Plan Assessment for Autonomous Manufacturing as Bayesian Inference *

Paul Maier, Dominik Jain, Stefan Waldherr and Martin Sachenbacher

Technische Universität München, Department of Informatics
Boltzmanstraße 3, 85748 Garching, Germany
{maierpa,jain,waldherr,sachenba}@in.tum.de

Abstract. Next-generation autonomous manufacturing plants create individualized products by automatically deriving manufacturing schedules from design specifications. However, because planning and scheduling are computationally hard, they must typically be done offline using a simplified system model, meaning that online observations and potential component faults cannot be considered. This leads to the problem of *plan assessment*: Given behavior models and current observations of the plant's (possibly faulty) behavior, what is the probability of a partially executed manufacturing plan succeeding? In this work, we propose 1) a statistical relational behavior model for a class of manufacturing scenarios and 2) a method to derive statistical bounds on plan success probabilities for each product from confidence intervals based on sampled system behaviors. Experimental results are presented for three hypothetical yet realistic manufacturing scenarios.

1 Introduction

In a scenario of mass customization using autonomous manufacturing, a factory is envisaged that generates, during the night, the manufacturing plans for numerous individualized products to be produced the next day. It employs model-based planning and scheduling capabilities, which use very abstract models to keep planning/scheduling tractable, omitting e.g. behavioral knowledge about potential failures of factory stations. In addition, observations made at execution time are not available at planning/scheduling time. In the light of such partial observations, it may become clear that certain plans will fail, e.g. if a plan operates a component that is now likely to be faulty. This leads to a problem of evaluating manufacturing plans with respect to online observations, based on models focussed on station behavior. It is especially interesting from the point of view of autonomous manufacturing control, where systems are rigid enough to allow automated advance planning/scheduling (rather than online planning), yet bear inherent uncertainties such as station failures.

We call this evaluation *plan assessment* [1]. The idea is to compute, for each product, bounds on the respective success probability. This allows to decide whether to 1) continue with a plan, 2) stop the plan because it probably will not succeed or 3) gather more information. It requires a) models of the complex, uncertain interactions among products and factory stations and b) efficient reasoning. In [1] we proposed using probabilistic automata models and a solution based on constraint optimization,

* Preprint submitted to KI 2010.

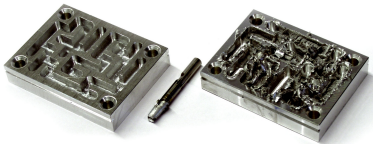

Fig. 1. Effects of cutter deterioration until breakage in machining. Image © Prof. Shea TUM PE.

which enumerates the k most probable system behaviors to estimate success probabilities. However, computing bounds is not yet possible with this approach. In this work we choose a different approach, where we a) model entire *classes of manufacturing systems* as Bayesian Logic Networks (BLNs) [2] and b) use sampling algorithms for efficient computation of *statistical* bounds on success probabilities based on *confidence intervals*. The contribution of this paper is 1) to present a BLN model for a class of manufacturing systems, 2) to propose a method to obtain said bounds from Clopper-Pearson confidence intervals [3] computed during inference and 3) to demonstrate the feasibility of this approach through experimental results.

Closest to our work are verification methods such as probabilistic model checking [4], online verification [5], or probabilistic verification [6]. However, [4] don't regard online observations, and [6] deal only with single most likely behaviors, whereas we have to consider many goal-violating (and achieving) behaviors, and all of them usually focus on models of single systems such as cars [5]. In contrast, we model a manufacturing facility and the products it processes. Other work addressed *automated manufacturability analysis*. They ask whether machining plans violate design tolerances or cost constraints [7,8], evaluating them against static constraints. In contrast, we are interested in dynamic machine behavior (nominal and off-nominal) induced by plan execution.

Assembly and Metal Machining Example Our factory test-bed – an iCim3000-based Festo Flexible Manufacturing System – consists of conveyor transports, storage, machining and assembly. It serves as the basis for hypothetical example scenarios, where a scheduler schedules the manufacturing of toy mazes (Fig. 1). A maze consists of an alloy base plate, a small metal ball and an acrylic glass cover fixed by metal pins. It is manufactured by first cutting the labyrinth groove into the maze base-plate, then drilling the fixation holes, putting the ball into the labyrinth, putting the glass cover onto the base plate and finally pushing the pins in place to fixate it.[1] While pushing, the assembly station measures the force to prevent applying too much of it. On its route through the factory the product might get flawed as a result of being worked on by faulty stations. Machining stations are suspicious candidates, because their cutter might break during operation. A broken cutter severely damages maze products (see Fig. 1). Since machining stations not only cut grooves but also drill the holes for the pins, broken cutters might also damage these holes. The damage, however, can only be detected later on: If an assembly station tries to push pins into damaged holes, too much force is applied and an alarm is triggered. The same alarm might be triggered if the assembly station's calibration is off, leading to a misalignment of the gripper holding the pin

[1] In our abstract scenarios, we disregard transportation processes.

and the base-plate's hole. This means that one cannot infer from the alarm whether the assembly station or the machining station is at fault. The question now is: Is the alarm an indicator for a broken cutter, and how does this possibility affect the different manufacturing plans?

2 Plan Assessment with Predicted System Behaviors

We address the following *plan assessment problem*: Given a model M_{assess} and observations $o_{0:t}$ obtained up to time point t, compute good lower and upper bounds p_l and p_u on the probability $Pr(\mathcal{G}_i \mid o_{0:t})$ of a manufacturing plan $\mathcal{P}_i$ succeeding: $p_l \leq Pr(\mathcal{G}_i \mid o_{0:t}) \leq p_u$.

We assume a given schedule $\mathcal{S}$ of manufacturing operations, i.e. a sequence of N tuples $\langle (p_{id}, c_{id}, t, a) \rangle_j$, where a tuple specifies that component c_{id} performs action a on product p_{id} at time t. It can be seen as a composition of the individual plans $\mathcal{P}_i$ for each product, which are sequences of actions a. M_{assess} is a probabilistic state space model derived from $\mathcal{S}$, which encodes factory station and product behavior, as well as possible observations. It defines a distribution describing the possible state evolutions for all modeled factory stations and products over time and the influence of observations on this evolution. Variables X^t encode possible states at time t, where $St^t(M_{\text{assess}})$ is the set of all possible (atomic) assignment vectors for X^t, and $St(M_{\text{assess}})$ is the set of all assignment vectors over all N time steps, i.e. the set of all possible *system trajectories*. These trajectories go beyond current time t, thereby *predicting system behavior*. $\mathcal{G}_i$ represents the event that a manufacturing plan $\mathcal{P}_i$ succeeds, i.e. generates a product according to its specification (e.g. a CAD/CAM model or a Bill of Materials (BOM)). Our basic assumption is that $\mathcal{P}_i$ succeeds *as long as no component of the factory fails*. Therefore, we model products as Boolean variables $G_i^{t_{\text{end}}}$ (with True/False for "product ok/flawed"). *Success* of $\mathcal{P}_i$ means that $G_i^{t_{\text{end}}} = \mathsf{True}$ at the future finishing time point t_{end} of the product. This simple modeling could be extended to cover multiple intermediate product states by using richer (finite) domains than $\{\mathsf{True}, \mathsf{False}\}$. We define $\mathcal{G}_i$ as the set of all *goal-achieving* trajectories $\mathcal{G}_i = \{\theta \in St(M_{\text{assess}}) \mid \theta \models G_i^{t_{\text{end}}} = \mathsf{True}\}$. We now define the success probability in terms of goal-achieving system trajectories:

Definition 1. Plan Success Probability *Given a model M_{assess}, observations $o_{0:t}$ and a manufacturing plan $\mathcal{P}_i$, we define the probability that $\mathcal{P}_i$ will succeed as*

$$Pr(\mathcal{G}_i \mid o_{0:t}) = \sum_{\theta \in \mathcal{G}_i} Pr(\theta \mid o_{0:t})$$

In most cases, it is infeasible to compute $Pr(\mathcal{G}_i \mid o_{0:t})$ exactly as it requires enumerating all trajectories to generate the complete distribution. Approximations can be computed based on a reduced set of trajectories $\Theta^* \subset St(M_{\text{assess}})$. In [1] we introduced an approach that enumerates only the k most probable trajectories. Even better is to compute hard bounds p_l and p_u defined as sum over conditional probabilities $Pr(\theta \mid o_{0:t}) = \frac{Pr(\theta, o_{0:t})}{Pr(o_{0:t})}$ of goal-achieving and goal-violating trajectories (entailing $G_i^{t_{\text{end}}} = \mathsf{False}$) $\theta \in \Theta^*$. Unfortunately, these bounds require to exactly compute $Pr(o_{0:t})$, which again requires the complete distribution. Therefore, in this work, we

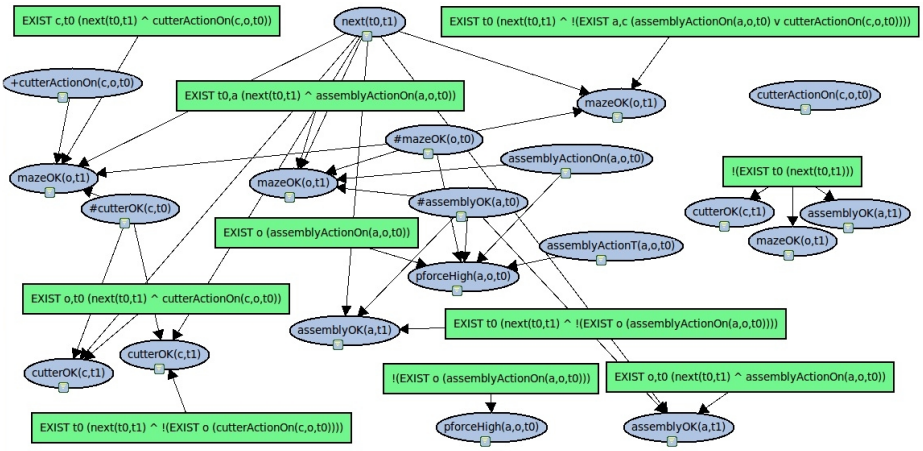

Fig. 2. Bayesian logic network that models a class of manufacturing scenarios with machining (called cutter) and assembly stations, and their interaction with maze products.

use sampling, which allows us to derive *statistical* bounds p_u^* and p_l^* from *confidence intervals* that we compute from the samples. Then we can apply a decision procedure we described in [1], with two modifications: 1) p_l^* is compared against a threshold ω_{success} and p_u^* against a threshold ω_{fail} (which we assume as given) and 2) approximation with k most probable trajectories is replaced by sampling.

3 A Bayesian Logic Network Model of Manufacturing Scenarios

We modeled the behavior of factory machining and assembly stations as well as products as a Bayesian logic network (BLN) [2] $\mathcal{B}_{\text{assess}}$ (see Fig. 2). BLNs combine first-order logic (FOL) with probabilistic modeling, allowing for compact representations of typical manufacturing interactions as well as uncertain events for classes of manufacturing systems, and they are geared towards practical application of many inference algorithms. A BLN is a *template* for the construction of a mixed deterministic/probabilistic network [9], which This means $\mathcal{B}_{\text{assess}}$ models a class of manufacturing systems, capturing general relations between stations and products, from which models M_{assess} are instantiated for concrete factories and schedules. We then either convert M_{assess} to a standard Bayesian net, so we can apply the large body of Bayesian inference techniques, or do inference in mixed networks directly [10].[2] BLNs generalize the well-known formalisms of hidden Markov models (HMM) and dynamic Bayesian networks (DBN).

Key elements of a BLN $\mathcal{B}_{\text{assess}}$ are abstract random variables (ARVs), entity types, fragments and logical formulas. ARVs correspond to logical predicates evaluating to true or false and model states of stations and products as well as relations such as stations working specific products. Placeholders are used within ARVs to refer to abstract typed entities. Fragments are associated with conditional probability tables. Distribu-

[2] We refer to both Bayesian and mixed nets with M_{assess}

tions over instantiations of ARVs are defined through multiple fragments (ellipses in Fig. 2) with mutually exclusive first-order logic preconditions (boxes in Fig. 2).

Our model $\mathcal{B}_{\text{assess}}$ realizes state evolution over time with two abstract entities $t0$ and $t1$, representing successive time points, and ARVs relating to them for successive actions, states, etc. A time line is enforced through ARV $next(t0, t1)$, which encodes that $t0$ precedes $t1$. When instanciating, successiveness of time points $T0, T1, \ldots$ is ensured by clamping $next(T0, T1)$, $next(T1, T2)$, and so on to True. Uncertain station evolution is modeled using two ARVs: $\mathsf{assemblyOK}(a, t0)$, $\mathsf{assemblyOK}(a, t1)$ and $\mathsf{cutterOk}(c, t0)$, $\mathsf{cutterOk}(c, t1)$ for assembly and machining stations. The failure probabilities 0.03 and 0.01 for machining and assembly stations are encoded in the fragments for these ARVs. Product state evolution is modeled in a similar way, i.e. we have the ARVs $\mathsf{mazeOK}(o, t0)$, $\mathsf{mazeOK}(o, t1)$. Their fragments are different in that they currently don't encode any uncertainty. The force alarm observations are encoded as evidence ARVs: $\mathsf{pforceHigh}(a, t)$ encodes that the force measured at the assembly station a at time t was too high (if True). To encode actions, $\mathcal{B}_{\text{assess}}$ consists of relations for assembly and machining stations working mazes at a certain time, i.e. ARVs $\mathsf{assemblyActionOn}(a, o, t)$ and $\mathsf{cutterActionOn}(c, o, t)$. The complex relation that a force alarm can be triggered by cutter-damaged holes as well as a miscalibrated assembly (section 1) can be expressed as a FOL formula: $\mathsf{assemblyActionOn}(a, o, t) \Rightarrow (\mathsf{pforceHigh}(a, t) \Leftrightarrow (\neg\mathsf{assemblyOK}(a, t) \vee \neg\mathsf{mazeOK}(o, t)))$[3].

To perform plan assessment for a specific scenario, $\mathcal{B}_{\text{assess}}$ is instantiated to a concrete model M_{assess}. ARVs are thereby compiled to a set of random variables (e.g. $\mathsf{mazeOK}(\mathsf{Maze0}, \mathsf{T0})$), fragments to conditional probability distributions over them, and logical formulas to propositional logical constraints. In particular, instances of ARVs representing product states when the product should be finished encode product success, i.e. the goal variables $G_i^{t_{\text{end}}}$. In our scenarios we use goals $\mathsf{mazeOK}(\mathsf{Maze0}, \mathsf{T5})$, $\mathsf{mazeOK}(\mathsf{Maze1}, \mathsf{T9})$, $\mathsf{mazeOK}(\mathsf{Maze2}, \mathsf{T7})$ and $\mathsf{mazeOK}(\mathsf{Maze3}, \mathsf{T12})$. A given schedule $\mathcal{S}$ determines the set E' of concrete instances of machining and assembly stations as well as maze products (e.g. Mach0, Maze0, Assy0). *Evidence variables* (instantiated from evidence ARVs) are clamped to values representing the actual observations, e.g. $\mathsf{pforceHigh}(\mathsf{Assy0}, \mathsf{T4}) = \mathsf{True}$.

4 Computing Confidence Intervals for Plan Success Probabilities

Now we can use M_{assess} to assess manufacturing plans for individual products. Since we usually cannot have an exact $Pr(\mathcal{G}_i \mid o_{0:t})$, nor hard bounds (Section 2), we propose to compute the confidence interval of $Pr(\mathcal{G}_i \mid o_{0:t})$: "soft" bounds p_u^* and p_l^* according to a predefined probability γ, the *coverage probability*, that $Pr(\mathcal{G}_i \mid o_{0:t})$ will be within these bounds.

Theorem 1. *The bounds p_u^* and p_l^* on $Pr(\mathcal{G}_i \mid o_{0:t})$ are given by the Clopper-Pearson interval [3].*

Proof. Let G be a Bernoulli-distributed Boolean random variable (BBRV) with parameter p, which is being sampled in a Bernoulli-process, counting appearances of

[3] For technical reasons we modeled it using deterministic fragments (i.e. with prob. values 1.0 and 0.0), the FOL formula however is the more elegant equivalent.

coverage rate γ		Number of samples			Exact
		100	2500	10000	
0.95	mazeOK(M2,T7)	$[0.002, 0.054]$	$[0.035, 0.051]$	$[0.035, 0.042]$	0.0387
	mazeOK(M1,T9)	$[0.593, 0.772]$	$[0.591, 0.629]$	$[0.587, 0.606]$	0.6041
	mazeOK(M0,T5)	$[0.000, 0.029]$	$[0.000, 0.001]$	$[0.000, 0.000]$	0.0000
0.999	mazeOK(M2,T7)	$[0.010, 0.162]$	$[0.030, 0.057]$	$[0.035, 0.048]$	0.0387
	mazeOK(M1,T9)	$[0.359, 0.677]$	$[0.573, 0.637]$	$[0.594, 0.626]$	0.6041
	mazeOK(M0,T5)	$[0.000, 0.066]$	$[0.000, 0.003]$	$[0.000, 0.001]$	0.0000
	runtime	0.06	1.61	6.24	58.76
		0.06	1.52	6.00	58.76

Table 1. Confidence intervals on success probabilities for the mazes in scenario 1 obtained with likelihood weighting and exact results obtained through variable elimination.

$G = \text{True}$ and $G = \text{False}$ as a and b, respectively. Let γ be the coverage probability. Then the Clopper-Pearson interval defines bounds $p_l^* = F_{a,b,\gamma}^{-1}(1 - \frac{\alpha}{2})$ and $p_u^* = F_{a,b,\gamma}^{-1}(\frac{\alpha}{2})$, where $\alpha = 1 - \gamma$ and $F_{a,b,\gamma}^{-1} = I_{a+1,b+1}^{-1}$, I being the regularized incomplete beta function; I^{-1} is thus the inverse of the cumulative distribution function (CDF) of the beta distribution. Any manufacturing goal $G_i^{t_{\text{end}}}$ is a BBRV with parameter $p = Pr(\mathcal{G}_i \mid o_{0:t})$. We sample trajectories that correspond to the observations and entail assignments to $G_i^{t_{\text{end}}}$. Therefore, the trajectory sampling can be seen as a sampling of $G_i^{t_{\text{end}}}$. The sampling yields n samples, $n_{G_i^{t_{\text{end}}}}$ goal-achieving and $n - n_{G_i^{t_{\text{end}}}}$ goal-violating. If we now set $G = G_i^{t_{\text{end}}}$, $a = n_{G_i^{t_{\text{end}}}}$, $b = n - n_{G_i^{t_{\text{end}}}}$, the theorem follows.

We quickly recap why this works. We abbreviate $G_i^{t_{\text{end}}}$ to G. The quantities n_G, $n - n_G$ determine a *distribution over* p, which (assuming a uniform distribution over parameters when there are no samples) is given by the beta distribution [11]. The CDF $F_{n_G,n-n_G}(x) = Pr(p \leq x)$ allows to compute the probability that p is at most x. Observe now that the *complement* of the given $\gamma = Pr(p_l^* \leq p \leq p_u^*)$, i.e. the probability that p is *outside* the bounds of the interval, can be written as $Pr(p < p_l^*) + Pr(p > p_u^*) = 1 - \gamma = \alpha$. We can rewrite this equation as $Pr(p \leq p_l^*) + 1 - Pr(p \leq p_u^*) = \alpha$, where all probabilities are represented through the CDF $F_{n_G,n-n_G}(x)$. Now we can use the inverse CDF $F_{n_G,n-n_G}^{-1}(y)$ to compute the bounds p_l^* and p_u^*. Except in extreme cases, we can assume that α is to equal parts composed of $Pr(p \leq p_l^*)$ and 1 - $Pr(p \leq p_u^*)$, i.e. $Pr(p \leq p_l^*) = \frac{\alpha}{2} = 1 - Pr(p \leq p_u^*)$. Resolving for p_l^* yields $p_l^* = F_{n_G,n-n_G}^{-1}(\frac{\alpha}{2})$ and for p_u^* gives $p_u^* = F_{n_G,n-n_G}^{-1}(1 - \frac{\alpha}{2})$.

Of course we would like to have as narrow intervals as possible. Increasing the number of samples gives us narrower intervals. Thus, a practical stop criterion for sampling algorithms is to predefine the size of the interval to be sufficiently small, and then sample until the interval is narrower than this predefined size. Note that an estimate for the success probability itself can be computed in the usual way by dividing the number of goal-achieving samples by the number of all samples, $Pr(\mathcal{G}_i \mid o_{0:t}) \approx \frac{n_G}{n}$.

5 Experimental Results

We inferred the success probability of mazes for three different (hypothetical) scenarios, with corresponding ground models instantiated from $\mathcal{B}_{\text{assess}}$, using three sampling algorithms [12,13,10]. We ran Java implementations of the algorithms on an Intel Core2 Duo with 2.53 Ghz and 4GB of RAM. In all scenarios there were two machining stations (Mach0/1) and one assembly station (Assy0). In the smaller *scenario 1* (310 nodes) Mach0 has become faulty. Its schedule ranges over nine time points for three mazes (Maze0/1/2). Observations have been made up to T4, and a force alarm was triggered at T4 while the assembly station was pushing pins into Maze0. In *scenario 2* (520 nodes) Assy0 is faulty. Here, four mazes (Maze0/1/2/3) are scheduled, covering 12 time points. Observations are available up to T8. The pin assembly is done at T4, T6 and T8 for Maze0, Maze2 and Maze1 respectively. A force alarm is observed at all three time points. *Scenario 3* (520 nodes) is similar to the former, with the difference that again Mach0 is faulty. Consequently, at T8 no force alarm is triggered. In all scenarios Maze0 and Maze2 are cut on Mach0, while Maze1 and (in scenarios 2 and 3) Maze3 are cut on Mach1. Further, Maze0/1/2/3 should be finished by T5/9/7/12, respectively. Note that results for different products in the same scenario are obtained simultaneously.

In all scenarios we can infer meaningful bounds on the products' success probabilities and thereby identify jeopardized products. In scenario 2 (Maze0:$[0.000, 0.003]$, Maze1: $[0.000, 0.003]$, Maze2: $[0.000, 0.003]$, Maze3: $[0.002, 0.011]$), the observations strongly indicate a faulty assembly (which is a lot more likely than both Mach0 and Mach1 failing simultaneously), which means that the unfinished Maze3 will certainly fail, too. In scenario 3, in contrast, (Maze0: $[0.000, 0.000]$, Maze1: $[1.000, 1.000]$, Maze2: $[0.000, 0.000]$, Maze3: $[0.908, 0.918]$) Maze3 is very likely to succeed since, given the observations, it is highly likely that Mach0 is faulty. Scenario 1 is less conclusive (see table 1): While Maze2 is clearly certain to fail, there's an uncertain chance that Maze1 will be ok. So all in all the result could lead to these decisions: In scenario 2, stop all manufacturing and in scenario 3, continue to finish Maze3. In uncertain cases such as scenario 1 methods to actively gather information could be triggered [14].

Table 1 illustrates how choosing stricter coverage probabilities γ widens the interval. It also confirms that increasing the number of samples results in better intervals. Good intervals ($\gamma = 0.95$, width less than 0.01) can already be retrieved in under a minute, sometimes even in under a second (scenario 2, SampleSearch), with less than 1000 samples. Choosing a good algorithm seems to depend on the scenarios (see Table 2): for 1 and 2 SampleSearch is best, while for 3, likelihood weighting is the better choice. Notably, the runtime does not strictly depend on the problem size, i.e. no single algorithm can be trusted to be equally quick for all problems. A solution could be to run algorithms simultaneously (taking advantage of current multi-core technology) up to a time limit and then take the result with the narrowest interval, or stop when a predefined interval width is reached.

Note that simplified inference methods such as forward filtering (which one might use for HMMs) are not applicable to the type of problem we considered, because the interactions that we modelled result in several coupled temporal chains, which, in particular, do not have the Markov property. Inference is, therefore, considerably more dif-

	Algorithm								
$\max_I \lvert I \rvert$	likelihood weighting			backward simulation			SampleSearch		
≤ 0.025	5900	320	2020	5820	200	-[4]	6180	200	1380
	3.61	0.90	3.82	1.23	5.44	-	0.84	0.06	13.27
≤ 0.01	36800	1000	11800	37280	300	-	38440	660	5080
	22.42	2.67	21.47	7.00	7.56	-	4.76	0.16	42.08
≤ 0.0025	588020	17420	185820	588860	1200	-	614700	6460	181320
	384.38	50.06	362.08	111.95	31.75	-	79.70	1.40	1551.25

Table 2. Comparing algorithms on scenarios 1 / 2 / 3 by average number of samples (above) needed to reach a target confidence interval width, and runtime in seconds (below).

ficult and presents a challenge as problem sizes increase. Advances in lifted inference [15,16] might soon alleviate this problem.

6 Conclusion and Future Work

We presented a model-based method that samples behavior-trajectories of stations and products in a manufacturing scenario to compute *confidence intervals* for success probabilities of the products, thereby allowing an autonomous manufacturing system to react to failures and other unforeseen events. The method uses a Bayesian logic network (BLN) model of a class of manufacturing systems. Results show that, for multiple scenarios generated from the same abstract BLN, we can indeed identify jeopardized products based on the computed bounds. Future work will concern a direct comparison with our previous work [1] and more experiments in order to assess the method's scalability limits. We are also interested in exploiting the obtained results in order to update the underlying model, for instance, to automatically adapt to parameter drifts of components, or to learn, e.g., failure probabilities in the first place.

References

1. Maier, P., Sachenbacher, M., Rühr, T., Kuhn, L.: Constraint-Based Integration of Plan Tracking and Prognosis for Autonomous Production. In: Proc. KI-2009. LNCS, Paderborn, Germany, Springer (2009)
2. Jain, D., Waldherr, S., Beetz, M.: Bayesian Logic Networks. Technical report, Technische Universität München (2009)
3. Clopper, C., Pearson, E.: The use of confidence or fiducial limits illustrated in the case of the binomial. Biometrika **26** (1934) 404
4. Rutten, J., Kwiatkowska, M., Norman, G., Parker, D.: Mathematical Techniques for Analyzing Concurrent and Probabilistic Systems. Volume 23 of CRM Monograph Series. American Mathematical Society (2004)
5. Althoff, M., Stursberg, O., Buss, M.: Online Verification of Cognitive Car Decisions. In: Proc. IV-2007. (2007) 728–733
6. Mahtab, T., Sullivan, G., Williams, B.C.: Automated Verification of Model-Based Programs Under Uncertainty. In: Proc. ISDA-2004. (2004)
7. Nau, D.S., Gupta, S.K., Regli, W.C.: Manufacturing-Operation Planning versus AI Planning. In: Integrated Planning Applications: Papers from the 1995 AAAI Spring Symposium, AAAI Press (1995) 92–101
8. Kiritsis, D., Neuendorf, K.P., Xirouchakis, P.: Petri Net Techniques for Process Planning Cost Estimation. Advances in Engineering Software **30** (1999) 375–387
9. Mateescu, R., Dechter, R.: Mixed Deterministic and Probabilistic Networks. Annals of Mathematics and Artificial Intelligence **54** (2008) 3–51
10. Gogate, V., Dechter, R.: SampleSearch: A Scheme that Searches for Consistent Samples. In: Proc. AISTATS-2007. (2007)
11. Bishop, C., et al.: 2. Probability Distributions. In: Pattern Recognition and Machine Learning. Springer (2006) 67–74

[4] Backward simulation did not produce any results for scenario 3, because the problem was too ill-conditioned, such that no countable samples could be generated. SampleSearch does not have this problem and will always generate usable samples (given enough time).

12. Fung, R.M., Chang, K.C.: Weighting and integrating evidence for stochastic simulation in bayesian networks. In: Proc. UAI-1989, North-Holland Publishing (1989) 209–220
13. Fung, R., Del Favero, B.: Backward Simulation in Bayesian Networks. In: Proc. UAI-1994, Morgan Kaufmann (1994) 227
14. Kuhn, L., Price, B., de Kleer, J., Do, M.B., Zhou, R.: Pervasive Diagnosis: The Integration of Diagnostic Goals into Production Plans. In Fox, D., Gomes, C.P., eds.: Proc. AAAI-2008, AAAI Press (2008) 1306–1312
15. Rodrigo de Salvo Braz and Eyal Amir and Dan Roth: Lifted First-Order Probabilistic Inference. In: IJCAI. (2005) 1319–1325
16. Parag Singla and Pedro Domingos: Lifted First-Order Belief Propagation. In: Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence. (2008)