a simulation model, the outputs follow some usually unknown and highly complex probability distribution. In its most basic form, a metamodel is a mapping from the inputs to the expected values of outputs. It can, e.g., take the form of a regression model [5, 25]. It may also be of interest to assess other properties of the output. There exist different types of simulation metamodels based on, e.g., neural networks [3] or Kriging [24].

A Bayesian network (see, e.g., [22]) represents the joint probability distribution of a set of random variables as a directed acyclic graph where nodes represent variables and arcs their dependencies. DBNs are Bayesian networks where temporal information is also considered [32, 33]. Each node represents either a time-dependent random variable at a specific time instant or a random variable that does not change value over time.

Interpreting the random variables of a DBN as the variables of a simulation model results in a metamodel that provides the probability distributions of both outputs and state variables at all times included in the DBN [39]. Additionally, interpolation is used to generate approximations for probability distributions at time instants that are not included in the DBN [38]. The DBN metamodel thus describes the time evolution of the state variables of the simulation model which allows for analysis that is not possible with input-output mappings. DBN metamodels are used to conduct what-if analysis by examining conditional probability distributions.

The construction of a DBN metamodel consists of several phases. These are design of experiment, preprocessing of the data, selection of time instants for the variables in the DBN, creation of the DBN, and validation of the metamodel. Automation of these phases in order to make the automated approach to constructing DBN metamodels as efficient as possible is presented

in this thesis. The selection of time instants in particular is modified from previous approaches for constructing DBN metamodels by introducing a new algorithm based on dynamic programming (see, e.g., [4]).

A DBN contains nodes corresponding to the same time instants for all of the relevant variables. In this thesis, however, the time instants in the DBN are selected separately for each state variable. Unnecessary nodes are excluded which results in a smaller DBN and therefore a smaller metamodel without loss of accuracy. The metamodel requires less memory and has better computation efficiency when performing analysis. The automated approach presented in this thesis ensures that the variable specific selection of time instants does not significantly complicate the construction.

The practicality of the automated approach is illustrated and verified by applying the DBN metamodeling tool and examining two example applications. The tool provides a graphical user interface for managing the construction of DBNs by visualizing results and letting the user easily input information when needed. It is run in the MATLAB [29] environment and uses the SMILE [11] library to construct and utilize Bayesian networks. The example applications presented in this thesis deal with air combat and the operation of an air base. In these examples, various aspects of the automated construction are demonstrated and the resulting metamodels are analyzed.

The thesis proceeds by presenting the concepts of simulation metamodeling and DBNs more closely in Sections 2 and 3, respectively. Section 4 introduces the phases involved in the construction of a DBN metamodel in detail with emphasis on the automation of the construction process. Section 5 describes how to utilize the metamodel once it has been constructed. An implementation of the automated construction of DBN metamodels in the form of the DBN metamodeling tool is discussed in Section 6. Section 7 presents example applications of DBN metamodels. Finally, Section 8 contains a summary and concluding remarks.

2 Simulation modeling and metamodeling

This section provides background on simulation modeling and metamodeling. Simulation models are first discussed in Section 2.1. One form of simulation modeling, discrete event simulation (DES), is considered more closely in Section 2.2. Section 2.3 deals with simulation metamodels.

2.1 Simulation models

In a simulation study, the behavior of a system is investigated by creating a mathematical model that imitates it. A computer implementation of the model is then constructed for analysis. Reasons for using a simulation model instead of experimenting with the actual system include speed, cost, and safety. It might also be downright impossible to use the actual system, e.g., because it doesn't exist yet or because it can only be run once. For general information about simulation, see, e.g., [1, 28].

Simulation models are categorized according to various criteria. The flow of time is implemented in dynamic simulation models [2]. A stochastic simulation model [2] includes at least one random variable. A simulation model that is not dynamic is considered static and a model that is not stochastic is deterministic. This thesis considers only dynamic stochastic simulation models.

Stochastic simulation models are generally far too complex to be solved analytically. Instead, a model is examined by running it multiple times, i.e., generating observed values for its variables, and analyzing the outputs. Pseudo-random numbers (see, e.g., [6]) are used to imitate stochastic behavior in a system.

The inputs of a simulation model are assigned values at the start of the simulation. They can, for instance, be parameters that determine the arrival intensity of a queue. A simulation model does not necessarily have any inputs at all. State variables in a dynamic simulation model are time-dependent random variables that describe the current state of the system at every time instant. The outputs are random variables that describe the outcome of each run of the simulation model. Common examples include the final and mean values of variables describing the state of the system, e.g., the average length of a queue. In this thesis, input, state, and output variables are denoted by u, $x(t)$, and z respectively. Indexes are used to identify separate variables of the same type when necessary. Fig. 1 illustrates the structure of a dynamic simulation model.

One further criterion for categorizing dynamic simulation models is the

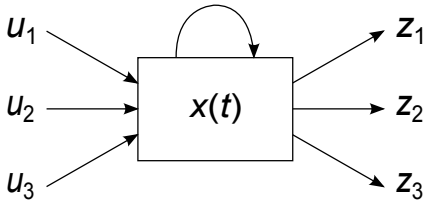


Figure 1: Structure of a dynamic simulation model with three inputs denoted by u_1, u_2, and u_3, the state variable $x(t)$, and three outputs z_1, z_2, and z_3.

manner in which the evolution of the state variables' values is described. In continuous simulation models, differential equations are given for the variables and the evolution is simulated using numerical integration [1]. In discrete simulation models [2], the values of variables only change when an event occurs and remain constant between the events. Both types of simulation models have their uses and are sometimes also combined [48]. This thesis focuses on discrete simulation.

2.2 Discrete event simulation

A DES model [1, 2] progresses as a series of changes of the state called events. What these events represent varies from case to case. They can, e.g., be arrivals or departures at some location. The execution of a DES model proceeds from one event to the next in chronological order. Each event consists of changes to the values of the model's variables and the creation of new events. The creation of new events is a stochastic procedure, both concerning the timing and the content of the event. The list of events is therefore not known in advance but created dynamically during each simulation run.

The progression of a simulation run can be described explicitly by listing the initial state of the system, the timing and changes in the values of variable at each event, and the terminating time of the simulation. The terminating time can be excluded if it is defined directly in the model rather than determined during execution. The value of any variable at any time instant can be determined from the produced simulation data. The last preceding change of value for the variable in question is looked up, or the initial value is used if no change in value has occurred. With the help of the Monte Carlo method [43], it is then possible to investigate the probability distribution of any variable at any time instant.

There exist many software packages for conducting discrete event simulation including ProModel [40], Arena [41], and SimEvents [30]. Alternatively, DES models can be constructed using general programming languages. The platform that is used determines how the model is constructed and how

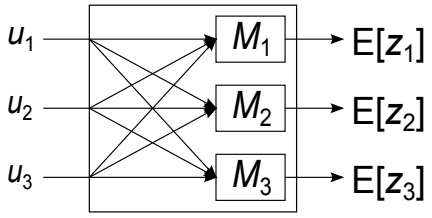


Figure 2: Structure of a metamodel corresponding to the simulation model in Fig. 1. The metamodel is a mapping from the inputs u_1, u_2, and u_3 to the expected outputs E[z_1], E[z_2], and E[z_3]. The metamodel consists of a model M_i for each output. The state variable $x(t)$ is not included in the metamodel.

random numbers are generated. In the context of DBN metamodels, the format of the DES model is irrelevant. Only the data produced by the model is of interest.

2.3 Simulation metamodels

A simulation metamodel [3, 13] describes the behavior of a simulation model without imitating the underlying system. The marginal and conditional probability distributions of the output variables are often of particular interest. Metamodels are created to better understand the operation of the underlying system, to predict the behavior of the system, to find optimal inputs, and to validate the simulation model [26]. The metamodel is constructed based on simulation data and optionally also prior knowledge of the underlying system.

There are various metamodeling techniques available, usually designed for specific situations. The first metamodels were simple regression mappings from the input to the expected output [5]. Other more complex metamodeling methods have been developed over time including radial basis function [3], neural network [3], spatial correlation [3], frequency domain [3], and Kriging [24]. Many metamodels are single output models [26]. A separate metamodel is constructed for each output when a system with multiple outputs is considered, although this collection of single output metamodels can be considered as one large multiple output metamodel. Any dependence between outputs is lost with this approach. Fig. 2 shows the structure of such a metamodel corresponding to the simulation model in Fig. 1.

The most significant advantage of simulation metamodels over simulation models is computational efficiency. They describe the behavior of outputs as a function of inputs without the need to consider individual observations of variables [14]. However, constructing metamodels can be laborious. Once a metamodel is completed, though, it can give useful results quicker than

a simulation model. The DBN based metamodels [35, 38, 39] discussed in this thesis include state variables and not just input and output variables. They therefore consider the entire probability distributions of output variables rather than just their expected values and also take into account the joint probability distribution of all variables. DBN metamodels are discussed in more detail in Section 3.3.

3 Dynamic Bayesian networks (DBNs) in simulation metamodeling

This section provides background on DBNs. Bayesian networks are first discussed in Section 3.1. Section 3.2 deals with features that are specific to DBNs. The concept of DBN metamodels is introduced in Section 3.3.

3.1 Bayesian networks

Bayesian networks (see, e.g., [22, 33]) are utilized in numerous different fields, ranging from computational biology [15] to software maintenance [46], health care [42], and air combat [47]. A Bayesian network represents the joint probability distribution of a finite number of random variables as a directed acyclic graph. The variables are usually discrete valued, although Bayesian networks can also be applied to some continuous variables [33]. Only discrete valued Bayesian networks are considered in this thesis.

Each node in a Bayesian network corresponds to a random variable. The arcs of the graph indicate dependencies between the variables – the probability distribution of the child node is dependent on the value of the parent node. The Bayesian network representation of a joint probability distribution is not always unique. It is often possible to reverse the direction of some arcs and to adjust the probabilities accordingly, leading to another Bayesian network with the same joint probability distribution. The alignment of the arcs that is most consistent with the actual causality between the random variables in question is preferred. This requires additional knowledge of the system that is considered. An example of a Bayesian network is shown in Fig. 3.

The joint probability distribution of the variables in a Bayesian network is defined by conditional probability tables (CPTs). There is a CPT for every node, where one can look up any combination of values for the node's parents and get the probability distribution for the node when the parents obtain these values. The CPT for nodes without any parent nodes simply consists of the marginal probability distribution of the node. Table 1 depicts the CPT of the node G in the Bayesian network presented in Fig. 3.

To get the probability of any combination of values for all of the nodes, the corresponding conditional probabilities of the nodes are simply multiplied. Software exists for efficient calculation of marginal and conditional probabilities of any node [10]. A common application of Bayesian networks is to fix the values of some variables and examining how this affects the probability distributions of other variables [33].

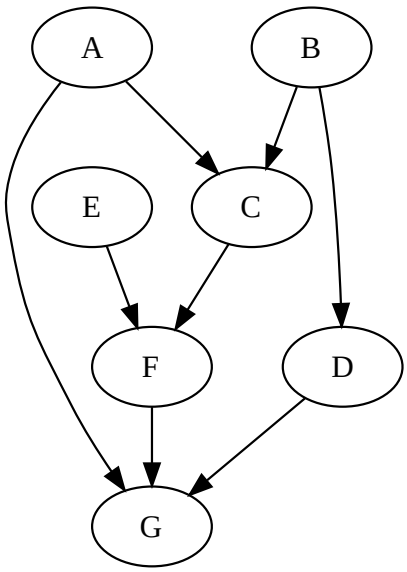


Figure 3: Example of a Bayesian network.

Table 1: Example of a CPT for the node G in Fig. 3. The top three rows contain the values of G's parents, while the bottom four rows contain the probabilities of the different values of G for each combination of values for the parents.

A	0				1				2			
D	0		1		0		1		0		1	
F	0	1	0	1	0	1	0	1	0	1	0	1
0	.1	.2	0	.1	0	.1	0	.1	.6	.7	.8	1
1	.2	.1	.3	.2	.4	.3	.4	.3	.2	.2	.1	0
2	.3	.4	.2	.3	.4	.5	.4	.5	.2	.1	.1	0
3	.4	.3	.5	.4	.2	.1	.2	.1	0	0	0	0

3.2 Dynamic Bayesian networks

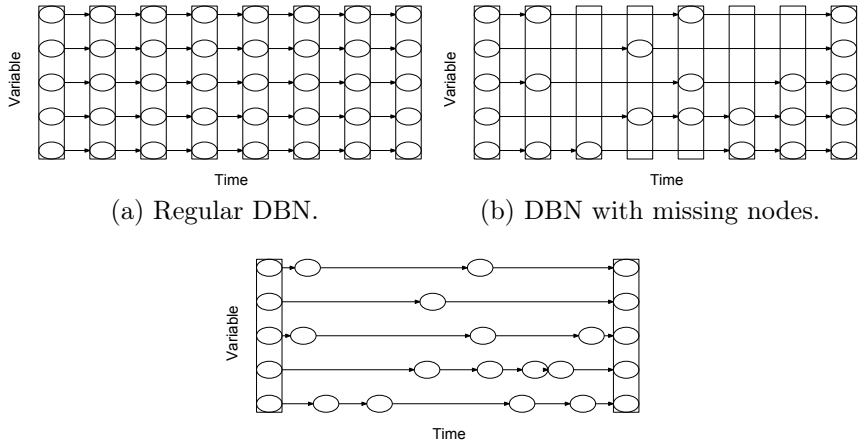
A DBN [32] is a Bayesian network where timing is also taken into consideration. Each node related to a time-dependent variable corresponds to the variable at a specific time instant. This typically means that the probability distribution of the variable changes over time. Technically, a DBN is no different from other Bayesian networks. Arcs have the same purpose, although they should be aligned so that a node never corresponds to an earlier time instant than any of its parent nodes. CPTs also work as they do with regular Bayesian networks. A DBN can also include variables that are not time-dependent. They are represented by single nodes in the graph. For convenience, such nodes might still be assigned to some specific time instant.

While the technical aspects of DBNs are similar to those of other Bayesian networks, there are more links between the nodes in DBNs. The time evolution of a variable can be examined with the help of the nodes corresponding to it. All the nodes corresponding to the same time instant compose a structure referred to as a time slice. The time slices can be viewed as layers of the DBN, with arcs either contained within a layer or leading to a later one. Arcs going from each node to the node corresponding to the same variable in the following time slice are usually included in DBNs.

Arcs going forward in time do not necessarily lead to the next time slice that includes the variable in question. They can also go further ahead. This indicates that the underlying system is one where past states, and not just the current ones, influence the future.

DBNs are constructed such that every time slice includes nodes corresponding to all time-dependent variables. As a result, the DBN can contain many redundant nodes. If one variable changes value much more rapidly than another, it makes sense for the second variable to have fewer nodes in the DBN. Similarly, a variable might change value more often within a specific time interval than at other times. In this case, it is reasonable for the variable's nodes to correspond to time instants more frequently within the time interval than outside it.

The concept of selecting time instants for nodes separately for different variables is illustrated in Fig. 4. The graph of a DBN is shown in Fig. 4a, while in Fig. 4b, excess nodes are removed. The concept of common time slices can also be discarded completely. In Fig. 4c, the time instants are chosen separately for each variable, apart from the initial and final times. For simplicity, any arcs between variables are omitted in Fig. 4. The structure shown in Fig. 4c is referred to as multiple time scales in this thesis. The DBNs presented in this thesis feature multiple time scales, since DBNs with this property describe the time evolution of the variables more accurately



(a) Regular DBN. (b) DBN with missing nodes.

(c) DBN with multiple time scales.

Figure 4: Example DBNs with different time slice structures. The nodes on each row correspond to the same variable and the nodes on each column to the same time instant.

than other DBNs with the same number of nodes.

3.3 DBN metamodels

A DBN can be utilized as a simulation metamodel by associating its nodes with variables in the simulation model [39]. Continuous variables of the simulation model are discretized to make them suitable for Bayesian networks and the values of discrete variables can also be combined into fewer unique values. The nodes of the DBN correspond to the input and output variables of the simulation model as well as the state variables at chosen time instants. The DBN thus represents the joint probability distribution of all the variables of the simulation model.

The possibility to examine the joint probability distribution of all variables separates DBN metamodels from input-output mappings. State variables are included in the metamodel at specific time instants and, as is discussed in Section 5.2, interpolation is used to approximate the probability distribution for other time instants as well. This allows for various forms of what-if analysis that are not otherwise possible. Since the DBN describes the joint probability distribution of variables, the dependency between outputs is taken into account and there is no need to construct separate metamodels for each output variable. The utilization of the joint probability distribution also enables inverse reasoning where the values of outputs are fixed and the values

of inputs are instead examined.

Simulation metamodels based on DBNs were first introduced in [36] with an example application of the methodology. In [37], influence diagrams, which are Bayesian networks that include decision nodes, are incorporated in DBN metamodeling in order to aid in decision making. The DBN based metamodeling methodology was expanded to handle continuous time in [38] with the help of multivariate interpolation. Multiple input and multiple output metamodels based on Bayesian networks are presented in [35]. DBN metamodels have previously been constructed specifically for each simulation model. This can be time consuming, in particular since constructing DBN metamodels is often an iterative process. The automated approach to constructing DBN metamodels presented in this thesis simplifies the construction significantly, thus making DBN metamodels a more viable option in practical studies. The introduction of multiple time scales makes the metamodels more efficient by eliminating redundant information.

4 Automated construction of DBN metamodels

In order to automate the construction of DBN metamodels, the process must be clearly structured. It is divided into five distinct phases in this thesis. A chart of the phases is presented in Fig. 5.

The first phase is the creation of simulation data which includes determining the number of replications, i.e., the number of times the simulation model is run, determining the values of input variables for each replication, performing the simulation, and storing the data. This phase is discussed in Section 4.1. The phase is skipped if the DBN metamodel is constructed based on some preexisting data set rather than data generated in connection with the construction of the metamodel. The simulation data is preprocessed in the second phase which is considered in Section 4.2. The preprocessing consists of defining the variables, i.e., selecting which variables of the simulation model to include in the metamodel, distinguishing between input and output variables, and discretizing the values of the variables if required.

The third phase is discussed in Section 4.3 and consists of selecting the time instants that are included in the DBN for each variable. The selection is performed separately for each variable but is treated as a separate phase from the preprocessing of the data due to being computationally the most demanding step in the construction of the metamodel. The graph of the DBN, i.e., the arcs between nodes, is determined with the help of expert knowledge and machine learning in the fourth phase which is presented in Section 4.4. Learning algorithms are also used to determine the CPTs of the DBN. The fifth and final phase of the construction is the validation of the DBN metamodel. The validation is discussed in Section 4.5.

During the construction, it can become apparent that the data produced by a previous phase is inadequate. In such a situation, it is necessary to return to this earlier phase and redo it. The most common backtracking is from the validation phase back to the selection of time instants or the design of experiment but a move from any phase back to any previous phase can be justified.

4.1 Simulation data

DBN metamodels are constructed based on simulation data. If the data is created in connection with the construction of the metamodel, the simulation model is applied in a manner that creates useful data without the need to run an excessive number of replications. Regardless of how the simulation

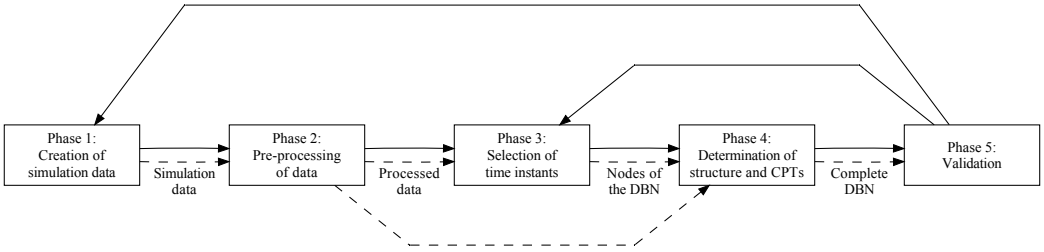


Figure 5: Phases of the construction of a DBN metamodel. The continuous lines represent the progression of the construction and the dashed lines the transfer of data between phases. Processed data is also supplied to the validation phase if a section of the simulation data is reserved for validation.

data is created, it is then stored in a format that contains all the required information.

4.1.1 Design of experiment

The number of simulation replications to be performed as well as the values of input variables in each of these are determined in the design of experiment phase. Choosing the number of simulation replications to perform is an important decision not just in the construction of DBN metamodels but in simulation analysis in general [20]. A guideline for the number of replications needed to achieve a specific level of accuracy in the metamodel can be calculated by assuming that the metamodel is approximately as accurate as the simulation data that it is based upon.

The metamodel is based on the relative frequencies in the simulation data, so the variance of these frequencies also describe the accuracy of the resulting metamodel. Denote the probability of variable x_k obtaining value j at time instant t when running the simulation model as P. There are N replications of the simulation model, and $x_k(t)$ obtains the value j in n of those. This means that n is a random variable. The relative frequency estimator for P is $p = n/N$. The notation $p_k^j(t)$ is used when j, k, and t need to be explicitly stated. The expected value of p is

$$\mathrm{E}[p] = \mathrm{E}[n/N] = \mathrm{E}[n]/N = NP/N = P, \tag{1}$$

and the variance of p is

$$\mathrm{Var}[p] = \mathrm{Var}[n/N] = \mathrm{Var}[n]/N^2 = NP(1-P)/N^2 = P(1-P)/N. \tag{2}$$

In the utilization of DBN metamodels, conditional probability estimates are commonly used. They can also be estimated from relative frequencies.

The procedure is identical to the one above for marginal probabilities, except that only those replications where the conditions hold are useful. Consider the previous probability with the added condition of $z_{k'} = j'$. The conditional probability in question is denoted here as P'. $z_{k'} = j'$ is observed N' times and both $x_k(t) = j$ and $z_{k'} = j'$ simultaneously n' times. Since N' is not known in advance, it must be treated as a random variable. The relative frequency estimate for P' is $p' = n'/N'$. Assuming that $N' > 0$, the expected value of p' does not depend on the probability distribution of N' and is equal to P'. The variance of p' does depend on the probability distribution of N' and is equal to $P'(1 - P')\mathrm{E}[1/N']$. Since the purpose is merely to approximate the number of necessary replications, a further simplification of $\mathrm{E}[1/N'] \approx 1/\mathrm{E}[N'] = 1/(QN)$ is made where Q is the probability of $z_{k'}$ obtaining the value j' in a replication of the simulation model.

The variance is greatest when $P' = 1/2$ which gives the upper bound $\mathrm{Var}[p'] \leq 1/(4QN)$. N can be selected such that the variance for all conditional probabilities of interest is sufficiently small if the corresponding probabilities Q are estimated prior to the simulation. The selection of the sample size is discussed further in [39].

The DBN metamodel under consideration can only handle discrete variables. All variables are discretized in the second phase, but the discretization of input variables needs to be considered in connection with the generation of simulation data. The metamodel should be able to produce useful results with any plausible combination of values for the input variables. Because of this, it is recommended to use every possible combination of discretized values of the inputs when running the simulation model. The exact distribution of simulation replications between different input values is judged on a case by case basis.

DBN metamodels can only be constructed for simulation models that describe the state of the system during a fixed time interval. Now, the initial and terminating time of the simulation model are denoted by t_0 and t_f, respectively. If the simulation model does not have a fixed terminating time, it can be extended up to some predetermined ending time by keeping all state variables unchanged after the original terminating time. Optionally, a boolean state variable indicating whether or not the original final time has been exceeded is added to the model.

4.1.2 Storage of simulation data

In order to automate the construction of DBN metamodels, it is necessary for the simulation data to be stored in a predefined format. There is no common standard format for storing DES data, so the format to be used needs to be

defined explicitly. Regardless of what that format is, the simulation data needs to at least include the values of input and output variables, the initial values of state variables, and the timing and value changes associated with each event. This information is stored for every replication of the simulation. Additional information can also be included, such as the names of the variables.

4.2 Definition of variables

The metamodel is constructed based on data produced by a simulation model. Before the DBN can be constructed, the data needs to be preprocessed. This consists of selecting, categorizing, and discretizing the variables.

4.2.1 Selection of variables

While a DES model should include every variable that is relevant to the operation of the underlying system, the variables of a corresponding DBN metamodel can contain any subset of the DES model's variables. The selection can therefore be made purely on the grounds of which variables are going to be used in the analysis that is conducted with the help of the completed metamodel.

As mentioned before, the variables are categorized into input variables denoted by u_k, state variables denoted by $x_k(t)$, and output variables denoted by z_k. The state variables are the only time-dependent variables of the model and can therefore be automatically identified. A variables must be a state variable if its value changes even once in just one replication of the simulation model. Input and output variables cannot be automatically distinguished from each other unless this information is explicitly included in the format of the simulation data. It is therefore necessary to make this distinction manually.

The input variables are variables whose values are determined at the start of a simulation and remain unchanged. They can, e.g., be parameters specifying a setting of the model, such as the arrival intensity of customers in a queuing system. Alternatively, inputs can also represent decisions whose consequences are studied, such as the number of servers to use.

The state variables are time-dependent variables that may change value during the simulation. After some further preprocessing described in the following section, the recorded values of the state variables form the basis of the DBN metamodel. The state variables are subject to the selection of time instants for the nodes in the DBN in the third phase of the construction of the metamodel.

The use of multiple time scales makes it easier to take into consideration the individual characteristics of each variable when creating the metamodel. The use of multiple time scales in the metamodel also has an impact on the choice of variables. A state variable can be modeled less accurately with fewer nodes if it has only little impact on the system as a whole. This does not complicate the metamodel much, so the threshold to include less important state variables is lower when using multiple time scales.

4.2.2 Discretization of variables

It is necessary to limit the number of possible values for each variable in order to create a useful DBN metamodel. Continuous variables need to be discretized and discrete variables are also categorized into a smaller number of bins when necessary. The fewer values there are for each variable, the more computationally efficient the metamodel becomes. On the other hand, some information is always lost in the discretization of variables. The chosen discretization is therefore a compromise between accuracy and efficiency.

The details of the discretization, such as the exact locations of the edges between bins that the values are placed in, should ideally be determined on a case by case basis since the interpretation of the values of the variables requires knowledge of the system that is being analyzed. Because of this, it is recommended to do the discretization separately, prior to the actual metamodeling process. This is not always possible, so it is necessary to also allow the discretization to be done in an automated manner in connection with the construction of the metamodel. An algorithm for accomplishing this is presented here. It is based on the k-means clustering algorithm [18]. k-means is an algorithm that is commonly used in clustering analysis and the discretization can be seen as a special case of clustering where the data is one dimensional.

The k-means algorithm divides a set of observations into clusters such that each point is in the cluster whose mean point it is closest to. The problem is usually solved by iterative heuristic algorithms [18]. When the data is one-dimensional, a more systematic approach is possible. There can be several clusterings of the data points for which each point is assigned to the cluster whose mean point is closest. One of them is the clustering that minimizes the expression

$$\sum_{i \in I} \left(v_i - c(v_i)\right)^2, \tag{3}$$

where v_i is the value of an observation, $c(v_i)$ the mean of the cluster that it is assigned to and I an appropriate set of indexes [12].

With one-dimensional data, each cluster consists of the observations falling within an interval with no overlapping between clusters. Because of this, the clustering can be described by determining the break points where one cluster ends and the next one begins. This makes the problem suitable for dynamic programming (see, e.g., [4]) similarly to the selection of the time instants for the nodes in the DBN which is presented in Section 4.3.3.

The challenge with this approach is that the number of observations can be too large for dynamic programming to solve the problem efficiently. This can be bypassed by first rounding the values slightly. This reduces the number of unique values enough to make dynamic programming viable without significant loss of accuracy. A histogram is created of the rounded values and is in turn used in the k-means algorithm.

The expression (3) can be replaced by

$$\sum_{i \in I'} F_i \left(y_i - c(y_i) \right)^2, \tag{4}$$

where y_i is a rounded value, F_i the number of observations rounded to it, and I' the appropriate set of indexes. The sum is calculated for one cluster at a time and then added together. Given an upper and lower bound for the values assigned to a cluster, its mean point can be calculated. With the help of that, the terms in (4) assigned to that cluster are evaluated.

A dynamic programming algorithm is then applied. It iterates over the unique rounded values to identify the optimal clustering breakpoints for any number of clusters. Once the number of clusters has been chosen, each observation is replaced by the mean of its cluster. This means that the number of unique values for the variable is the same as the number of clusters, i.e., the values have been discretized.

4.3 Selection of time instants for nodes

The time instants for the DBN are picked separately for each state variable, rather than jointly for the entire metamodel, in this thesis. Selecting the time instants is an optimization problem that has previously been solved using genetic algorithms [39]. In connection with this thesis, an alternative approach based on dynamic programming is developed for solving this problem. Since the same procedure is applied to each state variable one at a time, indexes identifying the variable are omitted in order to simplify the notation.

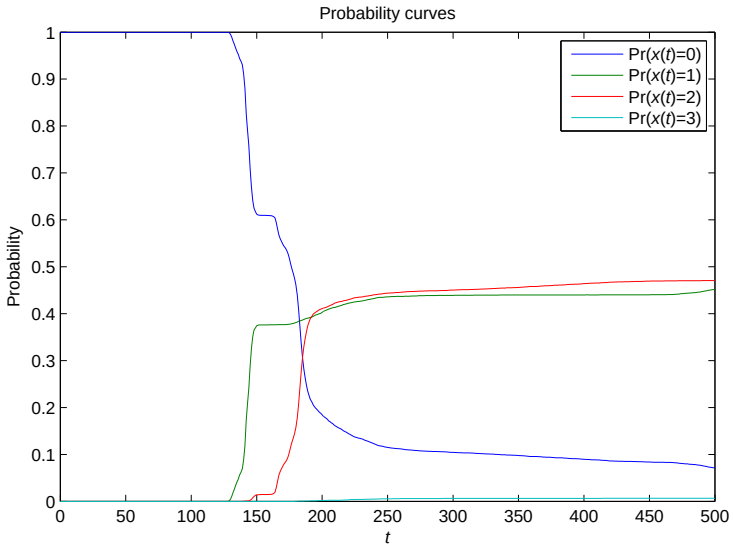


Figure 6: Example of probability curves.

4.3.1 Probability curves

For the state variable x, a discrete set of time instants $T = \{t_0, t_1, t_2, \ldots, t_f\}$ is used in the construction of the DBN instead of the continuous time interval $[t_0, t_f]$. In order to select the most suitable time instants, an initial discretization $T^* = \{t_0, t_0 + \delta, t_0 + 2\delta, \ldots, t_0 + m\delta\}$ is first created, where m determines the density of the discretization and $\delta = (t_f - t_0)/m$. T is then chosen among the subsets of T^*. Alternatively, T can be selected from the entire interval $[t_0, t_f]$ as has been done previously, but that approach is incompatible with the optimization algorithms presented here and computationally inefficient.

The characteristics of the variable in question should be considered when choosing the value of m. Greater values are needed if the probability distribution of the variable changes rapidly. This comes at a cost, as a large m increases the number of candidates for T and therefore slows down the selection. Increasing the number of time instants by too much renders the use of the dynamic programming algorithm discussed in Section 4.3.3 impractical.

The marginal probability distributions of the state variable at the time instants T^* are estimated by using the relative frequencies gathered from the simulation data. The relative frequency of obtaining value j at time instant t is denoted as $p^j(t)$. The estimates can be visualized as time dependent probability curves for each state variable, as is demonstrated in Fig. 6.

4.3.2 Optimization problem

The probability curves are the basis for the selection of time instants for the nodes. The objective is to create an estimate for the probability curves based only on the probability distributions at the time instants T. Linear interpolation between consecutive time instants is applied to estimate the probability distributions at time instants not included in T. The estimate for obtaining the value j at time instant $t \notin T$ is

$$\hat{p}^j(t) = \frac{(t_+ - t)p^j(t_-) + (t - t_-)p^j(t_+)}{t_+ - t_-}, \tag{5}$$

where $t_- = \max\{t^* \in T | t^* < t\}$ and $t_+ = \min\{t^* \in T | t^* > t\}$. In other words, the estimate is piecewise linear with the additional restriction that the break points are on the actual curves. The interpolation is discussed in more detail in Section 5.2.

The importance of the selection of time instants for the nodes is demonstrated in Fig. 7. In 7a, the time instants are spread out evenly over the time interval. As a result, the crucial part of the simulation when the state changes often is inadequately covered by the interpolation. In 7b, the same number of nodes are optimally placed and the interpolation fits the original probability curves much better.

Selecting the time instants for the nodes consists of choosing the number of nodes and the time instant for each node. The latter forms a complex optimization problem. Choosing the number of nodes can be treated as a separate decision or it can be included in the optimization problem in the form of a penalty for large node counts.

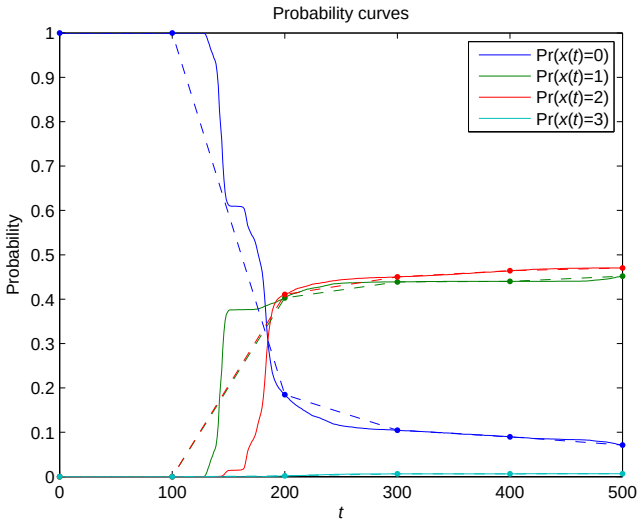
The objective function of the optimization problem measures the error of the piecewise linear approximation of the probability curves. There are several such measures, including the maximum absolute estimate error

$$g(T) = \max_{j \in X} \max_{t \in T^*} |\hat{p}^j(t) - p^j(t)|, \tag{6}$$

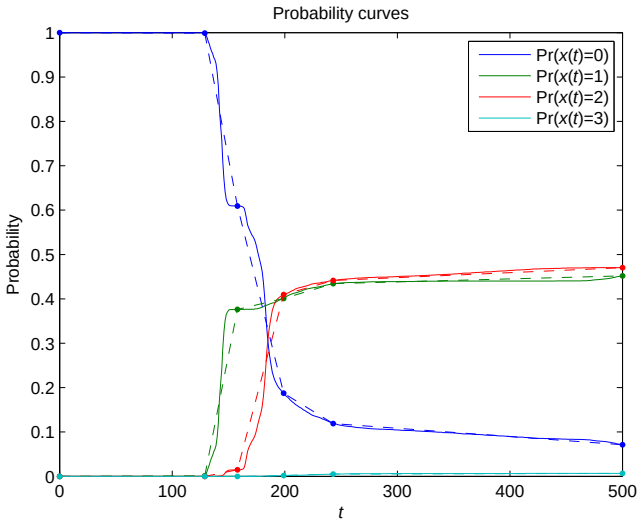
where X is the set of possible discretized values for x, and the root mean squares of the estimate error (RMSE)

$$g(T) = \sqrt{\frac{\sum_{j \in X} \sum_{t \in T^*} \left(\hat{p}^j(t) - p^j(t)\right)^2}{|X||T^*|}}. \tag{7}$$

The sum of the absolute estimate error, however, is not recommended for this problem, since the function should be sensitive to spikes in the probability curves. In this thesis, RMSE or the equivalent sum of squares is used as the



(a) Evenly spaced time instants.



(b) Optimally spaced time instants.

Figure 7: Comparison of evenly spaced time instants and optimally spaced time instants. Continuous lines represent the probability curves derived from data while the dashed lines represent estimates based on interpolation.

error measurement, but similar results hold for the maximum absolute error as well.

The error is calculated separately for sets of time instants, in particular the time instants between two consecutive instants in T, and then aggregated to evaluate $g(T)$. This simplifies the solving of the problem, as partial results from evaluating one candidate for T can be used in the evaluation of other candidates.

The choice of the number of time instants to include is dependent on the probability curves and the accuracy requirement for the variable in question. More nodes are needed if the curves are particularly non-linear. It is also worth considering that while the time instants are selected based on the marginal distributions, they are also applied to conditional probability distributions. Because of this, large gaps between time instants in T can be problematic even if the time evolution of the marginal distribution is close to linear in the gap. Sometimes, it is impossible to know that too few nodes have been chosen without first analyzing the resulting DBN. Because of this, it may be necessary to repeat the construction of the metamodel with different selections of time instants until a satisfactory metamodel is found.

4.3.3 Dynamic programming

Dynamic programming is an algorithm used for solving dynamic optimization problems [4]. The main principle of the algorithm is that a problem is solved with the help of subproblems which in turn are solved one by one, starting with the smallest. The solutions to smaller subproblems are used to solve larger ones until the main problem has been solved [4].

Dynamic programming has not previously been used in the construction of DBN metamodels but it is an efficient way of finding the global minimum for the optimization problem. The algorithm also has the advantage of simultaneously solving the optimal selections of time instants for multiple numbers of nodes at once. The procedure for finding the optimal time instants is similar to the fitting of line segments to a continuous curve presented in [16].

The algorithm is now described with the sum of squares as the objective function. The initial step is to choose the maximum number of time instants allowed. The problem is then solved for all numbers of time instants up to this maximum. The running time of the algorithm is only slightly affected by this number as long as it is within reason, so it is recommended to use a higher limit than what is expected to be necessary.

The notation $d(a, b)$ is used for the estimation error of a linear segment

from $t_0 + a\delta$ to $t_0 + b\delta$. It is calculated as

$$d(a,b) = \sum_{j \in X} \sum_{i=a}^{b} \left(p^j(t_0 + i\delta) - \frac{(b-i)p^j(t_0 + a\delta) + (i-a)p^j(t_0 + b\delta)}{b-a} \right)^2. \tag{8}$$

The minimum estimation error when covering the set $\{t_0, ..., t_0 + b\delta\}$ with l linear segments is denoted by $f_l(b)$. The optimal value of the objective function can be calculated iteratively. Set $f_l(0) = 0$ for all values of l and $f_1(b) = d(0, b)$ for all b. All other values of $f_l(b)$ can then be calculated with the help of the equation

$$f_l(b) = \min_{0 \leq i < b} f_{l-1}(i) + d(i, b). \tag{9}$$

In order to find the optimal time instants, it is necessary to also keep a record of which time instants lead to the minimum values. This is denoted as $S_l(b)$ and is calculated by

$$S_l(b) = \arg\min_{0 \leq i < b} f_{l-1}(i) + d(i, b). \tag{10}$$

The optimal time instants are looked up once the algorithm has terminated. Suppose that time instants for n nodes are sought, including t_0 and t_f. They are, in reverse order, $t_f, t_0 + S_{n-1}(m)\delta, t_0 + S_{n-2}(S_{n-1}(m))\delta, t_0 + S_{n-3}(S_{n-2}(S_{n-1}(m)))\delta, \ldots, t_0$. The values of $d(a, b)$ should be calculated for one value of b at a time, so that each $f_l(b)$ needs to be evaluated just once and at most m values of $d(a, b)$ need to be stored in memory at a time.

Since every linear segment between time instants is checked, the number of such segments is proportional to the square of m. It turns out that this is also the dominating factor in the running time of the dynamic programming algorithm. Because the running time increases significantly when m is increased, other alternatives for solving the optimization problem are also considered. It is possible to reduce the effect of a large m by taking prior information about the model into consideration. If a maximum acceptable gap between consecutive time instants in T is known, then automatically eliminating all larger segments causes an improvement to the running time of the algorithm.

4.3.4 Genetic algorithm

Evolutionary algorithms (see, e.g., [45]) are a family of stochastic heuristic optimization algorithms that are based on imitating aspects of natural evolution. The basic premise of such algorithms is the concept of a population of

solution candidates, as opposed to the approach employed by most optimization algorithms of attempting to improve a single solution until it reaches the optimum. The population is expected to converge towards the optimum solution. Evolutionary algorithms consist of three main steps: individual solutions are selected for reproduction, new individual solutions are created, and the new solutions replace old ones to create a new generation. This process is repeated, starting from an initial population, until a terminating criterion is fulfilled.

Genetic algorithms (GAs) [45] are a form of evolutionary algorithms. In GAs, individuals are selected for reproduction based on the values of the objective function. New individuals are created by crossing over two old ones. Individuals are also modified by mutation. In GAs, the old generation is completely replaced by the new one, although the best solution candidate found is stored separately for reference.

A heuristic approach, such as a GA, is needed if solving the optimization problem in the construction of a DBN metamodel by dynamic programming takes too long. Compared to dynamic programming, the GA has some clear disadvantages. It usually only finds sub-optimal solutions and only provides one solution at a time, while the dynamic programming algorithm solves the problem for all numbers of nodes at once. The running time of the GA, however, isn't heavily dependent on the number of potential time instants. The exact nature of this dependence is difficult to verify experimentally, because the running time of the GA is not deterministic. The number of time instants at which the GA starts to outperform dynamic programming depends on the parameters of the genetic algorithm.

The GA takes advantage of the same property of the objective function as the dynamic programming algorithm – it evaluates each gap between time instants of consecutive nodes separately. In dynamic programming, this evaluation is carried out for every possible pair of time instants. When running the genetic algorithm, only the pairs of time instants that the algorithm stumbles upon are evaluated. To make the algorithm efficient, it is crucial to store the estimation error of all pairs of time instants that have already been evaluated to avoid what would otherwise be a large amount of redundant calculations.

In the GA, the number of nodes needs to be included in the optimization problem. Each additional node increases the value of the error function by a set amount. The inclusion of nodes that don't improve the estimate sufficiently is thus avoided. Even though the number of nodes cannot be explicitly chosen, the number can still be forced into a suitable range by adjusting the size of the penalty for additional nodes.

Each individual in the solution population is a list of bits, one for each

possible time instant. The solution consists of the time instants for which the corresponding bit obtains the value 1. Since the selection of time instants must always include the first and the last time instant, the first and the last bit always have the value 1.

The initial population for the algorithm consists of randomly generated individuals. The bits of each solution are all assigned randomly and independently. The probability of a bit initially obtaining the value 1 need not be correctly estimated, because the solution population will converge from any reasonable initial population.

The fitness value of a solution is the inverse of the error function with the node penalty included in the error, i.e., a higher fitness is better. When creating a new generation, two parents are selected randomly from the previous one. The probability of each individual being selected is relative to its fitness.

Crossover is done with a cutoff point selected randomly among the time instants. The new solution consists of the bits prior to the cutoff point from the first parent and the bits after it from the second parent. Mutation is then performed on the new population. Each individual has a fixed probability of being subject to mutation. For those individuals, a subset of bits, excluding the first and last one, are chosen randomly. These bits are then given random values. Since the clear majority of bits have the value 0, this is considered when assigning a new value. The probability of obtaining the value 1 must be of the same magnitude as the relative frequency of ones in the current population.

The genetic algorithm includes several parameters, such as the size of the population, the number of generations to be computed, and the probability of a bit being chosen for mutation. None of these have any obvious default value, and the tuning of the parameters is a typically solved by trial and error.

4.3.5 Other alternatives

There are also other options for the selection the time instants. Heuristics other than genetic algorithms can be used to optimize the selection. Since such algorithms are only needed when the dynamic programming algorithms is too slow and settling for a good sub-optimal solution is necessary, there is not much practical benefit in switching to, e.g., simulated annealing. All such heuristics are likely to eventually find decent solutions.

A more interesting alternative is to let an expert be directly involved in the optimization, e.g., by manually giving a starting solution that a heuristic algorithm then attempts to fine tune. This could potentially decrease the amount of computing effort required considerably but would rely heavily on the judgment of the person involved. This approach would reduce the level of

automation but since a person is already required to make the final choice as to which solution candidate to accept, increasing his responsibilities would not necessarily slow down the process excessively.

One problem in the automated selection of time instants is situations where it is known that the value of a variable often changes at a certain time, but the probability curves are still linear around that time instant. In such situations, the issue is not with the algorithms used to solve the optimization problem, but the formulation of the problem itself. One possible solution is to include manually selected time instants in the DBN metamodel in addition to the time instants selected based on the optimization problem.

4.4 Determination of the DBN

With N nodes, there are $3^{N(N-1)/2}$ different possible directed graphs. Such a large number of alternatives cannot all be evaluated individually. The collection of potential graphs is first cut down with the help of expert knowledge and by ruling out all graphs containing arcs going backwards in time. After that, machine learning is used to find a suitable graph among the remaining possibilities.

Even though the estimation of CPTs in the DBN is performed after the graph has been determined, the two sections are presented here in reverse order. This is because some algorithms for determining the graph use the estimation of CPTs as a subroutine.

4.4.1 Estimation of conditional probability tables

When the graph has been determined, and therefore the parents of each node, CPTs are estimated. This estimation is carried out similarly for each combination of values for the parents of every node, so it is sufficient to consider just one such instance. Whether this node represents a state variable at a particular time instant or an output variable is irrelevant when it comes to estimating the CPTs. The estimation of the CPTs is one aspect of the construction of the metamodel that is always conducted in an automated manner directly from the simulation data.

To properly motivate the choice of estimate, the likelihood of each distribution being correct is considered, i.e., a probability distribution of probability distributions. To avoid confusion, the distributions in the CPTs are referred to as relative frequencies, since they are based on the relative frequencies in the simulation data. Further mentions of probability distribution in this section refer to the distribution of relative frequencies.

In order to model the probability distribution of relative frequencies in a meaningful way, a parametrized distribution is required. The Dirichlet distribution, denoted by $D(\alpha)$, turns out to be a natural choice [33]. The parameter α is a vector of positive elements, one for each value of the variable in question. The elements can be interpreted as the number of prior observations of each value. If the prior distribution of relative frequencies is $D(\alpha_1, ..., \alpha_k, ..., \alpha_n)$ and then an observation of $x = k$ is made, the posterior distribution is $D(\alpha_1, ..., \alpha_k + 1, ..., \alpha_n)$. This makes the Dirichlet distribution particularly well suited for modeling the probability distributions of relative frequencies.

In order to use the Dirichlet distribution, initial values must be provided for the parameter α. If $\alpha = (1, ..., 1)$ is used as the initial values, then all relative frequencies are considered equally likely. Larger values for elements of α can be used to represent prior information about the relative frequencies. Values of less then 1 can also be used in order to minimize the estimated relative frequency for values that are never observed. This might be the case if it is suspected that the probability of some value is actually 0.

With the initial values of α fixed, the simulation data is screened. For each replication, the values of the parent nodes are looked up and the relevant column in the CPT and the corresponding α is identified. The element in α corresponding to the observed value of the variable is increased by 1. When this has been carried out for every replication, the resulting Dirichlet distribution takes into consideration both the initial parameter values as well as all the simulation data.

The CPTs consist of relative frequency estimates based on the Dirichlet distributions. One obvious choice of estimate is the maximum posterior likelihood estimate, i.e., the mode of the probability distribution. The relative frequency of the k:th value at the mode of the Dirichlet distribution is

$$\frac{\alpha_k - 1}{\sum_{i=1}^{n}(\alpha_i - 1)} \tag{11}$$

which is problematic because it is only defined if the value of each element in α is at least 1 and at least one element has a value greater than 1. Another option for the estimate is the expected value of the distribution. The relative frequency of the k:th value at the expected value of the Dirichlet distribution is

$$\frac{\alpha_k}{\sum_{i=1}^{n}\alpha_i}. \tag{12}$$

This is always defined since the elements of α are positive. The expected value is identical to the mode but with different initial parameter values. The

mean is therefore more convenient to use as the estimate and this is taken into consideration when selecting the initial values for α.

Apart from the estimate of the relative frequencies, the Dirichlet distribution also provides the means to calculate the probability of a given data set originating from a Bayesian network with given prior relative frequencies. This can be done iteratively, by calculating the probability of one sample, updating the relative frequencies and moving on to the next sample. The ordering of the samples does not affect the final probability in any way. The probability can also be calculated directly using the formula

$$\prod_{i=1}^{n}\prod_{j=1}^{q_i}\frac{\Gamma(N_{ij})}{\Gamma(N_{ij}+M_{ij})}\prod_{k=1}^{r_i}\frac{\Gamma(a_{ijk}+s_{ijk})}{\Gamma(a_{ijk})}, \tag{13}$$

where n is the number of variables, q_i the number of combinations of values for variable i's parents and r_i the number of values for variable i. a_{ijk} is the prior parameter value for variable i, values j for i's parents and value k for i, and $N_{ij} = \sum_{k=1}^{r_i} a_{ijk}$. s_{ijk} is the number of observations, with similar indexing as for a_{ijk}, and $M_{ij} = \sum_{k=1}^{r_i} s_{ijk}$. Γ is the gamma function.

The parameters a_{ijk} can be assigned values according to different principles. One option is to simply set $a_{ijk} = 1$ which is called the K2 prior[23]. Another alternative is $a_{ijk} = a_0/(q_i r_i)$ which is called the BDeu prior[23]. When using BDeu priors, the value of the parameter a_0 can be chosen freely, but $a_0 = 1$ is a good default value.

If the sample size is large compared to the prior values of α, the resulting estimates for the CPTs are close to the relative frequencies in the simulation data and the exact prior values become insignificant. The CPTs therefore provide probability distributions for the variables in the DBN that are consistent with the simulation data. The convergence towards the relative frequencies in the simulation data also justifies the formulation of the estimate in Section 4.1.1 when calculating the required sample size.

The probability distributions of input variables cannot be determined based on the simulation data unless their values have been generated according to their probability distribution. The probability distributions instead originate from some external source. The most natural origin is some form of expert knowledge. Knowledge of the underlying system is used to construct a distribution that is judged to be most accurate. The purpose of the simulation is not always to merely imitate a system. The objective might instead be to investigate the effect of various inputs on the system. In such a situation, the inputs of the simulation metamodel are fixed to correspond to the inputs that are of interest even if they are less likely to occur.

In the algorithms that are used for determining the graph, input variables

are treated identically to output variable variables and their distributions are estimated from the simulation data alone. These distributions are not necessarily correct, but are used in the validation of the metamodel. When the metamodel is otherwise completed, the probability distributions of the input variables are modified from the relative frequencies in the simulation data to what expert knowledge dictates.

4.4.2 Estimation of the graph

The graph of the DBN is estimated using both expert knowledge and learning algorithms. Expert knowledge is first introduced automatically by forcing the DBN to be chronologically consistent. No arc can lead to an input variable and arcs between nodes corresponding to state variables can only go in one direction unless the nodes correspond to the same time instant.

Expert knowledge is then introduced manually to determine the graph more closely. Arcs are introduced if there is judged to be a direct causal link between two nodes. It is necessary to make sure that the graph does not become cyclic when doing this. Similarly, arcs are also explicitly forbidden if the nodes are judged to be independent. It is not recommended, however, to forbid arcs simply because their direction is considered to be opposite to causality, as such unrealistic arcs can still improve the accuracy of the DBN.

It is unlikely that enough expert information is available for the graph to be determined explicitly. Learning algorithms are used to finalize the graph. There are several algorithms for accomplishing this. Two of them are presented here. Including some learning algorithm in estimating the graph is essential in order to automate the construction of the metamodel. Unless the system that is being modeled is exceptionally simple, it is both difficult and time consuming to estimate the graph manually.

The first learning algorithm to be presented is called PC [44] and is based on independence tests. It is a two staged algorithm that first creates an undirected graph and then attempts to determine the direction of the arcs. The algorithm begins by creating a complete undirected graph covering all of the nodes. The next step is to remove arcs between nodes that are conditionally independent. The arcs are removed iteratively with regards to the number of nodes in the condition set. That is, nodes are first checked for unconditional independence, then for conditional independence given the value of some single node, and so on. This procedure, however, would be too time consuming as such. To counter this, the PC algorithm takes advantage of the fact that the graph is becoming increasingly sparse. Conditional independencies are no longer searched for if the condition set becomes larger than the set of remaining neighbors for any node.

When no more arcs can be removed, the algorithm moves on to determining the direction of the remaining ones. The algorithm looks for triplets of nodes A, B, C, such that A and B are adjacent, as are B and C, but A and C are not. Furthermore, B must not be in the condition set that eliminated the arc between A and C. The two arcs are oriented to point towards B if this is the case. Finally, logical reasoning is used to orient some more arcs. In a situation as described above, but with B in the condition set, there cannot be arcs directed to B from both A and C. Since Bayesian networks are directed acyclic graphs, no directed cycles can be allowed to form. The algorithm usually does not result in a graph where all of the arcs directed. It instead gives an equivalence class of graphs, all of which can correspond to the same joint distribution of the variables.

A problem with the PC algorithm is determining the independencies. The simulation data set is used for this, so no definite answers exist. Statistical methods such as cross entropy, also known as mutual information, [9] and the χ^2-test [17], can be used. There is no guarantee, though, that the selected methods result in correct conclusions.

An alternative way to determining the graph is to use score functions. Each potential graph is given a score based on how compatible it is with the data and the goal is find the graph with the best score. One such method is called the Greedy Thick Thinning algorithm (GTT). The probability in Eq. (13) is the scoring function used by GTT.

Going through all potential graphs is not feasible due to the large number of possibilities. Because of this, it is necessary to potentially settle for a suboptimal solution. GTT searches for a local optimum with a greedy heuristic algorithm that consists of two stages. The algorithm begins with a graph with no arcs. Arcs are added one at a time such that each new arc increases the value of the score function by a maximal amount. When arcs can no longer be added without decreasing the value of the score function, the algorithm moves on to the second stage. In this stage, arcs are removed from the graph. Arcs are removed one at a time so that each step maximizes the score function. When it is no longer possible to remove arcs without decreasing the value of the score function, the algorithm terminates.

Similar but more complex heuristic search algorithms also exist. Rather than dividing the algorithm into two stages, it is possible to use just one stage where each step can be the addition of an arc, the removal of an arc, or the reversal of the orientation of an arc. With this approach, the initial graph does not need to be empty. One further enhancement is to use several different starting graphs to get a number of candidate solutions and then pick the best one [19].

The two approaches to estimating the graph mentioned above, testing

for conditional independence with PC and maximizing the score function with GTT, are of similar level of accuracy. One important difference is that score maximizing algorithms such as GTT produce a graph where all arcs are always oriented. From an automatization point of view, this is an important advantage and the main reason why GTT is more useful in the automated construction of DBN metamodels. GTT is therefore used in the rest of this thesis.

4.5 Validation

A DBN metamodel is validated to verify that it is consistent with the simulation model under consideration. A thorough validation of the metamodel would compare it to the underlying system as well [26]. In this thesis, however, no assumptions are made regarding the possibility of conducting experiments on the system. The validation of the metamodel is therefore restricted to comparisons with the simulation model.

The metamodel is constructed from simulation data generated by the simulation model. There are two main sources of possible inaccuracies in the metamodel. Firstly, the generated data might not describe the simulation model accurately enough if an insufficient number of simulation replications is performed. Secondly, the metamodel might not be consistent with the simulation data if it is not constructed in sufficient detail.

These errors are looked for in an iterative process where conditional probabilities are examined in the form of different scenarios where specific conditions are set. Which scenarios to examine is a design choice and depends on the intended use of the metamodel. For each scenario, the sample size is checked and the probabilities produced by the metamodel are compared to the simulation data. The accuracy of the metamodel is measured in both regards, but judgment is required to determine whether the result is acceptable or not. This is repeated for different scenarios until one is found where the metamodel is invalid, in which case a previous phase of the construction needs to be repeated. Recall that the phases are described in Fig. 5. The metamodel is deemed to be valid if it isn't invalid in any relevant scenario. A flowchart of the validation process is presented in Fig. 8.

4.5.1 Sufficiency of simulation data

As discussed in Section 4.1, the required number of simulation replications depends on the intended application of the metamodel. The variances of relative frequencies in the data depend on the sample size and the corresponding true probabilities. The variance is the greatest when the probability in question is

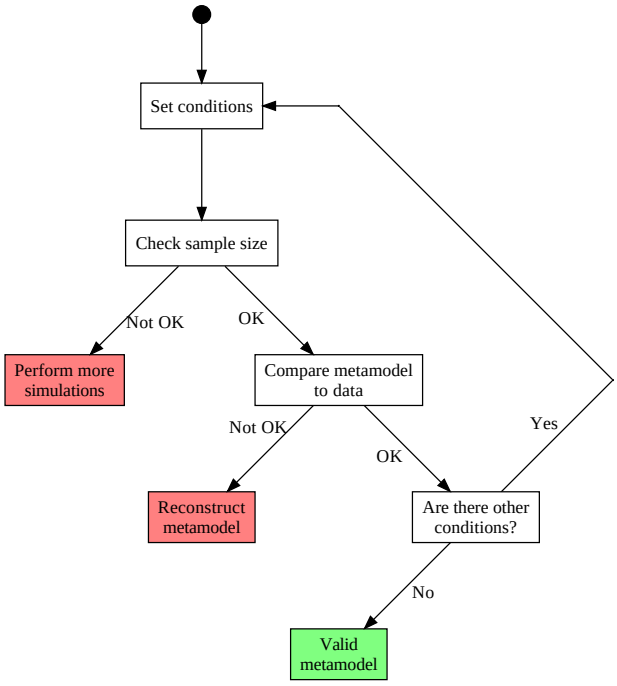


Figure 8: The iterative validation process.

1/2. This provides an upper limit for the variance when the sample size is known.

The simulation data is validated by setting conditions on the values of variables. The data is screened for those replications where the conditions hold. The number of such replications is the sample size that determines the maximum variance for the conditional relative frequencies. If this variance is judged to be too great for the conditional probability distribution in question, then more simulation data is needed and the metamodel construction returns to the first phase.

4.5.2 Consistency between simulation data and metamodel

DBN metamodels produce conditional probability distributions. The same distributions can also be estimated directly from the relative frequencies in the simulation data. Assuming that the sample size is large enough, the relative frequencies correspond closely to the true probability distributions of the simulation model. The metamodel's probability distributions must be similar to the relative frequencies in the simulation data if it accurately describes the simulation model.

The comparison between the metamodel's probability distribution and

the relative frequencies in the data is performed for one variable at a time. For input and output variables, the probability of each value is subtracted from the corresponding relative frequency and the resulting differences are aggregated into a single measurement using RMS. The metamodel is invalid if the difference is judged to be unacceptably large.

For state variables, the time-evolution of the variables must be considered. The time interval is first discretized and then the differences in probabilities at each time instant are calculated. The differences are then collated over all time instants, again using RMS, to get a single value representing the total error for every variable. If a measurement other than RMSE was used to select the time instants in the third phase, it is recommended to use the same measurement in the validation as well. It may also be useful to check the differences between the probability distribution and the relative frequencies for individual variable values and time instants separately. The situations where the deviation is largest can thus be identified.

The metamodel needs to be reconstructed if the errors are judged to be unacceptable. This is most conveniently carried out by returning to the third phase and increasing the number of selected time instants.

4.5.3 Comparison to independent validation data

A DBN metamodel can also be validated by comparing probability distributions provided by it to relative frequencies in a separate set of simulation data, i.e., the independent validation data. Due to the limited sample size of the validation data, the relative frequencies are not identical to the probability distribution in the simulation model. The difference between these two is independent from any differences between the simulation model and the metamodel, because the validation data is generated independently. As a result, the probability distributions provided by the metamodel are on average closer to the probability distributions in the simulation model than they are to the relative frequencies in the validation data. The connection between the models and the data sets is illustrated in Fig. 9. If the metamodel is deemed to be consistent with the validation data, then it is likely to also be consistent with the simulation model and therefore valid.

Comparison to independent validation data differs from the other validation techniques mentioned in this section in several ways. While other comparisons focus on a particular part of the construction, comparison to independent data concerns the DBN metamodel as a whole. All phases of the construction are likely to be successful if the resulting metamodel appears valid. On the other hand, the comparison to independent data does not help in identifying the cause of inaccuracies when the metamodel is not valid.

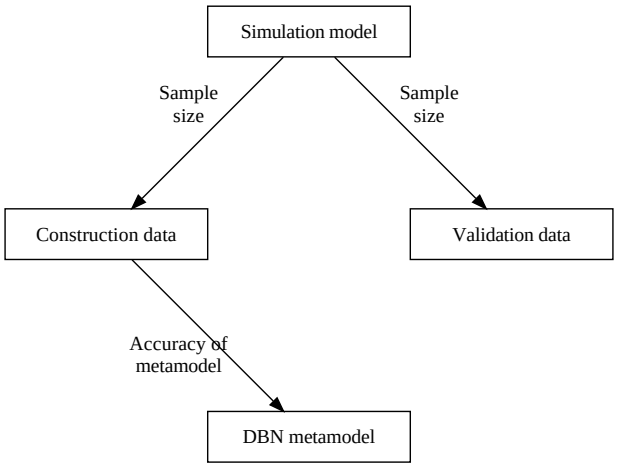


Figure 9: Connections between the simulation model, the DBN metamodel, the simulation data, and the independent validation data. The cause for inaccuracies introduced by each connection is also given.

Furthermore, the sample size of the independent data is often smaller than the sample size of the data that the metamodel is constructed with. Because of this, there is a high probability that discrepancies between the metamodel and the validation data are caused by the sample size of the validation data. There is also a philosophical difference between comparison to independent data and other validation means as the comparison to independent data assesses the applicability of the DBN metamodel.

Automation is incorporated in all three validation means presented in this section. The measures of accuracy in each case are automatically calculated and presented in the form of a single number. This speeds up the process of validating the metamodel considerably, which in turn allows a more thorough examination of the its validity. By identifying problems in the sample size of the simulation and the construction of the metamodel separately, it is possible to determine which phase of the automated construction needs to be repeated in order to construct a new improved metamodel.

5 Utilization of DBN metamodels

The utilization of DBN metamodels is presented in this section. State variables are initially only considered at time instant with a corresponding node in the DBN. In Section 5.2, this restriction is lifted and linear interpolation is applied to conduct approximate reasoning in continuous time. The analysis techniques presented here are demonstrated by applying them to the examples in Section 7.

5.1 What-if analysis

The probability distribution of any node in a DBN is calculated efficiently with the help of Bayesian network software such as SMILE and its graphical interface GeNIe [10]. This applies to both marginal and conditional probability distributions. Conditions can be set on any number of nodes anywhere across the DBN. What-if analysis for various scenarios can thus be performed.

When performing what-if analysis, it is important that the conditions that are set are probable enough for the conclusions to be based on a sufficient amount of data. One way to verify this is to obtain the probability of all conditions being true and multiplying this with the size of the simulation data set to get an approximate sample size for the conditional probability distribution. This is, however, merely a guideline. Some conditions can reduce the approximate sample size substantially without having any significant impact on the accuracy of the probability distribution that is being investigated.

An application of DBN metamodels is to determine the marginal probability distributions of their variables. Such probabilities can be calculated for state variables at the time instants included in the DBN, i.e., $\Pr(x_k(t) = j)$ and for the output variables, $\Pr(z_k = j)$. It is also possible to calculate the expected value of a variable, $\mathrm{E}[x_k(t)]$ or $\mathrm{E}[z_k]$, when its probability distribution is known. This applies to both marginal and conditional probability distributions. If the variable in question is discretized during the construction of the metamodel, this also affects the expected value of the variable since it is calculated based on the discretized values.

More elaborate results are obtained when conditional probability distributions are examined. The DBN describes the joint probability distribution of all of its variables, so conditions can be set on any of them. The dependence between inputs and outputs, for instance, are investigated by fixing the values of the inputs and examining the probability distribution of outputs using probabilities such as $\Pr(z_k = j | u_{k'} = j')$.

Even though the marginal probability distributions of input variables are

known, the conditional probability distributions are not. It can therefore be of interest to investigate which inputs are most likely to lead to given values for outputs. This inverse reasoning is conducted by calculating probabilities such as $\Pr(u_k = j | z_{k'} = j')$.

The inclusion of state variables in the metamodel also allows the examination of conditional probabilities involving them. The probability distribution of a state variable at a specific time instant with fixed values for the inputs is calculated with probabilities such as $\Pr(x_k(t) = j | u_{k'} = j')$. The probability distribution of outputs with fixed values for a state variable at some time instant is similarly calculated with probabilities like $\Pr(z_k = j | x_{k'}(t) = j')$. It is also possible to set conditions on the same variable that is being examined in order to investigate probabilities like $\Pr(x_k(t) = j | x_k(t') = j')$.

Since the metamodel represents the joint probability distribution of all variables, conditions can be combined in numerous ways to investigate complex scenarios. The choice of conditions depends on the intended use of the metamodel. The probability distributions provided by the metamodel are more reliable when considering scenarios similar to the ones that were used in validating the metamodel.

5.2 Approximate reasoning in continuous time

Even though the DBN metamodel provides probability distributions for all of the nodes in the DBN, it is also of interest to examine probability distributions involving other time instants. Linear interpolation is used to create approximations of such distributions [38]. When marginal probabilities are considered, accurate approximations for the probability distributions can be obtained by linear interpolation between the probability distributions of the previous and following node corresponding to the variable in question.

A more challenging task is the approximation of conditional probabilities. There are two cases to consider. The simpler one is when the conditions coincide with the nodes. The more complex situation arises when a condition is set at a time instant that does not coincide with any node. In the first case, the conditions can be included directly in the DBN. It is then possible to proceed with linear interpolation as in the case of marginal distributions. The conditional probabilities of the nodes generally still provide accurate estimates but the interpolation between them can occasionally produce notable errors. This is because the selection of the time instants was optimized for the marginal distributions. Adding conditions makes the probability curves between nodes less linear in some cases which results in a greater estimation error. However, the error is still usually small and can be reduced when necessary by constructing a new metamodel with a larger number of nodes.

The situation is more complex if no node corresponds to the time instant of a condition. This problem is dealt with by setting conditions separately for the previous and following nodes corresponding to the variable in question and taking a weighted average of the resulting conditional probability distributions [38]. Effectively, interpolation is performed on the conditions. The resulting probability distribution is less accurate than the ones produced in the simpler cases, because even more non-linearity has been introduced. In practice, the estimate error can be large for time instants close to the condition, but is smaller for time instants that are further away. The accuracy of the estimate close to the condition can be increased if the condition applies to the same state variable whose probability distribution is examined. Since the variable's value is fixed at the time instant of the condition, an artificial extra node, where the appropriate value occurs with a probability of 1, can be added for the purpose of interpolation.

Multiple conditions can be set simultaneously at time instants that are not included in the DBN. Interpolation is then carried out for each of the conditions. The interpolations overlap and occasionally produce highly inaccurate results if the time instants of conditions are too close to each other. This problem is most easily solved by reconstructing the metamodel using a larger number of time instants for the variables in question to prevent the overlapping.

The SMILE library allows the study of the probability distribution of any node of the DBN. In particular, the distribution of every node corresponding to the same state variable can be examined to assess the time evolution of the probability distribution of the variable. This, however, only provides distributions for each time instant separately rather then their joint distribution. Because of this, it is not known how individual realizations of the simulation model are likely to behave. There usually is strong dependency between a state variable's value at different time instants, in particular when the time instants correspond to consecutive nodes. To explore a variable's time evolution more closely, conditional probabilities must be examined.

Automating the calculations involved in the utilization of DBN metamodels allows for more comprehensive analysis. The interpolation required for reasoning in continuous time in particular is tedious without automation. The time evolution of probability distributions can easily and quickly be examined graphically by calculating and plotting all the relevant distributions programmatically.

6 Implementation of the automated construction of DBN metamodels

A DBN metamodeling tool is created in this thesis using primarily the MATLAB environment [29]. MATLAB is used because it offers a suitable compromise between practicality and computational efficiency. The graphical user interface (GUI) of the tool is created with MATLAB, as is the algorithm for finding optimal time instants. The SMILE library [11] is used to handle DBNs, including the learning algorithms used to determine the graph and CPTs.

The DBN metamodeling tool automates most aspects of the construction of metamodels discussed in Section 4. Some decisions need be to made by the user of the tool but default options are provided when reasonable. Certain parameter values, such as technical parameters of the optimization algorithms, are included directly in the source code of the tool. These are not incorporated in the GUI because they are judged to merely clutter it up without providing practical benefits. Anyone with basic knowledge of MATLAB can change the relevant parts of the code if required. The tool saves the constructed DBN metamodels. It is possible to later open the metamodels for analysis with the tool or the GeNIe software [10].

The implementation of each of the five phases of the construction of DBN metamodels is discussed in this section. The importing of data into the DBN metamodeling tool is considered in Section 6.1. The preprocessing of data is presented in Section 6.2. The selection of time instants by the tool for the DBN is discussed in Section 6.3. In Section 6.4, the estimation of the graph and CPTs of the DBN is considered. The validation features of the tool are introduced in Section 6.5. Finally, the utilization of the resulting DBN metamodel with the help of the tool is illustrated in Section 6.6.

6.1 Data import

The simulation data that is used in the construction of DBN metamodels is created with a separate simulation model. The DBN metamodeling tool does not interact with the simulation model in any way apart from reading the data. If additional information about the simulation model is to be taken into consider during the construction of the metamodel, it must be entered by the user of the tool through the GUI.

The simulation data has to be stored in a data format that the DBN metamodeling tool supports. The data is saved in a plain ASCII file in order to avoid any unnecessary software dependency. When opening a data file

from the GUI, the tool verifies that the content of the file is of the appropriate format. Each simulation replication in the data must have the same starting time and every variable must be given an initial value in every replication.

6.2 Data preprocessing

The second phase of the construction of a DBN metamodel is the selection and discretization of variables. Any variable that obtains just one value in the entire simulation data is automatically discarded as useless. All other variables are included in the metamodel by default but they can also be discarded by the user.

The discretization of the values of variables is performed for one variable at a time. The GUI is used to choose number of discretized values or bins that the original values are sorted into. There is an upper bound of 16 for the possible number of bins in order to limit the size of CPTs in the DBN. This number has proven to be sufficient for describing almost any distribution with satisfactory accuracy. The number of bins should ideally be well below the limit for most variables. The more bins there are, the more simulation replications are needed to accurately estimate the CPTs.

The values of a variable are first rounded slightly by the DBN metamodeling tool for computational efficiency. A histogram is then drawn of the rounded values of the variable in the simulation data. The histogram is divided into sections of different colors to indicate which bin each value is assigned to. Fig. 10 shows the interface used in the selection and discretization of variables. The possibility to name each variable is also implemented in the tool.

A variable is automatically classified as a state variable if its value changes during any replication of the simulation model. The variable is otherwise assumed to be an output variable but it can also be specified by the user to be an input variable. The selected variables are transformed into a more efficient data structure for the following phases.

6.3 Selection of time instants

The third phase of the construction of DBN metamodels, the selection of time instants, is performed for one state variable at a time. The first step of the phase is the discretization of the simulation's time interval. The discretized time instants are spaced evenly over the interval but the number of segments that the time interval is divided into can be chosen freely. The default value for this number is 500, because it has proven to be sufficient in many cases and the running time of the dynamic programming algorithm for optimally

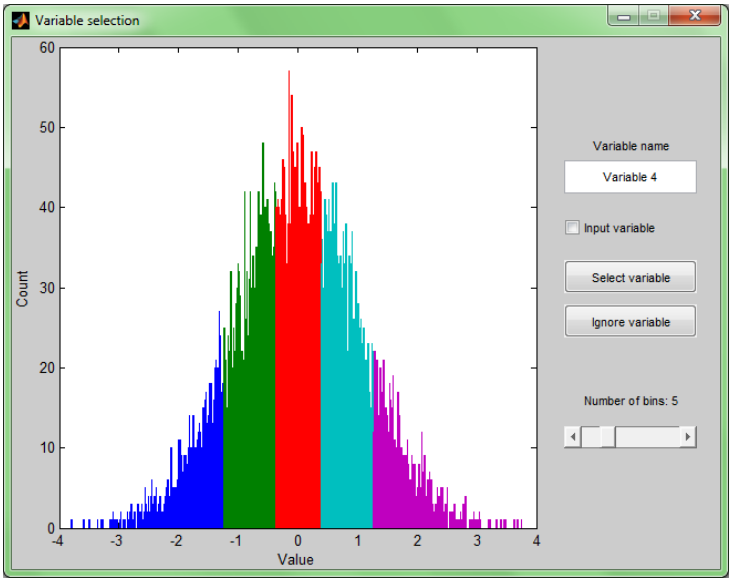


Figure 10: Interface for selecting and discretizing variables.

selecting time instants is not excessive. It is recommended to consider the characteristics of the variable in question when selecting the number of time instants. A larger number of time instants is justified if the probability curves fluctuate more rapidly. The GUI lets the user open up plots in a new window, allowing probability curves to be examined more closely before deciding on the discretization.

The values of the variable in all replications at all relevant time instants are looked up from the simulation data for each potential discretization. The probability curves are formed based on the relative frequencies of the variable values in the data. The values of the variable in every simulation replication and at every time instant are not explicitly recorded, as this would require a considerable amount of additional memory. The probability curves are plotted for visual inspection. The number of time instants can be changed repeatedly and the probability curves are recalculated each time. Once a discretization is chosen, the next step is to select which optimization algorithm to use in selecting optimal time instants. Dynamic programming is preferred but the genetic algorithm is also available and can be used if the dynamic programming algorithm is considered too slow. After selecting the optimization algorithm, it is still possible to return and change the algorithm or the number of time instants if the initial choices turn out to be inappropriate.

If dynamic programming is used, the maximum number of nodes allowed, the maximum time gap between consecutive nodes, and the objective function to minimize are provided to the DBN metamodeling tool through the GUI.

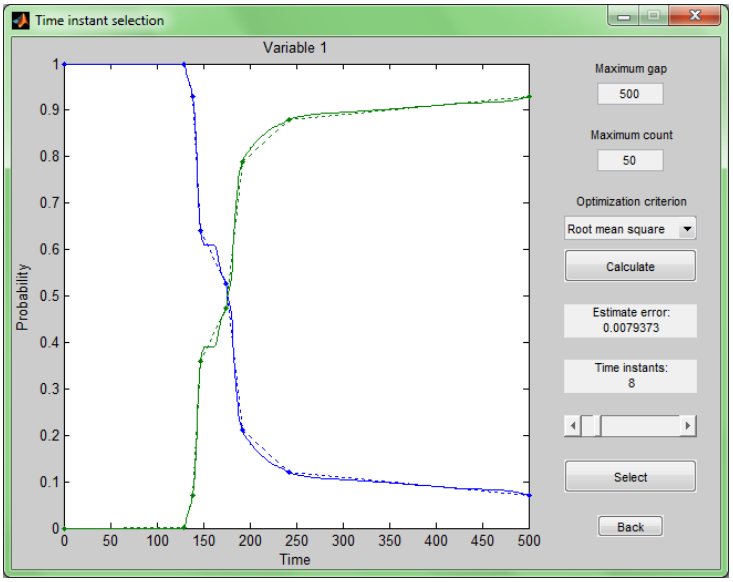


Figure 11: Interface for the dynamic programming algorithm in the selection of time instants.

The optimal time instants are calculated simultaneously for all numbers of nodes up to the given maximum. Since the optimization can take a long time if the initial time discretization is too dense, the calculation can be manually interrupted.

The time evolution of the estimate for the probability distribution of the variable in question is calculated based on the selected number of nodes. This is overlaid on the same plot as the probability curves. The size of the estimate error is also shown to provide a numerical indicator. This allows for comparison between different numbers of nodes which can be changed by moving a slider. It may be an indication that the discretization should be denser if the selected time instants include two consecutive ones from the initial discretization. The DBN metamodeling tool gives a warning if this is the case but does not disallow the selection. The interface of the dynamic programming algorithm is shown in Fig. 11.

When using the genetic algorithm, the DBN metamodeling tool can only return one solution candidate at a time. The number of nodes is indirectly controlled by adjusting a parameter representing the penalty added to the objective function for each additional node. Other parameters regarding population size, the number of generations, and the choice of objective function are also specified via the GUI. Default values are given by the tool, but they are not suitable for all situations. The estimate and the true probability curves are again overlaid and the estimate error is displayed. The

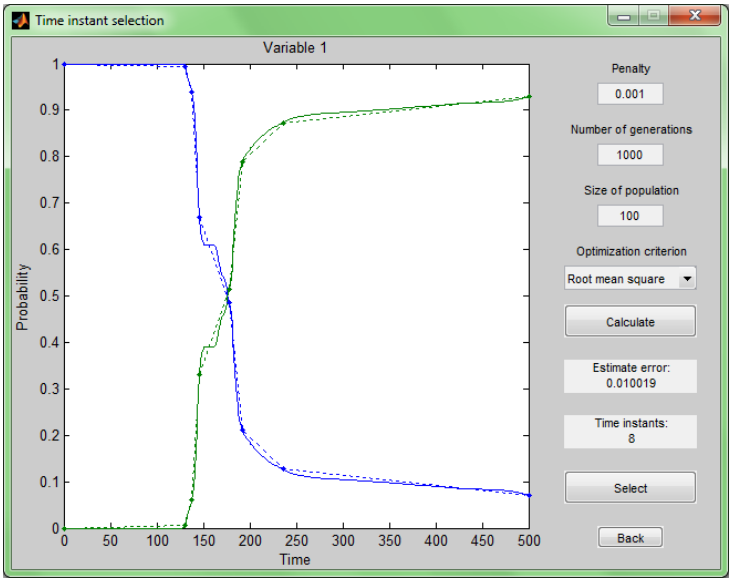


Figure 12: Interface for the genetic algorithm in the selection of time instants.

solution is accepted or a new one is calculated. Since the genetic algorithm is stochastic in nature, it might be worthwhile to run it multiple times with identical parameter values. As with the dynamic programming algorithm, the genetic algorithm can also be interrupted by the user if needed. The interface of the genetic algorithm is presented in Fig. 12.

As mentioned in Section 4.3.4, the estimate error for linear segments that are evaluated by the genetic algorithm should be stored. The DBN metamodeling tool uses MATLAB's sparse matrix structure. The degree to which this matrix is filled depends on the number of time instants as well as the parameter values of the genetic algorithm. The matrix is stored only for a single optimization run at a time. Fig. 13 illustrates the contents of the matrix in a sample case. Here, only about 2% of the possible segments between pairs of time instant are actually evaluated.

6.4 Estimation of the graph and conditional probability tables

The next phase is the estimation of the graph and CPTs of the DBN. Before applying the GTT algorithms for learning the DBN, the search space of the graph is restricted. This is accomplished by only considering graphs that include specific arcs and do not include other specified arcs. All arcs that would go backwards in time are first automatically disallowed.

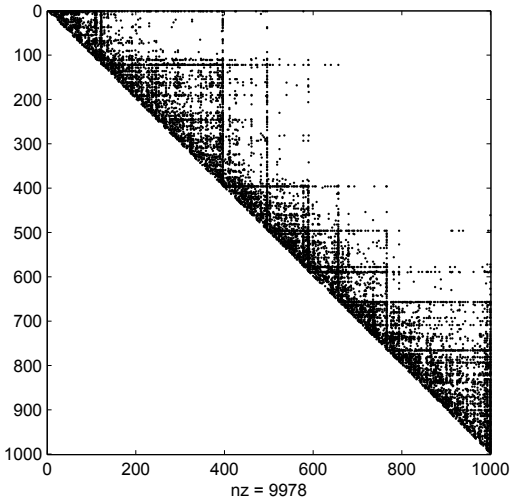


Figure 13: Time instant pairs for which the error of the segment between them is evaluated by the genetic algorithm in a test case. With 1000 time instants, there are 500,000 possible pairs, but only 9978 of them are evaluated.

The user can then introduce further restrictions on the graph through the GUI of the tool. Arcs given by the user are explicitly included or excluded. However, arcs between nodes corresponding to the same time instant can only be included in the DBN as long as the graph can still be acyclic. The GUI provides features for including multiple arcs at once. Arcs can be included from every node corresponding to a state variable to the next node corresponding to the same variable. Arcs can also be included between different variables in a similar fashion. The interface for entering user input regarding the graph is shown in Fig. 14.

Once all of the expert knowledge regarding the graph has been entered, the graph is determined automatically using the GTT algorithm implemented in the SMILE library with BDeu priors. The CPTs are also estimated automatically by the same library. The resulting DBN is stored in a new file by SMILE. In addition, a graphical representation of the graph is presented to the user.

6.5 Validation

The validation features of the DBN metamodeling tool allow the specification of conditions on any input or output variables as well as state variables at any time instants. The tool then determines the sample size for the conditions

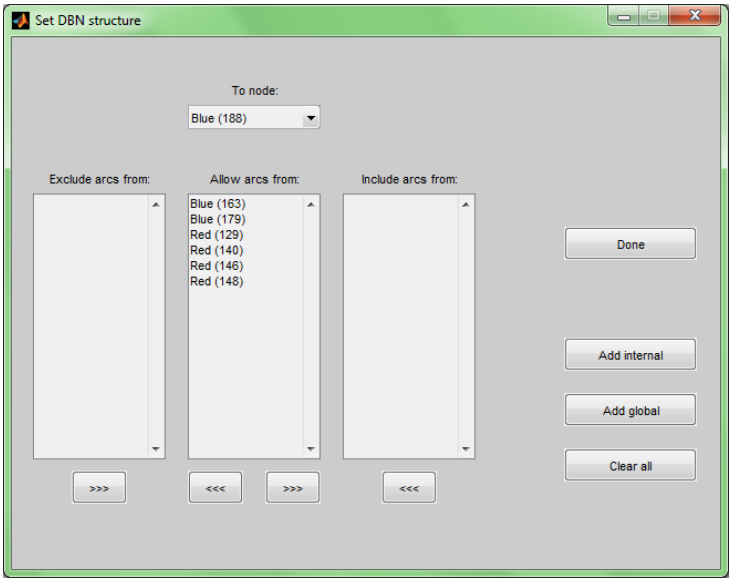


Figure 14: Interface for entering user input about the graph of the DBN.

by counting how many of the replications in the simulation data fulfill them. The maximum standard deviation for the relative frequencies of the variable values in the simulation data, i.e., the standard deviation for a conditional probability of 0.5, is calculated based on the sample size and displayed to the user.

A target variable, i.e., the variable whose probability distribution is investigated, must first be selected when comparing the metamodel to the simulation data. If the target variable is an input or output variable, a bar graph is displayed. The relative frequency and the metamodel's conditional probability are compared for each value of the variable. The RMSE of the variable is also presented. If the target variable is a state variable, a plot indicating the time evolution of the estimate errors is displayed. The plot includes a curve for each value of the variable. By examining the time evolution, it is simple to identify the time instants where the estimate error is greatest. The RMSE is also given. Alternatively, the DBN metamodeling tool can instead display the time evolution of the relative frequencies and the conditional probability distribution of the metamodel superimposed in a single plot. It is thus possible to examine what the relative frequencies are when the estimate error is large rather than just the timing of the large errors.

Fig. 15 shows the interface used for validating the metamodel. The state variable *temperature* is examined with a condition set on the input variable *month*. The maximum standard deviation in the simulation data is

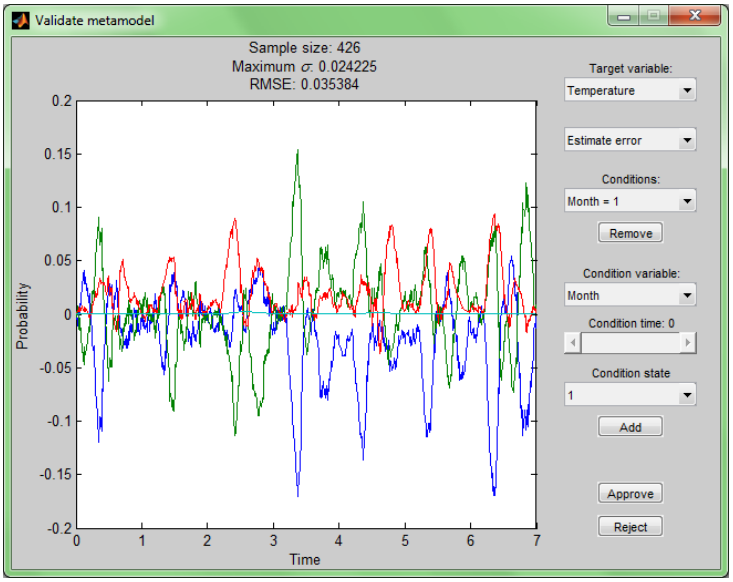


Figure 15: Interface for validating the metamodel.

approximately 0.02 while the RMSE of the metamodel is over 0.03. Whether or not this is acceptable depends on the requirements set for the metamodel. It is also worth noting that the estimate error for two particular values momentarily grows to as big as 0.15 which can be problematic if the time instants in question are of particular interest.

The metamodel is rejected if it appears to be invalid and a new refined one can be created instead. The metamodel is approved if it passes all validation tests. The DBN metamodeling tool then provides functionality to assist in the analysis of the constructed metamodel.

6.6 Utilization of DBN metamodels

In addition to constructing DBN metamodels, the DBN metamodeling tool is also capable of assisting in the use of the metamodels. Input variables can be assigned any probability distribution with the GUI. The state and output variables can be given fixed values in order to investigate conditional probabilities. The conditional probability distribution of any variable at any time instant, with conditions set on any variables at any time instants, can be calculated. The only restriction is that the conditions must actually be possible to fulfill. A node in the DBN cannot be assigned a value that it never obtains in the simulation data. A node also cannot be assigned two contradicting values. The tool cannot assist in analysis that require the graph to be extended by adding nodes that do not directly correspond to a variable

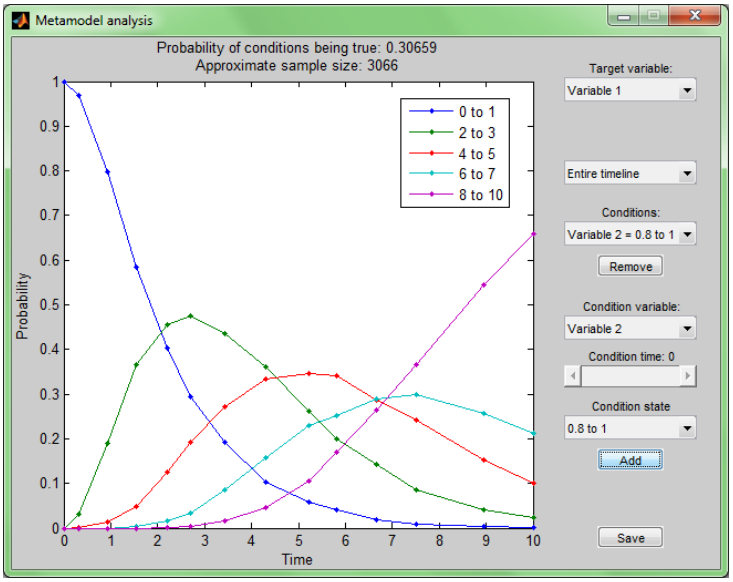


Figure 16: Interface for utilizing the metamodel.

of the simulation model. Fur such studies, it is necessary to open the DBN in GeNIe.

The DBN metamodeling tool can display the estimated probability distributions in three different ways. The first is as a bar graph for the probability distribution of a state variable at a specific time instant or the probability distribution of an input or output variable. For state variables, the distribution is first calculated for all of the nodes corresponding to the variable in question. When the time instant is changed, only the interpolation of the distributions needs to be recalculated. This requires little computational effort and can therefore be carried out quickly. A presentation that resembles animation can thus be created by continually advancing the time instant of the target variable.

The second option is to display the time evolution of the probability distribution of a state variable. The presentation is similar to that of probability curves discussed earlier. In this case, the curves are piecewise linear. Markers in the plot indicate the nodes corresponding to the variable in question. The probability of conditions being fulfilled as well as an approximate sample size derived from this are shown to give indication to the accuracy of the probability distributions. The interface for displaying the time evolution of a probability distribution is shown in Fig. 16.

The third option is to display the time evolution of the expected value of each state variable. The expected values of the variables are first scaled based on the maximum and minimum values that they can obtain and are

then displayed all in the same plot. It is impractical to determine the exact expected values based on the plot, but it provides a convenient way to compare the time evolution of different variables in a single plot.

The DBN metamodeling tool only works with metamodels that it has created. There is occasionally need to modify the DBN, for instance by adding a node that describes the joint probability distribution of two other nodes. This cannot be accomplished with the tool. The metamodel can instead be saved to file for modification and analysis in GeNIe.

7 Example simulation studies

The utilization of the automated approach to constructing DBN metamodels and the DBN metamodeling tool is illustrated by applying them in two example studies. The studies demonstrate key aspects of the construction and use of DBN metamodels. In the first example, combat between two aircraft is analyzed. This example demonstrates the effectiveness of the dynamic programming algorithm in selecting the time instants included in the DBN. Conditional time evolution of state variables is examined in different situations, including one that requires augmentation of the DBN. The second example deals with the operation of an air base. It includes a parameter of the simulation model treated as an input variable and the discretization of continuous variables. The validation of the DBN metamodels is demonstrated in both examples. The ability of DBN metamodels to investigate any conditional probability distributions is showcased by examining various studies involving input, state, and output variables. The time evolution of the expected value of a state variable is also investigated.

7.1 Air combat

The first example involves the simulation of two aircraft, referred to as blue and red, engaged in air combat for up to 500 seconds. A DBN metamodel is constructed based on simulation data produced by a separate air combat simulation software called X-Brawler [27]. The construction is carried out with the help of the DBN metamodeling tool. The validity of the metamodel is checked and analysis is conducted in the form of examining the time evolution of conditional probability distributions.

The simulation begins from an asymmetric starting position. The simulation is described by two state variables that obtain one of two possible values describing whether or not an aircraft has been shot down. The state variable x_1 equals 1 if blue is in the air at time instant t and $x_1(t) = 0$ if it has been shot down. x_2 obtains values similarly according to the situation of red. Certain conditional probabilities are trivial because of the properties of the simulation model. Once an aircraft has been shot down, it cannot return. If $x_i(t) = 0$, then $x_i(t') = 0$ for all $t' > t$. Similarly, if $x_i(t) = 1$, then $x_i(t') = 1$ for all $t' < t$. Furthermore, the state of one aircraft does not change if the other aircraft has already been shot down a while ago. It is possible for both aircraft to shoot each other down but this must happen within a short period of time.

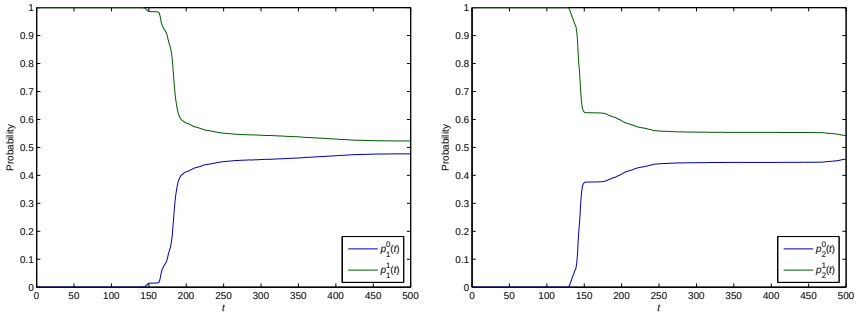
The simulation data consists of 10 000 replications and has been created

independently prior to the metamodeling so the first phase in the construction is bypassed. Since the state variables are discrete with only two possible values and both variables are included in the metamodel, there is no need for any preprocessing. The construction therefore actually begins only from the third phase.

The first step of the selection of time instants is the calculation of the probability curves which are displayed in Fig. 17. An example based on the same simulation model is included in [38] but without implementing multiple time scales in the metamodel and instead using the same 15 time instants for both variables. The piecewise linear approximation of the probability curves resulting from this selection of the time instants is compared to the approximation obtained with the time instants provided by dynamic programming. The time instants in [38] are selected using a different algorithm than the ones presented in this thesis but examining the RMSE still demonstrates the accuracy of approximation created with the dynamic programming algorithm. Using the time instants in [38], the RMSE is 0.0087. This is still significantly better than the RMSE of 15 equally spaced time instants which is 0.0288. Using 15 common time instants selected optimally with the help of the DBN metamodeling tool by employing the dynamic programming algorithm, the RMSE is 0.0017. By implementing multiple time scales and selecting 15 time instants separately for each variable, the RMSE drops even further to 0.0008. The significance of these errors depends on the requirements of the metamodel. It is clear, however, that using multiple time scales and the dynamic programming algorithm improves the selection of time instants.

The usefulness of the dynamic programming algorithm and multiple time scales is also demonstrated by creating approximations of the probability curves that are as accurate but contain fewer nodes. A RMSE equal to the optimal selection of 15 time instants with a common time scale is achieved by selecting 11 time instants separately for each variable. A RMSE equal to the time instant selection in [38] is achieved with just 6 optimally selected time instant per variable. The computational efficiency of the dynamic programming algorithm is also evident. Using 500 candidate time instants, it takes only a few seconds to calculate the optimal selection for a variable. This is a significant improvement compared to previously used optimization algorithms discussed in [39].

The optimally selected time instants include two consecutive time instant from the initial discretization of the time interval for both variables. This is an indication that a denser discretization of the time interval might be needed. In this case, however, the piecewise linear approximation fits the probability curves so well that a denser discretization is unlikely to improve it. The selection of consecutive time instant is in part caused by using a greater



(a) Probability curves for blue, x_1. (b) Probability curves for red, x_2.

Figure 17: Probability curves for the state variables of the simulation model.

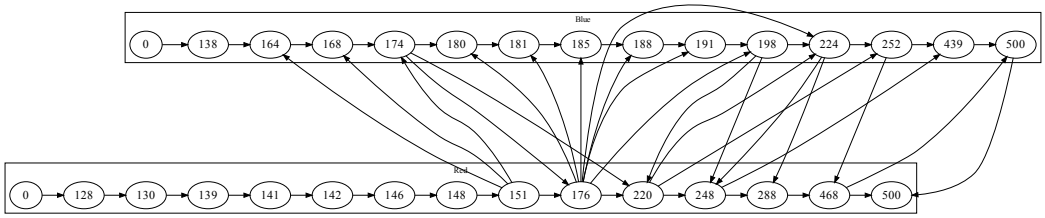


Figure 18: DBN metamodel of the air combat simulation model. The numbers indicate the time instants of the nodes.

number of nodes than is actually necessary in order to make the metamodel comparable to the one presented in [38].

After the selection of time instants, the graph of the DBN metamodel is estimated. Expert knowledge is applied to the two nodes in the DBN corresponding to the time instant 0, since they always obtain the same value in the simulation data. Such nodes are problematic for the learning algorithm of the DBN metamodeling tool as arcs are automatically placed between the nodes but they are never connected to any other nodes. All other dependencies in the DBN are identified by the algorithm. The resulting DBN is shown in Fig 18. There are arcs originating further back than just the previous node, such as the arc from $x_1(174)$ to $x_2(220)$. This suggests that the system is not a Markov process.

The optimally chosen time instants are different for x_1 than x_2 due to the asymmetrical nature of the underlying system. Fig. 17 implies that the probability distribution for x_2 changes most rapidly before $t = 150$ while the probability distribution for x_1 changes most rapidly after $t = 165$. This is illustrated further in Fig. 19 where the expected values of the state variables as given by the metamodel are overlaid. Because the only possible values of the state variables are 0 and 1, the expected value of a variable is identical

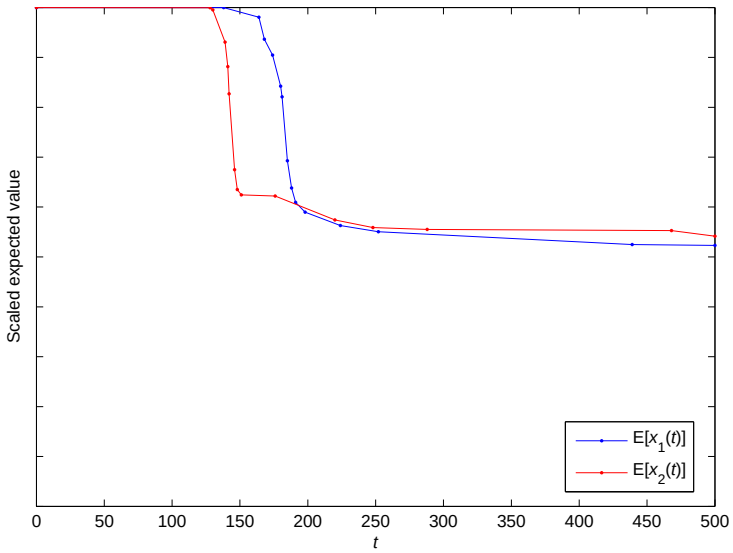


Figure 19: Time evolution of the expected value of the state variables x_1 and x_2. Since the two variables both only obtain the values 0 and 1, the expected values are automatically scaled identically and are equivalent to the probability of obtaining the value 1.

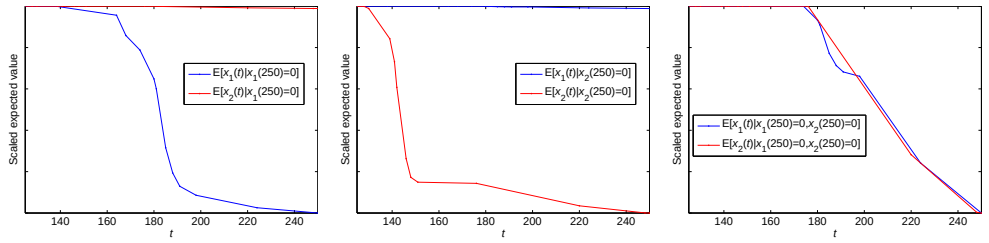
to the probability of it obtaining the value 1. It would be challenging to describe a system where the variables change value at different times like this efficiently without the use of multiple time scales.

Four scenarios are considered in the validation of the metamodel: no conditions (i.e., marginal probability distributions), blue is shot down at some point ($x_1(500) = 0$), red is shot down at some point ($x_2(500) = 0$), and both aircraft are shot down at some point ($x_1(500) = 0$, $x_2(500) = 0$). The maximum standard deviation of the relative frequencies and the RMSE for each variable in the scenarios are shown in Table 2. In each case, the standard deviation in the relative frequencies is greater than the RMSEs. This means that the sample size is the greater source of error in these scenarios. Since conducting additional simulation replications is not possible, this error cannot be reduced. It is therefore questionable whether it is worthwhile to attempt to create a more accurate metamodel that would decrease the RMSEs, since the effect on the overall accuracy of the metamodel is limited. For the purposes of this example, the metamodel is deemed valid.

The metamodel has no input or output variables which means that all what-if analysis is performed by only setting conditions on the state variables. The interdependence of x_1 and x_2 is investigated in Fig. 20, where the time evolution of the conditional probability distributions of the variables is shown

Table 2: Validation results for the metamodel. The maximum standard deviations of the relative frequencies denoted by σ and the RMSEs of the state variables in the four scenarios.

Condition	—	$x_1(500) = 0$	$x_2(500) = 0$	$x_1(500) = 0$, $x_2(500) = 0$
σ	0.0050	0.0074	0.0072	0.0620
RMSE of x_1	0.0009	0.0019	0.0005	0.0463
RMSE of x_2	0.0008	0.0004	0.0017	0.0312

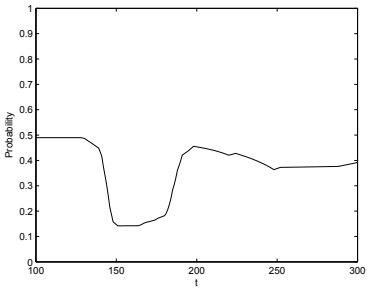


(a) $x_1(250) = 0$. (b) $x_2(250) = 0$. (c) $x_1(250) = 0$, $x_2(250) = 0$.

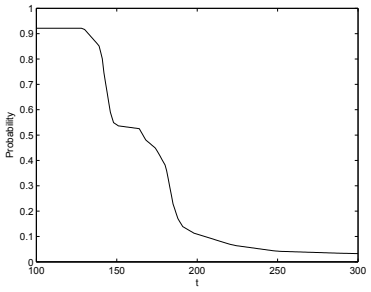
Figure 20: Time evolution of the expected values of x_1 and x_2 with the conditions set at $t = 250$.

for the time interval $t \in [125, 250]$. The conditions apply to the variables at time instant $t = 250$. In the first two scenarios, the value of one of the state variables is set to 0 at that time instant. In the third one, both conditions are applied simultaneously. This scenario only occurs with a probability of approximately 0.005 according to the simulation data, so the results are somewhat imprecise. What is noteworthy in these results is that in the first two scenarios, the probability distribution related to the aircraft that is shot down changes at different time instants. This is as expected based on Fig. 19 where the different time evolution of the variables is apparent. In the third scenario, however, the probability distributions of the variables evolve in a similar manner to each other. This reflects the properties of the underlying system. In order for both aircraft to be shot down, the two events must happen soon after one another.

A scenario with multiple conditions is considered next. Both aircraft are in the air at time instant t (i.e., $x_1(t) + x_2(t) = 2$) and exactly one is in the air at the end of the simulation ($x_1(500) + x_2(500) = 1$). The latter condition cannot be set directly in the DBN so either the model must be augmented or the two ways that the condition can be fulfilled must be considered separately. The conditional probability of blue being the aircraft



(a) Conditional probability of blue winning.



(b) Probability of the conditions.

Figure 21: Time evolution of the conditional probability $\Pr(x_1(500) = 1|x_1(t)+x_2(t) = 2, x_1(500) + x_2(500) = 1)$ of blue winning when both aircraft are in the air at time instant t but only one is at the end of the simulation, as well as the probability $\Pr(x_1(t) + x_2(t) = 2, x_1(500) + x_2(500) = 1)$ of the conditions being fulfilled.

to survive, i.e., $\Pr(x_1(500) = 1|x_1(t) + x_2(t) = 2, x_1(500) + x_2(500) = 1)$, is a function of t. Fig. 21 shows the time evolution of this conditional probability as well as the probability of the conditions being fulfilled.

The initial situation of the air combat is fairly even, as is seen in Fig. 21a. Red is the more likely victor if it survives blue's initial attack during the time interval 140-150. The situation is again practically even if blue survives red's counter attack at around 180-190. As is shown in Fig. 21b, the conditions are less likely for larger values of t which indicates that the corresponding conditional probabilities are less accurate.

The construction and utilization of DBN metamodels are demonstrated by this example study. The utilization of the dynamic programming algorithm and multiple time scales improves the selection of time instants which results in more accurate piecewise linear approximations of the probability curves. In the scenarios examined in the validation of the metamodel, the greatest source of inaccuracy is the quantity of simulation data. Since the data was generated separately from the construction of the metamodel, there is no way to address this potential inaccuracy. Analysis backward in time is conducted to investigate how different states of the system at time instant $t = 250$ are likely to have arisen. Finally, by extending the metamodel to consider a condition that involves multiple variables of the metamodel, it is possible to examine more complex conditional probabilities, such as comparing different final states of the system and determining their relative likelihoods.

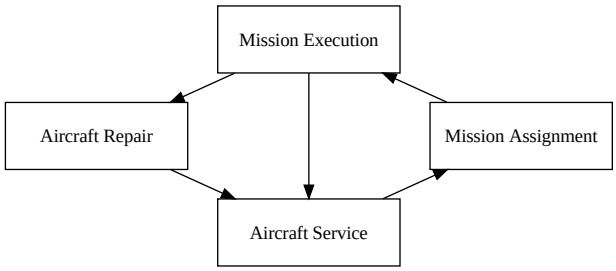


Figure 22: Flowchart of the simulation model. The aircraft repair, aircraft service, and mission assignment locations can each hold just one aircraft at a time and are therefore preceded by queues that are not included in the chart.

7.2 Air base operation

In the second example, the simulated operation of an air base is considered. A DBN metamodel is constructed according to the guidelines discussed in Section 4 with the help of the DBN metamodeling tool. The discretization of continuous variables is first presented. The validation of DBN metamodels is demonstrated by first constructing a metamodel that turns out to be invalid and then constructing a more accurate one. The resulting metamodel is analyzed by investigating various conditional probability distributions. These include the values of input and output variables and the conditional time evolution of a state variable.

In the simulation model, aircraft go repeatedly through a cycle consisting of mission assignment, mission execution, repair of possible damage obtained during the mission, and standard service such as fueling. There are three queues for the aircraft, all operating on the first in first out principle: one for mission assignment, one for repair, and one for service. The repair and service personnel can only work on one aircraft at a time. Aircraft that have not been damaged can move directly from mission execution to the service queue. An aircraft is released from the mission assignment queue every time a new mission becomes available. A backlog of missions is formed and the aircraft are assigned to the missions as soon as they arrive from service if there is no aircraft in this queue when a new mission is about to be assigned. A flowchart of the simulation model is presented in Fig. 22.

The missions are categorized into patrolling missions and combat missions. Patrolling missions are assigned regularly with the time between consecutive missions sampled from a uniform probability distribution. The aircraft are unlikely to be damaged during a patrolling mission. Combat missions are assigned as a Poisson process with a time dependent arrival intensity. Combat missions are shorter than patrolling missions on average but the aircraft

Table 3: Variables of the metamodel.

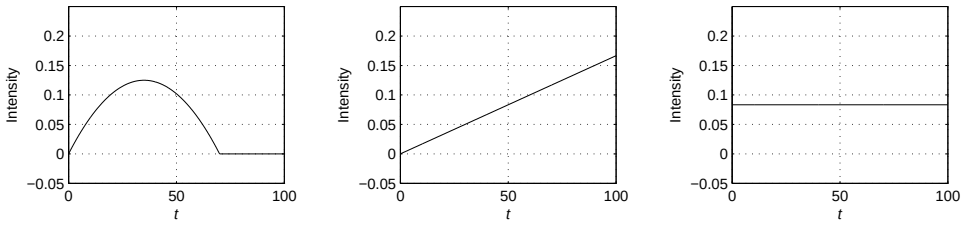
Name	Type	Range	Interpretation
u	Input	$\{1, \ldots, 4\}$	Scenario
x_1	State	$\{0, \ldots, 4\}$	Number of aircraft in mission assignment
x_2	State	$\{0, \ldots, 4\}$	Number of aircraft in aircraft repair
z	Output	$\{0, 1\}$	Mission assigned with no available aircraft

have a much higher probability of being damaged. The repair time of a damaged aircraft is stochastic and exponentially distributed. The service time is deterministic and depends on the length and type of the preceding mission.

The only input variable of the simulation model determines the time dependent intensity of the generation of combat missions and is denoted by u. There are a number of predefined scenarios for the time evolution of the intensity and u determines which of them applies to a particular simulation replication. The number of aircraft in each of the four locations are considered as state variables. In the metamodel to be studied, only the number of aircraft in mission assignment, denoted by $x_1(t)$, and in aircraft repair, denoted by $x_2(t)$, are taken into account. The output variable of the simulation model is an indicator, denoted by z, that describes whether or not at some point during the simulation replication, a situation occurs where no aircraft is available to execute an incoming mission. The variables included in the DBN and their ranges are summarized in Table 3.

Four combat mission generation scenarios are simulated. In the first one, the intensity of mission generation starts at 0, peaks early in the simulation, and return to 0 later. In the second scenario, the intensity slowly increases throughout the simulation. In the third one, the intensity is constant, and in the fourth one, there are no combat missions at all. The intensities related to the first three scenarios are illustrated in Fig. 23. The four scenarios occur with equal probability. In every simulation replication, four aircraft are included. The data is collected by running 2000 simulation replications for each scenario. The duration of each replication is 100 units of time. Since all of the variables under consideration are discrete and have a limited range of values, there is no need to discretize them.

In order to demonstrate the discretization feature of the DBN metamodeling tool, an additional output variable, denoted by w, is considered. w obtains the mean value of x_1 for each replication of the simulation model and is therefore a continuous variable. It is used for the demonstration of the discretization of variables, but is not included in the metamodel because

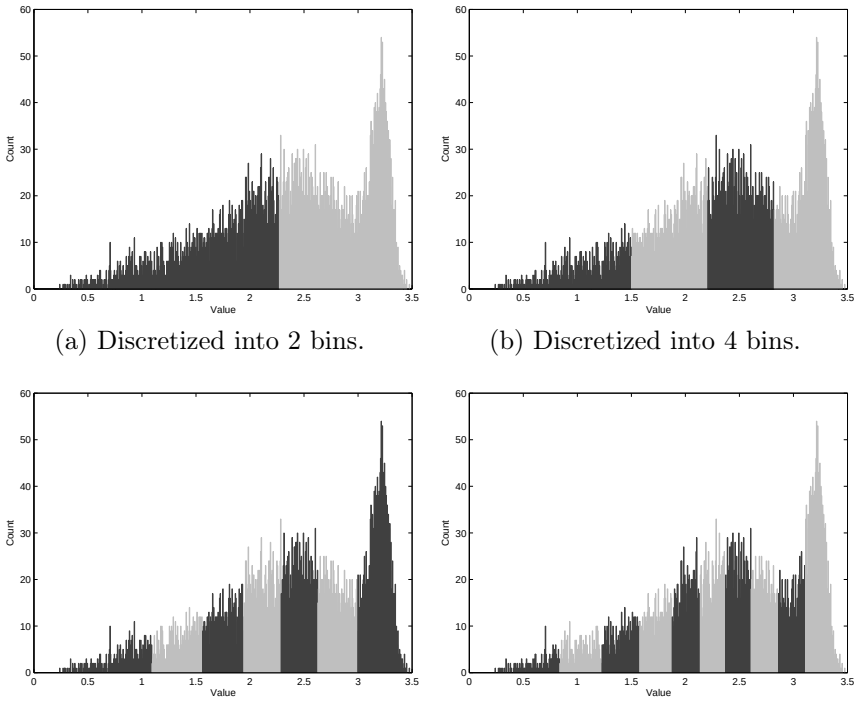


(a) Scenario $u = 1$. (b) Scenario $u = 2$. (c) Scenario $u = 3$.

Figure 23: Intensity of the generation of combat missions in three scenarios.

it is not needed in the analysis that is conducted with the metamodel. The probability distribution of w is skewed and has two distinct peaks, one corresponding to the scenarios $u \leq 3$ and one corresponding to the scenario $u = 4$. A histogram of the rounded observed values of w is drawn and the bars are colored by the tool according to what discretized value they are assigned to. Fig. 24 shows four of the possible discretizations of w into 2, 4, 7, and 10 bins, respectively. The figures imply that the width of the bins varies, as the longer left tail of the distribution is discretized into wider bins than the other values. The number of observations that falls into each bin also varies, since the large spike containing the greatest values of w is contained within just one bin unless the number of bins is particularly high. For this variable, it would be natural to let the spike correspond to one discretized value and using a total of 7 values is the best option in this regard.

The probability curves of the chosen state variables, calculated based on the simulation data, are shown in Fig. 25. The advantage of utilizing multiple time scales is evident. The probability distribution of x_1 changes rapidly due to the regularly scheduled patrolling missions while the distribution of x_2 changes more slowly. The repeatedly changing probability distribution of x_1 in particular requires a large number of nodes in order to be accurately described by the metamodel. Determining the optimal time instants for the state variables using the dynamic programming algorithm discussed in Section 4.3.3, 75 time instants are judged to be a suitable number for x_1, whereas 25 time instants are sufficient for x_2. The probability curves of x_2 can be quite accurately described by a smaller number of nodes but larger gaps between nodes are challenging when dealing with conditional probability distributions. This is a variable whose value can change quickly, even if its marginal probability distribution changes slowly. The RMSE is 0.0016 with the chosen time instants for x_1 and 0.0010 for x_2. Comparing these numbers to those in the first example in Section 7.1 shows how the probability distribution of x_1 is harder to approximate in a piecewise linear fashion. Even



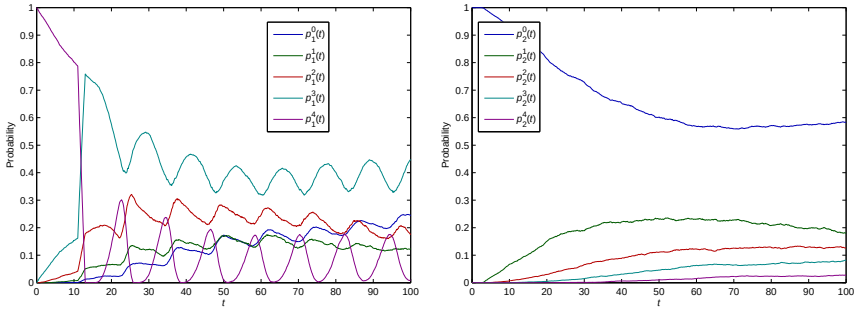
(a) Discretized into 2 bins.

(b) Discretized into 4 bins.

(c) Discretized into 7 bins.

(d) Discretized into 10 bins.

Figure 24: Alternatives for discretizing the variable w.



(a) Number of aircraft in mission assignment, x_1.

(b) Number of aircraft in repair, x_2.

Figure 25: Probability curves of the state variables of the simulation model.

with five times the number of nodes that were included for the variables in Section 7.1, the RMSE is still nearly double.

The graph for the DBN is determined in the fourth phase of its construction. Expert knowledge is initially used to add arcs only within each state variable from one node to the next. Learning algorithms are used to identify additional arcs in the DBN. A section of the DBN covering nodes with a time instant of less than 20, as well as the input and output variables, is shown in Fig. 26 This metamodel, however, proves to be invalid, as implied by Fig. 27, where the probability distributions provided by the metamodel are incompatible with the simulation data. This is also indicated by the RMSE that is 0.847. The time evolution of the probability distribution of x_2 given by the metamodel and the corresponding relative frequencies are compared in the figure. The probabilities in question are calculated with the condition of $u = 4$, i.e., the scenario where there are no combat missions. The curves deviate significantly. This implies that the learning algorithm does not add enough arcs to accurately describe the properties of the simulation model. Such an inaccurate metamodel is of little use and is therefore replaced by a more accurate metamodel.

In the second attempt at constructing a valid metamodel, more expert knowledge is included. Arcs are added to each node node corresponding to a state variable from the most recent nodes associated with each variable. Arcs are also added from the input and output variables to all other nodes. From a causal point of view, the arcs connecting the output variable should all be oriented towards it, but a node with such a large number of parent nodes is not computationally feasible. Orienting the arcs in the opposite direction can affect the conditional independence of other nodes but is a necessity in this case. A part of the DBN containing nodes corresponding to time instants of less than 20 is shown in Fig. 28. Validation based on the same scenario as

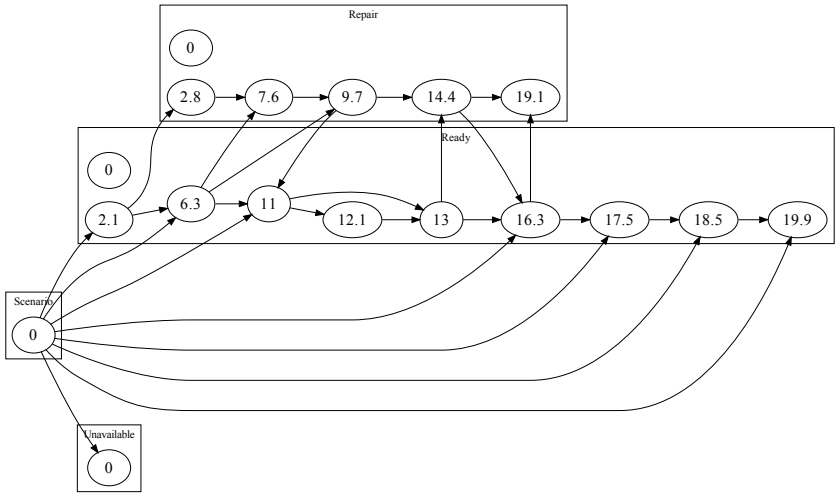


Figure 26: Part of the DBN metamodel containing nodes corresponding to time instants of less than 20. There are no arcs connecting the state variables at time instant 0, because they always obtain the same value and are therefore independent of all other nodes.

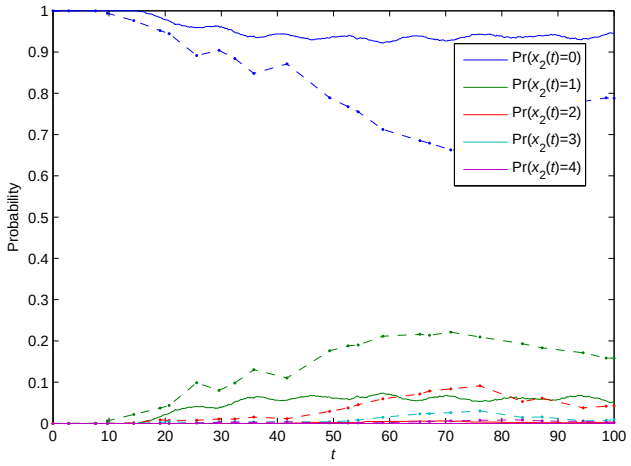


Figure 27: Comparison of the time evolution of the probability distribution of x_2, i.e., the number of aircraft in aircraft repair, with the condition $u = 4$. The distribution provided by the metamodel is represented by the dashed lines and the relative frequencies in the simulation data by the continuous lines. The sets of lines do not match which indicates that the metamodel is invalid.

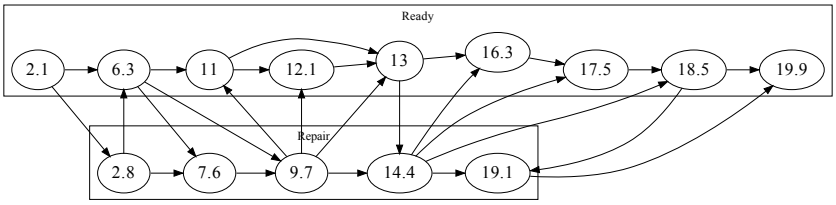


Figure 28: Part of the corrected DBN containing nodes corresponding to time instants of less than 20. The input and output variables are not included, because there are arcs from these to every other node which clutters up the figure.

before is presented in Fig. 29. Now, the probability distributions provided by the metamodel correspond well with the relative frequencies in the simulation data and the RMSE has dropped from what it was in the first metamodel to 0.0013. This is considerably less than the variance caused by the sample size. The metamodel therefore passes this validation test. This scenario shows no indication that the new metamodel would be invalid. The metamodel is considered to be valid for this example study. All further analysis discussed in this section involves only the corrected metamodel.

Fig. 30 depicts the unconditional time evolution of the simulation, i.e., the time evolution of the marginal probability distributions of the state variables provided by the corrected DBN metamodel. The distributions resemble the probability curves in Fig. 25. The periodical nature of x_1, caused by the regular patrolling missions, is evident in Fig. 30a. This is also the main reason why the concept of multiple time scales is useful in this example. The patrolling missions directly affect the number of aircraft available for missions, but have little impact on the number of aircraft needing repair.

Alternative what-if analyses facilitated by the DBN metamodel are demonstrated next. First, only the input and output variables are considered. Table 4 lists the marginal probability distribution of the output variable z as well as all its conditional probability distributions when the value of the input variable u is fixed. In Table 5, the situation is reversed and the probability distributions of the input variable u are examined with conditions on the output variable z. The results presented in Table 5 imply that there is practically always an aircraft available to execute incoming missions in the fourth scenario without any combat missions. The second scenario with a constantly increasing intensity in the generation of combat mission, on the other hand, is most likely to lead to a situation where a mission cannot immediately be carried out.

Since the second scenario proved to be the most demanding one with

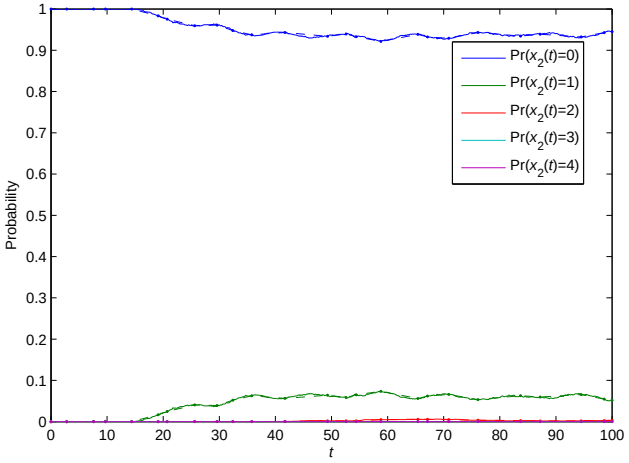
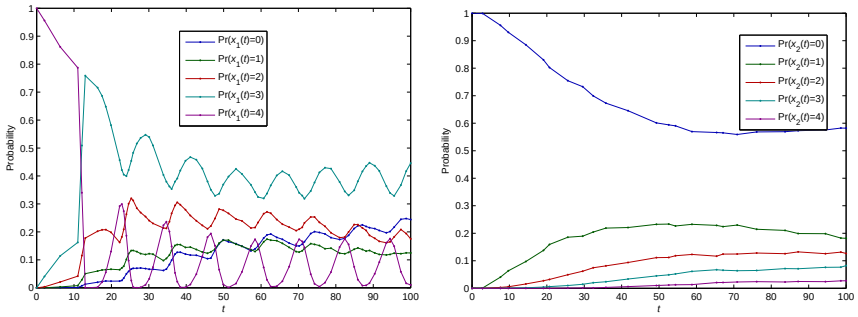


Figure 29: Comparison of the time evolution of the probability distribution of x_2, i.e., the number of aircraft in aircraft repair with the condition $u = 4$ in the reconstructed metamodel. The distribution provided by the metamodel is represented by the dashed lines and the relative frequencies in the simulation data by the continuous lines. The sets of lines are so close that it is hard to distinguish them in the figure.



(a) Number of aircraft in mission assignment, x_1.

(b) Number of aircraft in repair, x_2.

Figure 30: Time evolution of the marginal probability distributions of the state variables provided by the DBN metamodel.

Table 4: Conditional and marginal probability distributions of the output variable z conditional on the input variable u.

u	$\Pr(z = 0\|u)$	$\Pr(z = 1\|u)$
1	0.57	0.43
2	0.42	0.58
3	0.51	0.49
4	1	0
unknown	0.63	0.37

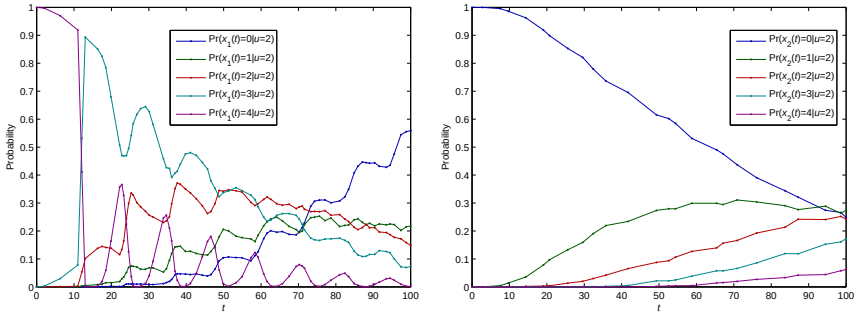
Table 5: Conditional and marginal probability distributions of the input variable u conditional on the output variable z.

z	$\Pr(u = 1\|z)$	$\Pr(u = 2\|z)$	$\Pr(u = 3\|z)$	$\Pr(u = 4\|z)$
0	0.23	0.17	0.20	0.40
1	0.29	0.39	0.33	0
unknown	0.25	0.25	0.25	0.25

regard to immediately executing missions, it is examined more closely. Fig. 31 shows the time evolution of the probability distributions of the state variables in this scenario. When comparing to Fig. 30, the biggest differences are towards the end of the simulation. It becomes increasingly unlikely that no aircraft is being repaired, while the probability of the mission assignment queue being empty steadily grows. This is consistent with the intensity of the generation of combat missions presented in Fig. 23.

The output variable z can obtain the value 1 only if the state variable x_1 obtains the value 0 at some point. This is explored more closely by setting $x_1(t) = 0$ at different time instants and examining the resulting conditional probability distribution of z. The conditional probabilities of $z = 1$ and the marginal probabilities of $x_1(t) = 0$ for seven different time instants t are shown in Table 6. Small values of t have not been included, since it is unlikely that the mission assignment queue would become empty early in the simulation. The probabilities vary somewhat between time instants due to the periodic nature of the probability distribution of x_1, but certain trends can be observed. The probability of the mission assignment queue being empty is at its highest right at the end of the simulation. However, it is most likely to lead to a backlog of missions if the queue is empty in sometime between time instants 70 and 80.

Finally, the effect on both state variables of fixing the value of x_2 at time



(a) Number of aircraft in mission assignment, x_1.

(b) Number of aircraft in repair, x_2.

Figure 31: Time evolution of the probability distributions of the state variables provided by the DBN metamodel conditional on $u = 2$.

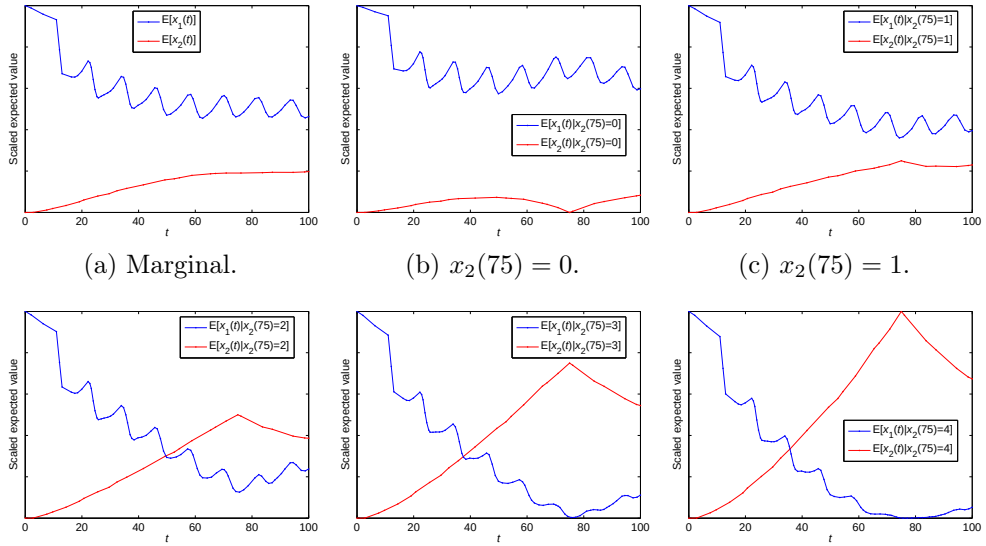
Table 6: Conditional probabilities of z and the marginal probabilities of the conditions.

t	$\Pr(z = 1 \mid x_1(t) = 0)$	$\Pr(x_1(t) = 0)$
40	0.87	0.12
50	0.85	0.17
60	0.90	0.16
70	0.95	0.15
80	0.94	0.18
90	0.90	0.21
100	0.83	0.24

instant 75 is examined. The values of the state variables are actual numeric values, so calculating expected values from their probability distributions is possible. In Fig. 32, the time evolution of the expected value is investigate for the marginal case as well as for all five alternative values of $x_2(75)$. The connection between the variables is evident: when there are more aircraft in repair, there are fewer waiting for a mission. It is also evident based on Fig. 32 that even though the values of the variables drift towards those in the marginal case after time instant 75, the differences are still large at the terminal time of the simulation.

The construction and analysis presented above is easy to perform with the help of the automated approach to constructing DBN metamodels and the DBN metamodeling tool. The tool automatically presents discretization alternatives for different numbers of discretized values when preprocessing continuous variables. The tool also assists in assessing the validity of resulting metamodels. Should the metamodel turn out to be invalid, a new refined one can be easily constructed.

DBN metamodels are applied as input-output mappings similar to other types of metamodels by examining the conditional probability distributions of output variables when the values of input variables are given. DBN metamodels are most useful, however, when investigating dynamic phenomena. The time evolution of state variables with fixed values for inputs are studied by setting conditions on the input variables. Similarly, the state of the simulation model at a particular time instant is fixed by setting conditions on state variables and then examining conditional probability distributions. When the values of state variables correspond to actual numeric values, the time evolution of expected values is also provided.



(a) Marginal. (b) $x_2(75) = 0$. (c) $x_2(75) = 1$.

(d) $x_2(75) = 2$. (e) $x_2(75) = 3$. (f) $x_2(75) = 4$.

Figure 32: Time evolution of the marginal and conditional expected values of the state variables with conditions applying to $x_2(75)$.

8 Conclusion

DBN metamodels are a recently developed type of simulation metamodels. Such metamodels have previously been constructed individually for each simulation study. This is time consuming and impractical. Such issues are bypassed by utilizing the automated approach to the construction of DBN metamodels developed and presented in this thesis and the DBN metamodeling tool. The construction is split into five distinct phases to more clearly define it: design of experiment, preprocessing of simulation data, selection of time instant, determination of the DBN, and validation.

In this thesis, new means have been adopted for some of the phases to make them better suited for automation. In the preprocessing of data, continuous variables are automatically turned into discrete ones and the number of values for discrete variables is reduced. An algorithm based on dynamic programming is used to select the time instants of DBNs. The accuracy of candidates for the time instants is quantified in order to simplify the choice between alternatives. In the validation of metamodels, descriptive measures of their accuracy are calculated in order to make the validation process more systematic.

The approach for automating the construction of DBN metamodels is implemented as the DBN metamodeling tool created using MATLAB. The tool provides a graphical user interface for inputting information required in the construction of the metamodel and for visualizing results. All of the construction phases except for the design of experiment are included in the tool, as are features for utilizing constructed metamodels. The tool is designed to work with generic DES data and can therefore be used regardless of the simulation model under consideration.

In this thesis, the automated construction approach is applied to two example studies. The first one deals with a previously generated data set describing combat between two aircraft. The other one involves the operation of an air base. The example cases demonstrate the construction of DBN metamodels and showcase the utilization of the metamodels in simulation studies.

DBN metamodels allow for simulation studies involving the state variables of a simulation model, unlike metamodels based on input-output mappings. Dependencies between output variables are also taken into consideration, while input-output mappings only consider each output variable separately. What-if analysis involving any of the simulation model's variables can be performed with the help of DBN metamodels. Automating the construction of the DBNs significantly simplifies the application of DBN metamodels by

allowing the focus to be on the analysis rather than the construction.

One area of potential future development in the construction of DBN metamodels is the selection of time instants for determining of the DBN's graph. The time instants that create the most accurate piecewise linear approximation of the probability curves are not always ideal for the DBN. Such challenging situation cannot always be solved by automation, but a closer integration of expert knowledge could be helpful. Features related to any particular simulation models could also be better taken into consideration instead of merely following the general automated approach.

The role of input variables is another area where there is still room for development. Input variables are currently assumed to be independent of each other which is not necessarily the case. Including possible dependencies between input variables in the metamodel would increase its accuracy. The learning algorithms employed to determine the graph of the DBN can be improved on, e.g., by incorporating expert knowledge regarding the prior probabilities of different graphs. Finally, the design of experiment phase could be linked more closely to the other phases of the construction. Sequential sampling could be used to adaptively generate more simulation data with input variables obtaining values that improve the accuracy of the resulting DBN metamodel the most.

The automatization of the construction of DBN metamodels makes the application of such models a more viable option in simulation studies. The unique features of the DBN metamodels can be utilized without having to spend a lot of effort on the construction. The dynamic programming algorithm presented in this thesis leads to the better selection of time instants and therefore a more accurate metamodel than with alternative selection methods for time instants. The DBN metamodeling tool developed in this thesis allows simulation practitioners unfamiliar with DBNs to take full advantage of the DBN metamodels through its intuitive user interface.

References

[1] Banks, J. *Handbook of simulation.* John Wiley & Sons, 1998.

[2] Banks, J., and Carson, J. S. *Discrete-event system simulation.* Prentice Hall, 1984.

[3] Barton, R. R. Simulation metamodels. In *Proceedings of the 1998 Winter Simulation Conference* (1998), pp. 167–174.

[4] Bertsekas, D. P. *Dynamic Programming and Optimal Control.* Athena Scientific, 2007.

[5] Blanning, R. W. The construction and implementation of metamodels. *Simulation 24*, 6 (1975), 177–184.

[6] Blum, M., and Micali, S. How to generate cryptographically strong sequences of pseudorandom bits. *SIAM Journal on Computing 13*, 4 (1984), 850–864.

[7] Caughlin, D. Automating the metamodeling process. In *Proceedings of the 1997 Winter Simulation Conference* (1997), pp. 978–985.

[8] Caughlin, D., and Sisti, A. F. A summary of model abstraction techniques. In *Proceedings of the SPIE* (1997), vol. 3083, pp. 14–21.

[9] Cheng, J., Bell, D. A., and Liu, W. Learning belief networks from data: An information theory based approach. In *Proceedings of the Sixth International Conference on Information and Knowledge Management CIKM97* (1997), pp. 325–331.

[10] Decision Systems Laboratory. GeNIe. Available via `http://genie.sis.pitt.edu/`, 2012.

[11] Decision Systems Laboratory. SMILE. Available via `http://genie.sis.pitt.edu/`, 2012.

[12] Faber, V. Clustering and the continuous k-means algorithm. *Los Alamos Science 22* (1994), 138–144.

[13] Friedman, L. W. *The simulation metamodel.* Kluwer Academic Publishers, 1996.

[14] Friedman, L. W., and Pressman, I. The metamodel in simulation analysis: Can it be trusted? *The Journal of the Operational Research Society 39*, 10 (1988), 939–948.

[15] Friedman, N., Linial, M., Nachman, I., and Pe'er, D. Using Bayesian networks to analyze expression data. *Journal of Computational Biology 7*, 3–4 (2004), 601–602.

[16] Gluss, B. Further remarks on line segment curve-fitting using dynamic programming. *Communications of the AMC 5*, 8 (1962), 441–443.

[17] Gratz, L. F. S. Z. S., and Bousquet, S. G. *The Art and Practice of Statistics*. Cengage Learning, 2008.

[18] Hartigan, J. A., and Wong, M. A. A *k*-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics) 28*, 1 (1979), 100–108.

[19] Heckerman, D. A tutorial on learning with Bayesian networks. Tech. Rep. MSR-TR-95-06, Microsoft Research, 1995.

[20] Hoad, K., Robinson, S., and Davies, R. Automated selection of the number of replications for a discrete-event simulation. *Journal of the Operational Research Society 61*, 11 (2010), 1632–1644.

[21] Hoad, K., Robinson, S., and Davies, R. Automating warm-up length estimation. *Journal of the Operational Research Society 61*, 9 (2010), 1389–1403.

[22] Jensen, F. V. *Bayesian Networks and Decision Graphs*. Springer, 2001.

[23] Kayaalp, M., and Cooper, G. F. A Bayesian network scoring metric that is based on globally uniform parameter priors. In *Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence* (2002), UAI'02, Morgan Kaufmann Publishers Inc., pp. 281–258.

[24] Kleijnen, J. P. C. Kriging metamodeling in simulation: A review. *European Journal of Operational Research 192*, 3 (2009), 707–716.

[25] Kleijnen, J. P. C., and Deflandre, D. Validation of regression metamodels in simulation: Bootstrap approach. *European Journal of Operational Research 170*, 1 (2006), 120–131.

[26] Kleijnen, J. P. C., and Sargent, R. G. A methodology for fitting and validating metamodels in simulation. *European Journal of Operational Research 120*, 1 (2000), 14–29.

[27] L-3 Communications Analytics Corporation. The X-Brawler air combat simulator management summary, 2002.

[28] Law, A. M., and Kelton, W. D. *Simulation Modeling and Analysis*. McGraw-Hill, 2000.

[29] MathWorks. MATLAB. Available via `http://www.mathworks.com/`, 2012.

[30] MathWorks. SimEvents. Available via `http://www.mathworks.com/products/simevents/`, 2012.

[31] Mattila, V., Virtanen, K., and Raivio, T. Improving maintenance decision-making in the Finnish air force through simulation. *Interfaces 38*, 3 (2008), 187–201.

[32] Murphy, K. P. *Dynamic Bayesian Networks: Representation, Inference and Learning*. PhD thesis, University of California, Berkeley, 2002.

[33] Neapolitan, R. E. *Learning Bayesian Networks*. Pearson Prentice Hall, 2004.

[34] Poropudas, J. *Bayesian networks, influence diagrams, and games in simulation metamodeling*. PhD thesis, Aalto Univeristy School of Science, 2011.

[35] Poropudas, J., Pousi, J., and Virtanen, K. Multiple input and multiple output simulation metamodeling using Bayesian networks. In *Proceedings of the 2011 Winter Simulation Conference* (2011), pp. 569–580.

[36] Poropudas, J., and Virtanen, K. Analyzing air combat simulation results with dynamic Bayesian networks. In *Proceedings of the 2007 Winter Simulation Conference* (2007), pp. 1370–1377.

[37] Poropudas, J., and Virtanen, K. Influence diagrams in analysis of discrete event simulation data. In *Proceedings of the 2009 Winter Simulation Conference* (2009), pp. 696–708.

[38] Poropudas, J., and Virtanen, K. Simulation metamodeling in continuous time using dynamic Bayesian networks. In *Proceedings of the 2010 Winter Simulation Conference* (2010), pp. 935–946.

[39] Poropudas, J., and Virtanen, K. Simulation metamodeling with dynamic Bayesian networks. *European Journal of Operational Research 214*, 3 (2011), 644–655.

[40] ProModel Corporation. ProModel optimization suite. Available via `http://www.promodel.com/`, 2011.

[41] Rockwell Automation. Arena. Available via `http://www.arenasimulation.com/`, 2012.

[42] Rubin, D., Burnside, E., and Shachter, R. A Bayesian network to assist mammography interpretation. *International Series in Operations Research & Management Science 70*, 4 (2005), 695–720.

[43] Rubinstein, R. Y., and Kroese, D. P. *Simulation and the Monte Carlo Method*. Wiley-interscience, 2007.

[44] Spirtes, P., Glymour, C., and Scheines, R. *Causation, Prediction, and Search*. The MIT Press, 2000.

[45] Talbi, E.-G. *Metaheuristics: From Design to Implementation*. John Wiley and Sons, 2009.

[46] van Koten, C., and Gray, A. R. An application of Bayesian network for predicting object-oriented software maintainability. *Information and Software Technology 48*, 1 (2006), 59–67.

[47] Virtanen, K. *Optimal Pilot Decisions and Flight Trajectories in Air Combat*. PhD thesis, 2005, Helsinki University of Technology.

[48] Zeigler, B. P., Praehofer, H., and Kim, T. G. *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*. Academic Press, 2000.