Root Cause Analysis through Abduction in Markov Logic Networks

Joerg Schoenfisch[1], Janno von Stülpnagel[2], Jens Ortmann[2], Christian Meilicke[1], and Heiner Stuckenschmidt[1]

[1] Research Group Data and Web Science, University of Mannheim, Germany
{joerg,christian,heiner}@informatik.uni-mannheim.de
[2] Softplant GmbH, Munich, Germany
{janno.stuelpnagel,jens.ortmann}@softplant.de

Abstract. In this paper we propose an approach for calculating the root cause for an observed failure in an IT infrastructure. Our approach is based on Markov Logic Networks. While Markov Logic supports a special type of deductive inference, known as maximum a posteriori inference, the computation of the most probable cause requires abductive reasoning. Abduction aims to find an explanation for a given observation in the light of some background knowledge. In failure diagnosis, the explanation corresponds to the root cause, the observation corresponds to the failure of a component or service, and the background knowledge corresponds to the dependency graph of the infrastructure extended by potential risks. We apply the method for abduction proposed by Kate et al. to extend a Markov Logic Network in order to conduct abductive reasoning [1]. We illustrate that our approach is a well suited method for root cause analysis by applying it to a sample scenario.

Key words: Root Cause Analysis, IT Infrastructure, Markov Logic Networks, Abductive Reasoning

1 Introduction

Root cause analysis (RCA) plays an important part in processes for problem solving in many different settings. Its purpose is to find the underlying source of the observed symptoms of a problem. Especially in IT, short response times to failures (e.g. inaccessible websites, or unresponsive accounting systems) are crucial. Today's IT infrastructures are getting increasingly complex with diverse explicit and implicit dependencies. This makes RCA a time intensive task as the cause for a problem might be unclear or the most probable cause might not be the most obvious one. Therefore, automating the process of RCA and helping an IT administrative to identify the source of a failure or outage as fast as possible is important to achieve a high service level.

In this paper we present our approach to root cause analysis that uses Markov Logic Networks (MLN) and abductive reasoning to enable an engineer to drill down fast on the source of a problem. Markov Logic Networks provide a formalism that combines logical formulas (to describe dependencies) and probabilities (to express various possible risks) in a single representation. We focus on abductive reasoning in MLNs and show how it can be used for the purpose of RCA. To our knowledge, the proposed approach is a novel method to RCA that combines probabilistic and logical aspects in a well-founded framework.

We represented the IT infrastructure as a logical dependency network that includes threats to its components. When a problem occurs, observations are fed into the system which then generates the MLN from the available observations, the given dependency network, and the general background knowledge related to the components of the infrastructure. Some of these observations might be specified manually, while other observations can be generated automatically via constantly running monitoring software. Typically they are incomplete in the sense that not all relevant components are monitored. Thus, there might still be several explanations for the problem that occurred.

We calculate, via abduction, the most probable cause(s) for the current problem, which is then presented to the user, e.g. the administrator of the IT infrastructure. The user can then investigate if it is indeed the source of the problem. If the proposed explanation is correct, counter-measures can be introduced immediately. If the additional observations revealed that the calculated explanation is wrong, the new observations are fed into the system as additional evidence and a better explanation is computed. This iterative, dialogue-based process is a practicable approach to quickly narrow down on a root cause.

In our approach, we represent the infrastructure and the possible risks in first-order logic. This allows us to automatically infer that certain threats are relevant for certain components. Relevant background knowledge can easily be maintained and used to generate the MLN. Moreover, our approach can take into account known probabilities of risks and failures. These probabilities are derived from expert judgment or statistical data. Instead of computing multiple candidate explanations, which is done in purely logic based approaches, we are able to generate the most probable explanation with our approach, while still leveraging the full power of an expressive declarative framework.

2 Preliminaries

2.1 Markov Logic Networks

MLNs generalize first-order logic and probabilistic graphical models by allowing hard and soft first-order formulas [2]. Hard formulas are regular first-order formulas, which have to be fulfilled by every interpretation. An interpretation is also referred to as a possible world. Soft formulas have weights that support (in case of positive weights) or penalize (in case of negative weights) worlds in which they are satisfied. The probability of a possible world, one that satisfies all hard formulas, is proportional to the exponential sum of the weights of the soft formulas that are satisfied in that world. This corresponds to the common understanding of Markov Networks as log-linear probabilistic model [2].

An MLN is a template for constructing a Markov Network. A formula is called a grounded formula if all variables have been replaced by constants. Given a set of constants, a Markov Network can be generated from the MLN by computing all possible groundings of the given formulas. Due to the closed world assumption, the domain of interest consists of only those entities that are defined by specifying the set of constants. An atom is a formula that consists of a single predicate. A possible world corresponds to a set of ground atoms, which is usually a small subset of all possible groundings.

An MLN L is a set of pairs $\langle F_i, w_i \rangle$, where F_i is a first-order logic formula and w_i is a real numbered weight. The description as a log-linear model leads to the following definition for the probability distribution over possible worlds x for the Markov Network $M_{L,C}$, with Z being a normalization constant and $n_i(x)$ the number of true groundings of F_i in x (for a more complete formal description see [2, p.113]):

$$P(X = x) = \frac{1}{Z} \exp\left(\sum_i w_i n_i(x) \right) \tag{1}$$

When describing the MLN we use the format $\langle$*first-order formula*, *weight*$\rangle$. Hard formulas have infinite weights. If the weight is $+\infty$ the formula must always be true, if the weight is $-\infty$ it must always be false. A soft formula with weight 0 has equal probabilities for being satisfied in a world or not.

There are two types of inference with Markov Logic: maximum a posteriori (MAP) inference and marginal inference. MAP inference finds the most probable world given some evidence. It does not compute probabilities of variables but the world with the highest overall probability for its variable assignment. Marginal inference computes the posteriori probability distribution over the values of all variables given some evidence. In other words, it calculates the sum of the probabilities of all the worlds in which a given variable is true. Note that the single most probable event does not have to be include in the overall most probable world. We are interested in MAP inference, as we want to determine the most probable world containing an explanation for a failure.

2.2 Abduction in Markov Logic Networks

Abductive reasoning – or simply *abduction* – is inference to the best explanation. It is applicable to a wide array of fields in which explanations need to be found for given observations, for example plan or intent recognition, medical diagnosis, criminology, or, as in our approach, root cause analysis. According to [1], abduction is usually defined as follows [3]:

Given: Background knowledge B and a set of observations O, both formulated in first-order logic with O being restricted to ground formulae.

Find: A hypothesis H, also a set of logical formulae, such that $B \cup H$ is consistent and $B \cup H \vdash O$.

In other words, find a set of assumptions (a hypothesis) that is consistent with the background knowledge and, combined with it, explains the observation. It is the opposite of deductive reasoning which infers effects from cause.

The relation between root cause analysis and abductive reasoning is rather straightforward. In our approach, the background knowledge is the dependency network, respectively the Markov Logic Network to which we transform it. The dependency graph and Markov Logic Networks both are based on first-order logic as a formalism and thus conveniently are already in the desired logical representation. The observations, i.e. information about components being available or unavailable, are not part of the model but rather are directly provided as evidence to the MLN. We then try to prove through abduction that a specific threat – the most plausible cause – has occurred.

The inference mechanism in Markov Logic Networks is inherently deductive, not abductive. Deductive reasoning draws new, logically sound conclusions from given statements. Kate et al. and Singla et al. [1, 4] proposed methods – Pairwise Constraint (PC) and Hidden Cause (HC) model – that adapt Markov Logic Networks to automatically perform probabilistic abductive reasoning through its standard deductive reasoning mechanism. Their method augments the clauses of the MLN to support abductive reasoning as defined above. In general, the methods first introduce a reverse implication for every logical implication already present in the network. For example, if there are formulas $p_1 \rightarrow q$, …, $p_n \rightarrow q$ in the MLN, the formula $q \rightarrow p_1 \vee \ldots \vee p_n$ is added to the MLN.

In a second step the model is then extended with mutual exclusivity constraints that bias the inference against choosing multiple explanations. The reverse implications and the mutual exclusivity clauses are modeled as soft rules and may occasionally be violated, for example, if multiple explanations provide a better proof for the hypothetical root cause than a single explanation. We follow this basic idea, however, we argue that the mutual exclusivity constraints are not required in the application that we are interested in.

3 Root Cause Analysis with Markov Logic Networks

Root cause analysis is the task of finding the underlying cause of an event. It is often applied to analyze system failures. System failures are commonly caused by a cascade of events. The goal of a root cause analysis is finding the original reason for the failure, so that a sustainable solution can be provided [5]. Root cause analysis typically comprises two phases: the detection of an event and the diagnosis of the event. In our work, we are concerned with the second phase and assume that a failure has already been detected.

3.1 Scenario Setting

In the subsequent sections we discuss our approach with the help of an infrastructure shown partially in Figure 1. This small sample revolving around an office multifunction printer consists of the following components.

The threats we are using in our example are defined in the *IT-Grundschutz Catalogues* [6, p. 417ff.]: They are a comprehensive collection of threats and

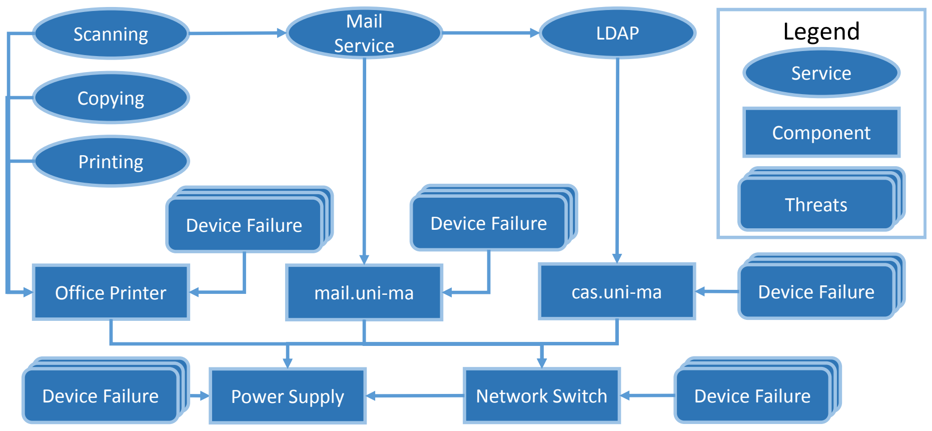

Fig. 1. Case Study: Office multifunction printer with risks attached

safeguards for various parts of an IT infrastructure; created and maintained by the German Federal Office for Information Security (BSI), and compatible to the ISO 27001 certification. For example threats like disruption of power supply, lack of resources, or malicious software are defined.

3.2 Modeling the Infrastructure

The foundation of our root cause analysis is the dependency model. It uses first-order logic to describe various aspects of the IT infrastructure. Our basic model uses five predicates:

specificallyDependsOn(*x*,*y*) specifies that component x is specifically dependent on component y, e.g. the mail service that runs on the mail server. This predicate does not allow for any redundancy of y.

genericallyDependsOn(*x*,*y*) specifies that component x depends on y. y may be replaced by some other redundant component. An example is a server running on the normal power supply or some uninterruptible power source.

redundancy(*x*,*y*) states that x and y are redundant and x can replace y.

hasRisk(*x*,*y*) assigns the risk y to component x, i.e. y is a threat that endangers the functionality of a component and it can affect x.

unavailable(*x*) designates a component x as unavailable, e.g. offline or not functioning properly.

Formulae 2a to 2f depict the basic MLN program built from those predicates:

$$\langle \forall x, y\ (\text{specificallyDependsOn}(x, y) \wedge \text{unavailable}(y) \Rightarrow \text{unavailable}(x)), \infty\rangle \tag{2a}$$

$$\begin{split}\langle \forall x, y\ (&\text{genericallyDependsOn}(x, y) \wedge \text{unavailable}(y) \\ &\wedge \neg\exists z\ (\text{redundancy}(y, z) \wedge \neg\text{unavailable}(z)) \Rightarrow \text{unavailable}(x)), \infty\rangle\end{split} \tag{2b}$$

$$\langle \forall x, y\ (\text{redundancy}(x, y) \Rightarrow \text{redundancy}(x, y)), \infty\rangle \tag{2c}$$

$$\langle \forall x, y\ (\text{redundancy}(x,y) \land \text{redundancy}(y,z) \Rightarrow \text{redundancy}(x,z)), \infty\rangle \tag{2d}$$

$$\langle \forall x, y\ (\text{affectedByRisk}(x,y) \Rightarrow \text{unavailable}(x)), \infty\rangle \tag{2e}$$

$$\langle \forall x, y\ \neg(\text{specificallyDependsOn}(x,y) \land \text{genericallyDependsOn}(x,y)), \infty\rangle \tag{2f}$$

Formula 2a forbids any world where infrastructure component x is unavailable and infrastructure component y is available, if there is a specific dependency from x to y. Formula 2b is similar to Formula 2a, but for generic dependencies with redundancies. Provided x is generically dependent on y, y is unavailable, and there exists no available component z that is redundant with y, then x is also unavailable. Thus, a component is only available if every specific dependency is available or if at least one redundant component is available for each generic dependency, respectively. The symmetry and transitivity of *redundancy* is modeled by Formulae 2c and 2d. By adding these two formulas, we ensure that it is not required to specify redundancy for all pairs in both directions. If we extend an infrastructure with an additional redundant component, we only need to add a single statement instead of specifying the information for all pairs in the group of redundant components. Formula 2e enforces that a component x that is affected by the effects of a risk y becomes unavailable. The predicates *specificallyDependsOn*(x, y) and *genericallyDependsOn*(x, y) are mutually exclusive (Formula 2f).

The known dependencies, risks, and unavailabilities are modeled as evidence as shown below. Note that these formulas are only two examples for all formulas required to describe the infrastructure depicted in Figure 1.

$$\langle \text{specificallyDependsOn}(\textit{MailService}, \textit{mail.uni-ma}), \infty\rangle \tag{3a}$$

$$\langle \text{affectedByRisk}(\textit{mail.uni-ma}, \textit{MaliciousSoftware}), -1.2\rangle \tag{3b}$$

Formula 3a is a hard fact, which states that the *MailService* depends on the server *mail.uni-ma*. The soft Formula 3b encodes that *mail.uni-ma* can be affected by *MaliciousSoftware*. This formula has a negative weight, i.e. has a low probability. As described before, the dependency relation must hold in every possible world. The soft formula is not fulfilled in most of the worlds. In fact, if only this evidence is given, the most probable world does not include it, as it lowers the sum of the weights of all formulas.

Determining the correct weight for the evidence is not trivial. However, there exist efficient learning algorithms for MLNs [2].

Our basic dependency model only contains relatively simple rules, and only soft formulas in the evidence. One way in which we extended the MLN program is by adding additional general knowledge about types of components.

For example, we can add information about the failure rate (in the form of a weight) of a specific hard drive model to our MLN program.

$$\langle \forall x\ (\text{SCSIHardDrive}(x) \Rightarrow \text{affectedByRisk}(x, \textit{HeadCrash})), -1.8\rangle \tag{4}$$

The hard drive model *SCSIHardDrive* is described as hard drive that has a certain risk of a head crash (4). The weight attached to this formula can be derived from available failure rates. If required we can also add further types related to, e.g. the manufacturer of the drive, since it might be known that drives produced by a certain company have a lower failure rate. We then can model individual drives as instance of this type. Subsequently, they inherit all the properties, i.e. the weighted risk of a head crash.

3.3 Computing Explanations

We now detail our approach and describe how the Markov Logic Network is constructed and extended, and how we use abductive reasoning for root cause analysis. The construction from background knowledge and extension for abduction of the Markov Logic Network is only done once and does not have to be changed during the root cause analysis. According to the method proposed in [1] we have to add one reverse implication for the Formulae 2a, 2b, and 2e:

$$\begin{aligned}\forall x\ (\text{unavailable}(x) \Rightarrow\ & (\exists y\ (\text{specificallyDependsOn}(x,y) \wedge \text{unavailable}(y))) \vee \\ & (\exists y\ (\text{genericallyDependsOn}(x,y) \wedge \text{unavailable}(y) \\ & \ \wedge \neg\exists z\ (\text{redundancy}(y,z) \wedge \neg\text{unavailable}(z)))) \vee \\ & (\exists y\ (\text{affectedByRisk}(x,y)))\end{aligned} \tag{5}$$

Additionally, Kate et al.s' method requires clauses for mutual exclusivity to be added. The purpose of these clauses is to *"explain away"* multiple causes for an observation and prefer a single one [7]. The reverse implications as well as the mutual exclusivity clauses are usually modeled as soft clauses. In general, for each set of reverse implications P_i with the same left-hand side, $(\frac{|P_i|^2+|P_i|}{2}) \in O(n^2)$ mutual exclusivity clauses are added.

However, different from networks in that general method, our approach exhibits a property that simplifies the additional rules needed for abduction: All the weights in the evidence are negative – based on the reasonable assumption that threats and risks only occur rarely, i.e. components are available more than 50% of the time. This property allows us to reduce the size of the Markov Logic Network by leaving out the mutual exclusivity clauses completely: Due to the reverse implication, the MLN solver has to chose one cause to make the clause *true*. However, as all causes have negative weights and thus every cause set to true is lowering the sum of the weights of a possible world, the solver is already biased against choosing multiple explanations. This saves us from generating the quadratic number of mutual exclusivity clauses.

After constructing and extending the Markov Logic Network, we can conduct the root cause analysis. The overall process flow of our approach is depicted in Figure 2. The analysis is a dialog-based and iterative process, with interaction between our system and an administrative user. A fully automatic workflow is desirable, however, not every information can be retrieved directly and sometimes manual investigation of log files or on the status of components is necessary.

In its normal state, without any hard evidence about availabilities or unavailabilities, all components are assumed to be available. Thus, when calculating the MAP state, it contains all components as available. When a problem occurs the

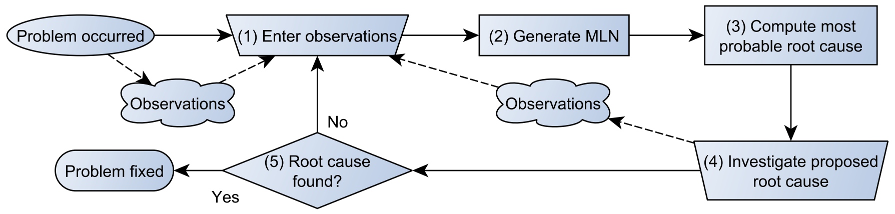

Fig. 2. Process flow for our approach on root cause analysis. Rectangles denote automatic action. Trapezoids require manual interaction by an administrative user.

user is required to provide observations as evidence for the MLN (1). These observations include any certain information about available and unavailable components. For example, the user can enter that printing over the network is not possible, although the network is functional as browsing the internet still works. This results in hard evidence for the printing service being unavailable and network services and hardware required for internet access being available.

Our approach extends the Markov Logic Network with the new evidence (2) and uses an MLN solver to run MAP inference on it (3). The calculated MAP state contains the evidence provided by the user (this must be always fulfilled), components being unavailable due to a direct or indirect dependency on components observed as not available, and (at least) one root cause that explains the unavailabilities. Components which are not affected by specified observations or the calculated root cause are listed as available.
The root cause fulfills the following properties: (i) It explains all unavailabilities in the evidence. This is the case due to the additional reverse implications. (ii) It is not affecting any component stated as available in the evidence. Otherwise a hard rule would be violated. (iii) It is the most probable cause for all the observations given as evidence and the risk probabilities specified as weights.

We make the assumption that all causes are unlikely (occurring less than 50% of the time). Thus, their weights are negative. As the objective of the MAP state is maximizing the sum of all weights, only the most likely cause that explains all observations is included. A less likely cause has a higher negative weight, causing the sum of the weights to be lower than optimal, and thus getting rejected.

Note that due to the soft formulas used or abduction, our approach only encourages to calculate a single root cause, but does not enforce it. It only presents multiple possible root causes, if the sum of their weights is less than the weight of a single possible cause. If there are two possible root causes with the same weight, only one is presented at random.

The user then has to investigate the presented root cause (5). If it is the source of the observed problem, the analysis is finished and the cause can be fixed. Otherwise the process starts over from (1) where the user enters additional observations. Those new observations can either be gathered while investigating the proposed root cause, or, for example, the user can verify the state of components that should also be affected by this cause.

3.4 Scenario Analysis

The following section describes the application of our approach to two different scenarios. These two scenarios illustrate failures that occurred in our IT infrastructure during the last months. Together with our system administrators, we modeled our infrastructure, analyzed these scenarios in hindsight, and tested the usefulness of our approach in retrospective. We used RockIt [8], a highly optimized and scalable MLN solver, to compute the MAP state.

The first scenario is the one depicted in Figure 1, revolving around the malfunction of our office multifunction printer. The printer offers three services: copying, printing via the network, and scanning to PDF which is then sent to an email address. A user reported the printer being broken, as scanning to PDF no longer worked. To check the proper functioning of the device, the administrator sent a print job and did a photo copy. Both tests worked successfully. Sending a test mail from his own account, the administrator also found the mail service working correctly. Further investigation finally revealed that the root cause of the scanning problem was a suspension of the account the printer used for the LDAP authentication. However, this cause was only considered after several discussions with two expert administrators involved.

We applied our approach to this scenario. The MLN was constructed automatically from the background knowledge that we maintained as a set of first-order formulas. We fed in the observations *available(PrintService)*, *available(CopyService)*, and $\neg$*available(ScanService)* and computed the most probable root cause. The MAP state that was generated as solution contained the root cause *affectedByRisk(cas.uni-ma, Systematic trying-out of passwords)*. While we could not definitely decide, in retrospective, if this risk was the underlying reason for the failure of the server *cas.uni-ma*, an authentication problem related to *cas.uni-ma* was definitely the cause for the problem.

The second scenario is an outage of our internal subversion server (SVN). It involves more components than the previous scenario and benefits from the iterative approach. The SVN is hosted on a virtual machine that is running on a blade server. Subversion was responding slowly and took long time for many operations. Neither SVN nor other processes on the virtual machine showed considerable resource utilization. Investigating resource usage on the blade server first did not reveal any abnormality. Later, a user discovered that our external website behaved similarly in performance as the SVN. This observation was first attributed to a slow internet connection in general, but we then discovered that the web server, which was hosted in a different VM but on the same blade server, produced very high network traffic, starving all other services. A member of our group had released a data set of several gigabytes in size, that was downloaded a few hundred times concurrently. That lead to congestion on the network interface of the server. Moving the download to another server resolved the problem and the behavior of the subversion server and our website went back to normal.

Analyzing this scenario with our approach, first, we only entered the observation of the unavailability of the SVN service: $\neg$*available(Service_Subversion)*. The computed MAP state proposed *affectedByRisk(VM_Subversion, Overload)* as root cause. After ruling out this cause by adding *available(VM_Subversion)* and the observation ¬*available(Service_WebHosting)*, the result of the computation was *affectedByRisk(NetworkInterface_BladeServer, Congestion)* as root cause. This risk has a high probability for that server which is running various other virtual machines, all hosting services sensitive to a high network load. The lack of other resources, e.g. CPU or RAM, is modeled as less probable, because all those services are usually not very computational complex or requiring lots of memory. For this scenario, our approach proposed reasonable root causes which we retrospectively could verify as the reason for the outage. The manual handling of the incident involved more guesswork by the system administrators and was long winded.

4 Related Work

4.1 Root Cause Analysis

In [9] an approach for requirements-driven root cause analysis for failures in software systems is proposed, wherein a Markov Logic Network is used as knowledge repository for diagnostic knowledge. The approach uses log data as observation information, the Markov Logic Network is used to deal with uncertainty stemming from incomplete log data. Their approach differs from ours in several points: they first model the background knowledge as goal trees and only convert it to first-order logic later; the evidence is solely generated from log data; and most importantly they use marginal inference, different to our approach which uses MAP inference. With marginal inference they compute probabilities for certain events occurring in all possible worlds, whereas our approach calculates the most probable combination of events. In [10] marginal inference was also used for the purpose of estimating unavailabilities in an IT infrastructure, where the authors referred to problems when marginal inference is applied to very low probabilities usually attached to the occurrence of risks in an IT setting. These problems are based on the use of sampling algorithms for performing marginal inference. Those sampling algorithms always introduce an error to the calculated numbers which are especially notable when having small probabilities. Our approach is based on solving an optimization problem, which is not affected negatively by very small probabilities.

In other works, failure diagnosis is conducted using correlation measures, e.g. using anomalies in the timing of program calls [11], or decision trees [12].

4.2 Applications of Abductive Reasoning

In [4], Singla et al. extend the approach presented in [1] and use it in the context of plan and intent recognition. Instead of adding reverse implication, they introduce a hidden cause for all implications with the same left-hand side. In general, this reduces the size of the MLN and subsequently increases performance. However, as detailed above, for our approach the mutual exclusivity clauses are not needed anyway. Nonetheless, if more probable events have to be included in the evidence, their optimization can also be included in our approach.

Most other approaches to abductive reasoning either use first-order logic to calculate a minimal set of assumptions sufficient to explain the hypothesis [13, 14, 15, 16], or Bayesian Networks to compute the posterior probability of alternative explanations given the observations [7, 17]. The former approaches are not able to estimate the likelihood of alternative explanations, as they do not support uncertainty in the background knowledge or evidence. Bayesian Networks, on the other hand, are designed to handle uncertainty. However, as they are propositional in nature, they cannot handle structured knowledge involving relations amongst multiple entities directly [1].

Bayesian Abductive Logic Programs (BALP) [18] are another approach that combines first-order logic and probabilistic graphical models. The main difference to MLNs is that BALPs are based on directed Bayesian Networks, and thus, undirected relations, like the symmetry of redundancy, are more complex.

5 Discussion and Conclusion

We presented our approach of applying abductive reasoning using Markov Logic Networks to compute the most probable root cause for a failure in an IT infrastructure. Our approach models the infrastructure with the help of first-order logic. In particular, we formulated the dependencies of the network as hard formulas. Moreover, we added weighted soft formulas to model the probability of risks that might result in the failure of components and services. Furthermore, we have argued how the expressiveness of first-order logic can be used to model general, reusable knowledge concerning risks and IT components. Our approach uses the same formalism for both knowledge presentation and abductive reasoning. Thus, all relevant information is readily available to compute the most probable root cause once an incident occurs. To the best of our knowledge, there exists no other approach that combines uncertainty and logical abductive reasoning to solve the problem of root cause analysis.

We conducted an evaluation of our approach by analyzing two failures that happened in the infrastructure of our research group. In both cases we were able to determine a root cause (respectively, a sequence of probable root causes) that turned out to be helpful for a system administrator to resolve the problem. Our approach is especially useful when the reasons for the failure are not obvious to the administrator that is in charge of resolving the problem. Thus, our approach will be more beneficial in IT infrastructures, where competences are scattered over the members of different organizational units.

We did not conduct an evaluation of the scalability of our approach. However, the MLN solver RockIt, which we used, was extensively tested in other complex, large scale settings and showed good performance for MLNs with several hundred formulas and tens of thousands of facts in the evidence [8, 19].

Further, we plan to implement a user interface that presents the computed root cause in a comprehensive way and allows to specify observations without the need for understanding the underlying formalism. Once we set up such an interface, we are able to study performance, acceptance, and benefits of the approach in a field study where the complete system is used in the daily work of a computer center.

Acknowledgement

This work has been partially supported by the German Federal Ministry of Economics and Technology (BMWI) in the framework of the Central Innovation Program SME (Zentrales Innovationsprogramm Mittelstand - ZIM) within the "project" "Risk management tool for complex IT infrastructures".

References

1. Kate, R.J., Mooney, R.J.: Probabilistic Abduction using Markov Logic Networks. IJCAI-09 Workshop on Plan, Activity, and Intent Recognition (2009)
2. Richardson, M., Domingos, P.: Markov logic networks. Machine Learning **62**(1-2) (January 2006) 107–136
3. Pople, H.E.: On the Mechanization of Abductive Logic. IJCAI (1973) 147—-152
4. Singla, P., Mooney, R.J.: Abductive Markov Logic for Plan Recognition. AAAI (2011) 1069–1075
5. Rooney, J., Heuvel, L.: Root cause analysis for beginners. Quality Progress **37**(7) (2004) 45–53
6. Bundesamt für Sicherheit in der Informationstechnik: IT-Grundschutz-Catalogues. http://goo.gl/UVuGEK (2013) Accessed 2015–02–16
7. Pearl, J.: Probabalistic Reasoning in Intelligent Systems. (1988)
8. Noessner, J., Niepert, M., Stuckenschmidt, H.: RockIt: Exploiting Parallelism and Symmetry for MAP Inference in Statistical Relational Models. CoRR (2013)
9. Zawawy, H., Kontogiannis, K., Mylopoulos, J., Mankovskii, S.: Requirements-driven root cause analysis using markov logic networks. In: Advanced Information Systems Engineering, Springer (2012) 350–365
10. von Stülpnagel, J., Ortmann, J., Schoenfisch, J.: IT Risk Management with Markov Logic Networks. Advanced Information Systems Engineering (2014) 301—-315
11. Marwede, N., Rohr, M., Hasselbring, W.: Automatic Failure Diagnosis Support in Distributed Large-Scale Software Systems based on Timing Behavior Anomaly Correlation. In: Proceeding of CSMR 2009. (2009) 47–57
12. Chen, M., Zheng, A., Lloyd, J., Jordan, M., Brewer, E.: Failure diagnosis using decision trees. In: International Conference on Autonomic Computing, 2004. Proceedings., IEEE (2004) 36–43
13. Poole, D.L., Goebel, R.G., Aleliunas, R.: Theorist: A logical reasoning system for defaults and diagnosis. In Cercone, N.J., McCalla, G., eds.: The Knowledge Frontier: Essays in the Representation of Knowledge. Springer (1987) 331–352
14. Stickel, M.E.: A Prolog-like inference system for computing minimum-cost abductive explanations in natural-language interpretation. Annals of Mathematics and Artificial Intelligence **4**(1-2) (1991) 89–105
15. Ng, H.T., Mooney, R.J.: An Efficient First-Order Abduction System Based on the ATMS. In: Proceedings of AAAI-91. (1991) 494–499

16. Kakas, A.C., Kowalski, R.A., Toni, F.: Abductive logic programming. Journal of logic and computation **1**(6) (1992) 719–770
17. Kandula, S., Katabi, D., Vasseur, J.: Shrink: A tool for failure diagnosis in IP networks. ACM SIGCOMM workshop on Mining network data (2005) 173–178
18. Raghavan, S., Mooney, R.: Bayesian Abductive Logic Programs. Statistical Relational Artificial Intelligence (Pople 1973) (2010) 82–87
19. Noessner, J.: Efficient Maximum A-Posteriori Inference in Markov Logic and Application in Description Logics. PhD thesis, University of Mannheim (2014)