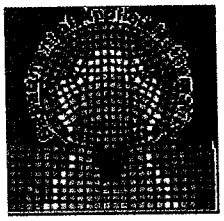
VŠB - Technical University of Ostrava, Czech Republic



RISK, QUALITY AND RELIABILITY

Edited by Radim Briš

2007

ISBN 978-80-248-1575-6

Bayesian belief networks for safety analysis and their enhancement with rough sets

A.J. Kornecki
Embry-Riddle Aeronautical University, Daytona Beach, FL 32114, USA

H.L. Pfister
Air Force Research Lab, Eglin AFB, FL 32542, USA

J. Zalewski
Florida Gulf Coast University, Ft. Myers, FL 33965, USA

ABSTRACT: System safety is traditionally dealt with in the design phase, to make sure that there are no design flaws that would jeopardize the users or the environment during system operation. However, in real-time safety-critical systems, where the system operation is controlled by software, as in modern aircraft or other kinds of vehicles, it is essential to monitor system's behavior with respect to safety during its execution and take safety related decisions while the system is running. There exist no good, if any, tools for this kind of tasks. The objective of this work is to study the potential use of Bayesian belief networks to reason about safety, with enhancements via rough sets for calculations of probability distributions for nodes with insufficient or uncertain information.

1 INTRODUCTION

System safety is traditionally dealt with in the design phase, to make sure that there are no design flaws that would jeopardize the users or the environment during system operation. However, in real-time safety-critical systems, where the system operation is controlled by software, as in modern aircraft or other kinds of vehicles, it is essential to monitor system's behavior with respect to safety during its execution and take safety related decisions while the system is running. Assessing software quality in safety-related applications of that sort has always been a problem, due to insufficient information and uncertainty associated with the available data.

There exist no good, if any, tools for this kind of tasks, mostly because of a lack of long term statistical data on which the assessment can be based. In other words, the problem is related to insufficient information and uncertainty associated with the available data. However, in the last decade, there have been a number of publications related to the assessment of various quality attributes of software for critical applications, relying on non-statistical data, with the use of Bayesian Belief Networks (BBNs) [1-4].

While this seems to be a viable approach, it suffers from the problem that in the canonical form BBNs require extensive set of prior conditional probabilities for each network node. Since typically little information is available on prior system behavior, it is difficult to initialize and keep updating the BBN, when there is uncertainty in evidence.

A few promising techniques that seem to help alleviate this problem are: fuzzy sets, neural networks, and rough sets. While fuzzy sets and neural networks are pretty well known, rough sets theory is a relatively new technique describing quantitatively uncertainty and vagueness [5]. The objective of this work is to study the potential use of Bayesian belief networks enhanced with rough sets for calculations of probability distributions for nodes with insufficient or uncertain information. The idea is that rough sets techniques have much less stringent requirements on prior probability distributions and use the outcome of the analysis to facilitate the process of initialization and updating of the BBN.

The rest of the paper is organized as follows. In Section 2, we present briefly the review of known works on the application of Bayesian belief networks to software quality assessment. Section 3 outlines the general problem of tool assessment in real-time safety-critical systems, and presents the application of BBNs to numerical evaluation of such tools. Section 4 outlines the idea of rough sets and their use to enhance computations of BBNs. Conclusions of the study are presented in Section 5.

2 SOFTWARE ASSESSMENT WITH BBN'S

A Bayesian belief network essentially relies on applying a Bayesian inference to a graphical representation of a problem in a form of a hierarchical net-

work [6], with nodes representing random variables and directed arcs representing probabilities (beliefs) about the dependencies (relationships) among these variables. Formally speaking, a network takes the form of a *directed acyclic graph*. Each node has a number of states, with some probability distribution over the states. Each arc represents a conditional probability reflecting dependencies on the predecessor nodes. In the simplest case, the states of a node could be just *true* or *false*. In a more complex case, the set of states may involve several discrete states (for example, *low*, *medium*, *high*, *very high*), and in the most general case, the domain of states can be continuous.

A BBN is intended to show the causal relationships among variables and allows, for such relationships, to conduct inference about changes in probability distributions of certain variables (parameters), once values of some observable variables become known. In other words, based on some knowledge of events (that is, evidence) acquired at leaf nodes, which changes their probability distribution, applying Bayesian reasoning one can deduct what might have caused such events to occur. This is called backward reasoning, and a specific example of a BBN is shown in Figure 1 for some evidence obtained for node D, which causes changes in probability distributions of other nodes depending on conditional probability distributions. One can also do forward reasoning, in case respective evidence becomes available for the root node(s).

This kind of reasoning is based on Bayes Theorem and is normally cumbersome and time consuming, if done by hand, but can be easily automated with software tools that became available throughout the 1990's. There is quite a few BBN tools to choose from, but the most commonly applied in software quality related problems are: Hugin [7], Netica [8], with which these diagrams have been produced, BUGS [9], and AgenaRisk [10].

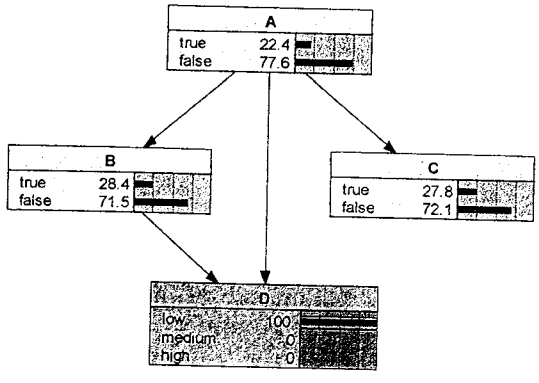


Figure 1. Demonstration of an impact of evidence on the predecessor variables.

In building the BBN model, there are always two questions that have to be addressed: (1) how to organize the network of nodes and links for a specific problem, and (2) what probabilities assign to them. Below, we discuss briefly how this is done in some of the studies related to software assessment published over the past decade.

2.1 *Application of BBN's to software quality in general*

In one of the first studies reported [1], Neil and Fenton addressed the eternal question: "Can we predict the quality of our software *before* we can use it?", by applying BBN's to assess the *defect density* as a measure of software quality. A simplified diagram from their study is presented in Figure 2. The nodes were built based on the understanding of life-cycle processes, from requirements specification through testing.

The probabilities of respective states were based on the analysis of literature and common-sense assumptions about the relations between variables. The node variables are shown on histograms of the predictions obtained by execution of the network after the evidence entered (the evidence is represented by nodes with probabilities equal to 1.0). As the authors say, the advantage of their model is that it "provides a way of simulating different events and identifying optimum courses of action based on uncertain knowledge.

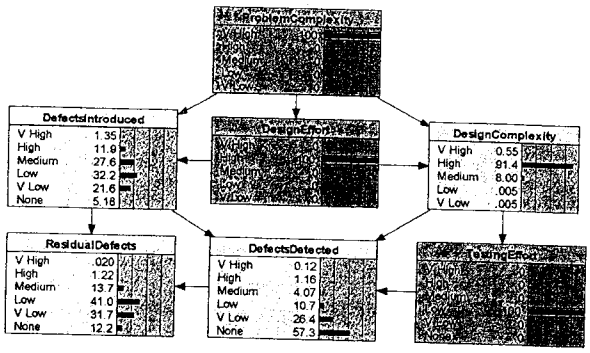


Figure 2. A simplified BBN model for assessing software density [1].

2.2 *Use of BBN's in assessment of software safety*

Dahll and Gran [2] applied BBN's to address safety assessment of software for acceptance purposes, in a more comprehensive way, using multiple information sources, such as complexity, testing, user experience, system quality, etc. Their BBN network for system quality, which is only a part of the entire model, is shown in Figure 3. It involves two root nodes: *UserExperience* and *VendorQuality*, and a

number of leaf nodes, corresponding to observable variables, of which *QualityMeasures* is of particular importance. This node shows evidence about the system quality, grouping quality attributes, such as readability, structuredness, etc., and can be expanded further.

Other observable variables include *FailuresInOther Products*, those related to the user experience (*NoOfProducts* and *TotalUseTime*), as well as those related to quality assurance policy. When evidence becomes available, entering respective observation data into these nodes and executing the network provides assessment of the variable in question, which in this case is *SystemQuality*.

The authors note, however, that their example is intended more as an illustration of the method rather than as a real attempt to compute the quality of the system. Their probability assignments to the node variables were chosen somewhat *ad hoc*, and not based on any deeper analysis of the problem. However, as the authors say in conclusion, the results of the study were positive and showed "that the method reflects the way of an assessor's thinking during the assessment process."

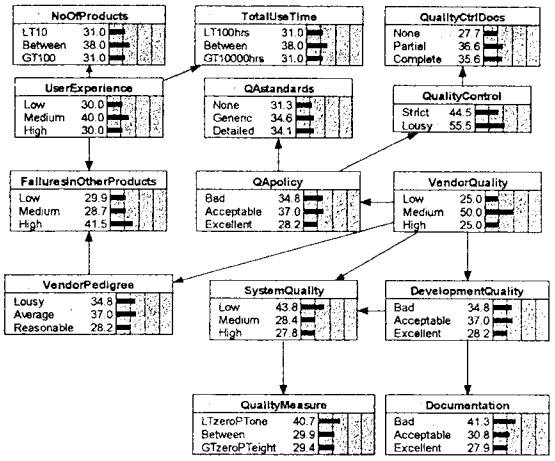


Figure 3. BBN for the system quality parameter in safety assessment.

2.3 *BBN's in dependability/reliability assessment*

Delic et al. [3] used BBNs to formalize reasoning about software dependability to facilitate the software assessment process. They constructed a network for evaluating dependability of a software-based safety system. It used the data associated with two primary assumptions: the excellence in development (called a process argument) and failure-free statistical testing (called a product argument). The network topology includes taking into consideration variables such as: Test Failures, Operational Failures, Initial Faults, Faults Found, Faults Delivered, and System PFD (Probability of Failure per Demand). The probability distributions have been derived from a sample of programs from an academic experiment.

The authors were interested in estimating the probabilities of failure during acceptance testing and during the operational life of the product (represented by two variables mentioned in previous paragraph), given the prior probabilities and observed events. In particular, positive results of an acceptance test allowed deriving numerical estimates about the PFD and operational performance of the product.

Helminen [4] used BBN's to attack the problem of software reliability estimation. His primary motivation to apply BBN's was that they allow all possible evidence (large number of variables, different potential sources, etc.) to be used in the analysis of the reliability of a programmable safety-critical system. The essential characteristic of such systems is that they involve a significant number of variables related to reliability, with very limited evidence.

The reliability of such systems is modeled as a *probability of failure*, that is, the probability that the programmable system fails when it is required to operate correctly. To develop an estimate of probability of failure, the authors built a series of BBN models, using evidence from such sources, as the system development process, system design features, and pre-testing, before the system is deployed. This is later enhanced by data from testing and operational experience.

The essential part of this work was building BBN models for various operational profiles for multiple test cycles, involving continuous probability distributions. As a result, using BUGS software that combines Bayesian inference with Gibbs sampling [9], via Markov chain Monte Carlo (MCMC) simulation, it was possible to estimate, how many tests had to be run for a single system in a particular operational environment to achieve certain level of reliability. To decrease the huge number of necessary tests, multiple operational profiles for the same system were used, which required building replicated BBN models to include other profiles' evidence. In essence, by expanding the BBN models further, this approach also allows reliability estimation over the entire lifespan of the software product, but respective experiments have not been conducted in this study.

3 PRELIMINARY EXPERIMENT IN SOFTWARE ASSESSMENT

To test the applicability of BBNs in software assessment, we applied this technique to evaluate the software development tools used in real-time safety-critical applications in avionics. The data for the project were taken from experiments described in

detail elsewhere [11]. The experiments involved applying a number of specific criteria, including: *efficiency* of the generated code, to conduct forward evaluation regarding the quality of code, and *traceability,* to allow backward evaluation regarding the tool capability of maintaining the right requirements. To evaluate the tool during its operation from perspective of the functions it provides and the ease of use, two additional criteria seemed to be appropriate: *functionality* and *usability.* The exact process of choosing criteria is described in [11].

For criteria selected that way, a series of experiments were conducted, with six industry-strength tools applied to embedded software development. The above mentioned criteria were quantified using the following measures:

- *Efficiency* measured as code size (in LOC)
- *Usability* measured as development effort (in hours)
- *Functionality* measured via the questionnaire (on a 0-5 points scale)
- *Traceability* measured by manual tracking (in number of defects).

Data for some measures were collected in multiple aspects, for example, data involving the development effort were divided into four categories: preparation, modeling and code generation, measurements, and postmortem (including report writing). Details of the software requirements and actual experimental results are discussed in [11].

Based on the adopted model of the tool evaluation process, and the results of experiments with the selected evaluation criteria outlined above, our high-level model of a BBN for tool assessment is illustrated in Figure 4. Its primary assumptions are that the tool assessment process should involve the following mutually interrelated factors:

- development of the tool itself (including the process, vendor quality and reputation, their quality assessment procedures, etc.)
- the tool use (including experimental evaluation based on predefined criteria, but also previous user experiences with this tool, etc.)
- quality of the products developed with this tool, based on product execution, static code analysis, etc.

Based on the results of this analysis and other acceptance procedures (such as, legal aspects, independent experts opinions, etc.), the tool qualification process can be completed, as reflected in a BBN in Figure 4.

Because of the limited data obtainable from experiments, we only deal with *ToolUse* part of the diagram in Figure 4. The logic of the BBN is similar to the ones reported by Dahll and Gran [2], where they had no real probability data, and Gran and Dahll [12], where the conditional probability values "were estimated based on judgments in a brainstorming activity among the project participants".

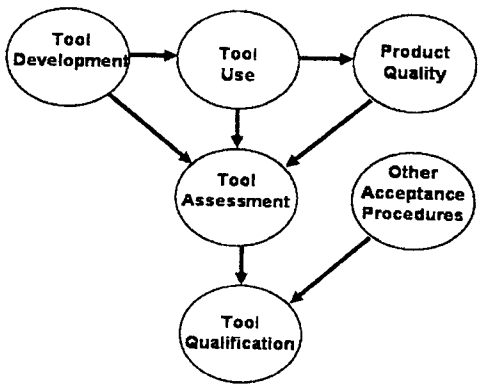


Figure 4. High-level BBN model for software tool evaluation..

For the experimentally collected data for six tools, nicknamed L, M, N, O, P and Q, a sample tool assessment BBN is shown in Figure 5 for a tool, which is likely to pass the qualification process with 80% confidence at the level *MediumToHigh* or *High.*

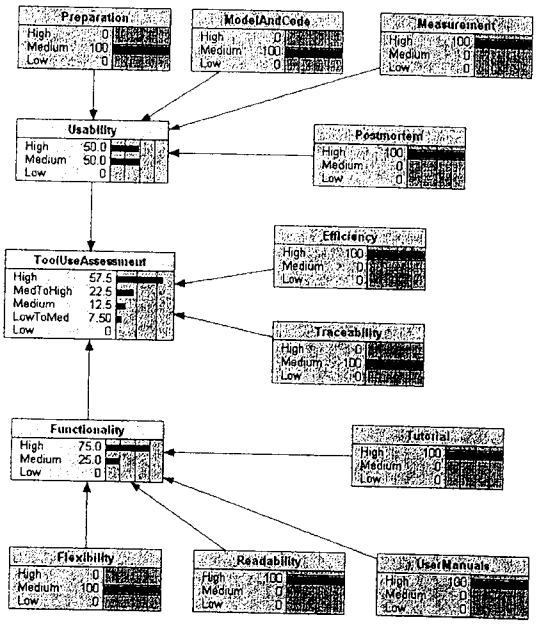


Figure 5. BBN to assess numerically quality of tool L.

4 IMPROVING THE ASESSMENT USING ROUGH SETS

The BBN technique seems to work well for the assessment projects, where there is plenty of experimental data available and the new evidence can be easily quantified. But if safety analysis needs to be conducted continuously during system operation, there may be too many unknowns or two little evidence to justify the validity of a Bayesian approach, which relies on the confidence in conditional probabilities.

4.1 *Rough sets theory*

To deal with these situations, we adopt a rough sets theory, which is a mathematical technique developed to describe quantitatively uncertainty, imprecision, and vagueness [5]. An important result from the rough sets theory is that it simplifies the search for dominating attributes leading to specific properties. Rough sets are thus suited for the analysis of the imprecise data, to evaluate the system's operation in terms of risks or uncertainty, or to how does it accomplish the mission. The rough sets theory is suitable for dealing with problems such as noise, unknown values, or errors due to inaccurate measuring equipment. It is a tool for handling vagueness and uncertainty inherent to decision processes. At the same time, the rough sets theory is mathematically relatively simple with a promise of fast run-time (and potential real-time) implementation.

The idea of a rough set is that unlike a conventional set, which has sharp boundaries, or a fuzzy set, which has vague boundaries, it is described by two approximations: the lower approximation and the upper approximation. Assuming that **U** is the domain of discourse, **R** is the relation of the sets known as knowledge base, $\mathbf{X} \subseteq \mathbf{R}$ is the rough set, the lower approximation of a set, $\mathbf{R_L X}$, is the set of all elements belonging to it with certainty, as in the following formula:

$$\mathbf{R_L X} = \cup\{\mathbf{Y} \in \mathbf{U} \mid \mathbf{R}: \mathbf{Y} \subseteq \mathbf{X}\}$$

while an upper approximation of a set, $\mathbf{R_U X}$, is a set of all elements that cannot be excluded from it with certainty:

$$\mathbf{R_U X} = \cup\{\mathbf{Y} \in \mathbf{U} \mid \mathbf{R}: \mathbf{X} \neq \varnothing\}$$

Relating this to the concept of uncertainty, one can say that the lower approximation, $\mathbf{R_L X}$, represents the certainty that the system has the investigated property, and the upper approximation, $\mathbf{R_U X}$, refers to the possibility that a system has the investigated property. Oversimplifying, if we are able to represent the system properties in terms of rough sets, taking the ratio of $\mathbf{r} = \mathbf{R_L X}/\mathbf{R_U X}$ would be an "indicator" whether the system has the investigated property or not. The measurement process will give us certain confidence if the value of **r** is close to 1. On the other hand, if the value of **r** is far from 1, the statement that the system has the studied property is questionable.

Rough sets theory has been successfully applied in a variety of decision-making processes ranging from medical data analysis, to aircraft pilot performance evaluation, to image processing, to voice recognition [13]. The authors' past work also involves using the concept of rough sets to evaluate safety of the software development processes [14].

4.2 *Application of rough sets*

The key issue in using rough sets for enhancement of reasoning using BBN's is to handle uncertainty. This issue comes into play when there is no information on certain behavior or some information previously available becomes scarce or unavailable. Using a rough set can help filling the gap caused by such circumstances. To illustrate this concept, we present a simple example using the rough sets tool named Rosetta [15].

The example concerns the determination of the type of a flag (*Union* or *Confederate*), based on values of some primary condition attributes of the flag (*Stars*, *Bars*, *Stripes*, *Hues*, etc.), as illustrated in Figure 6. *Type* is a decision attribute in rough sets terminology. Taking only two attributes for consideration, for example *Stars* and *Bars*, one can create equivalence classes (called *reducts*, in rough sets parlance) based on indiscernibility relation for flag objects in Figure 6. This can lead to the rough set approximation of the *Union* flags (for details, see [15] or [5]).

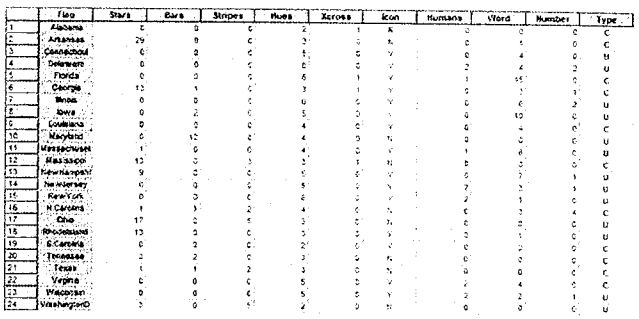
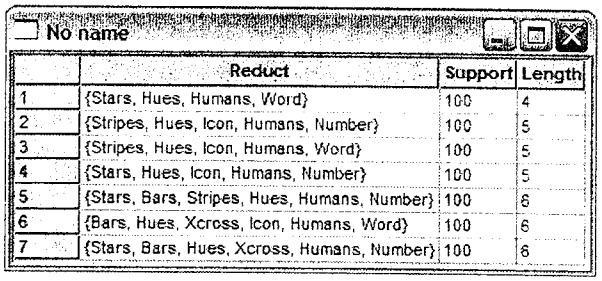


Figure 6. Example to determine the value of decision attribute based on condition attributes [15].



No name

	Reduct	Support	Length
1	{Stars, Hues, Humans, Word}	100	4
2	{Stripes, Hues, Icon, Humans, Number}	100	5
3	{Stripes, Hues, Icon, Humans, Word}	100	5
4	{Stars, Hues, Icon, Humans, Number}	100	5
5	{Stars, Bars, Stripes, Hues, Humans, Number}	100	6
6	{Bars, Hues, Xcross, Icon, Humans, Word}	100	6
7	{Stars, Bars, Hues, Xcross, Humans, Number}	100	6

Figure 7. Equivalence classes discerning Union flags [15].

In particular, if all the condition attributes are used to approximate this set, then it will be approximated with perfect accuracy (the upper and lower bounds are the same). However, one can still find subsets of these attributes that will not alter the equivalence classes (reducts) and approximate the *Union* flag with 100% accuracy. The resulting sets

are presented in Figure 7, where *support* is the rough sets term indicating the level of accuracy.

What this means to the BBN's is the following. If we treat specific variables from the BBN network as attributes of the rough set, with one of them being the decision attribute and all remaining ones – condition attributes, then we can determine (with some level of accuracy) the unavailable value of the decision attribute, using the reasoning just presented briefly and described in more detail by Pawlak [5]. In plain language, this would be equivalent to deriving the approximate value of a certain variable based on the context information.

This would help in making BBNs more valuable in case of the lack of evidence. It becomes particularly important, when BBNs are used in active safety systems, with information being supplied to the nodes during operation, because losing the source of information for one of the BBN nodes impairs the inference process in the next steps. Using rough set reasoning could help in keeping the BBN in good standing, disregarding the lost source of information, because new information coming to other nodes would be used to approximate the value of the missing variable. This logic is very similar to the use of a Kalman filter, when the information about the system is updated based on its previous behavior, however, in case of rough sets the information does not have a statistical nature, as in the case of Kalman filtering.

Such process can be easily automated with existing tools, since a Netica version exists that has a Java API and can read cases from a text file, and Rosetta can export its tables as text files to be grabbed by Netica. With an interface software reading Rosetta files and converting them to the Netica format, the whole system looks like in Figure 8.

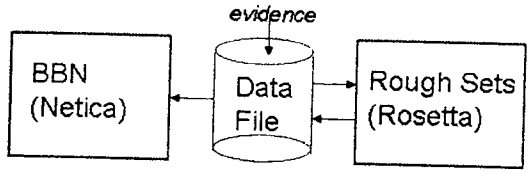


Figure 8. An architecture of enhancing BBN's with rough sets.

5 CONCLUSION

This study proves the applicability of Bayesian belief networks (BBN's) as a support tool for the numerical assessment of software development tools in real-time safety-critical applications. They can be used as a part of the tool qualification process, as well as play a supportive role in making decisions on certification in certain industries. In addition, a method of enhancing BBNs with rough sets, to derive contextual information in case of uncertainty, seems viable and beneficial, and may find application in situations when real-time updates in safety analysis are necessary.

REFERENCES

1. Neil M., N. Fenton, Predicting Software Quality Using Bayesian Belief Networks, *Proc. SEW-21, Annual NASA Goddard Software Engineering Workshop*, December 4-5, 1996, pp. 217-230.
2. Dahll G., B.A. Gran, The Use of Bayesian Belief Nets in Safety Assessment of Software Based Systems, *Int. Journal of General Systems*, Vol. 29, No. 2, pp. 205-229, 2000
3. Delic K.A., F. Mazzanti, L. Strigini, Formalising Engineering Judgement on Software Dependability via Belief Networks, *Proc. DCCA-6, 6th IFIP Int. Working Conf. on Dependable Computing for Critical Applications*, M. Dal Cin, C. Meadows, W.H. Sanders, (Eds.), IEEE Computer Society, 1998, pp. 291-305
4. Helminen A., *Reliability Estimation of Safety-Critical Software-Base Systems Using Bayesian Networks*, Report STUK-YTO-TR 178, Radiation and Nuclear Safety Authority, Helsinki, June 2001.
5. Pawlak Z., *Rough Sets: Theoretical Aspects of Reasoning about Data*, Kluwer Academic Publishers, Dordrecht, 1991
6. Jensen F.V., *An Introduction to Bayesian Networks*, Springer-Verlag, New York, 1996
7. *Hugin Explorer*, Hugin Expert A/S, Aalborg, Denmark, URL: `http://www.hugin.com/`
8. *Netica*, Norsys Software Corporation, Vancouver, Canada, URL: `http://www.norsys.com/`
9. *BUGS*, MRC Biostatistics Unit, Cambridge, UK, `http://www.mrc-bsu.cam.ac.uk/bugs/welcome.shtml`
10. *AgenaRisk*, Agena Limited, London, UK, URL: `http://www.agena.co.uk/`
11. Kornecki A., J. Zalewski, Experimental Evaluation of Software Development Tools for Safety-Critical Real-Time Systems. *NASA Journal on Innovations in Systems and Software Engineering*. Vol. 1, No. 2, pp. 176-188, September 2005
12. Gran B.A., G. Dahll et al., *Estimating Dependability of Programmable Systems Using BBNs*, Proc. SAFECOMP 2000, 19th Intern. Conf. on Computer Safety, Reliability and Security, Springer-Verlag, Berlin, pp. 309-320
13. Düntsch I., G. Gediga. *Rough Set Data Analysis: A Road to Non-invasive Knowledge Discovery*. Methodos Publishers, Bangor (UK), 2000
14. I.E. Chen-Jimenez, A. Kornecki, J. Zalewski, Software Safety Analysis Using Rough Sets, *Proc. IEEE SOUTHEASTCON'98*, IEEE Press, 1998, pp. 15-19
15. *Rosetta*, The Linnaeus Centre for Bioinformatics, Uppsala University, Sweden, URL: `http://rosetta.lcb.uu.se/general/download`