Integrating Model-based Diagnosis and Prognosis in Autonomous Production

Paul Maier [1], Martin Sachenbacher [1], Thomas Rühr [1], Lukas Kuhn [2]

[1] *Technische Universität München, Boltzmannstraße 3, 85748 Garching, Germany*
{maierpa, sachenba, ruehr}@in.tum.de
[2] *PARC, Palo Alto, CA, USA*
Lukas.Kuhn@parc.com

ABSTRACT

Today's complex production systems allow to simultaneously build different products following individual production plans. Such plans may fail due to component faults or unforeseen behavior, resulting in flawed products. In this paper, we propose a method to integrate diagnosis with plan assessment to prevent plan failure, and to gain diagnostic information when needed. In our setting, plans are generated from a planner before being executed on the system. If the underlying system drifts due to component faults or unforeseen behavior, plans that are ready for execution or already being executed are uncertain to succeed or fail. Therefore, our approach tracks plan execution using probabilistic hierarchical constraint automata (PHCA) models of the system. This allows to explain past system behavior, such as observed discrepancies, while at the same time it can be used to predict a plan's remaining chance of success or failure. We propose a formulation of this combined diagnosis/assessment problem as a constraint optimization problem, and present a fast solution algorithm that estimates success or failure probabilities by considering only a limited number k of system trajectories.

1 INTRODUCTION

As the market demands for customized and variant-rich products, the industry struggles to implement production systems that demonstrate the necessary flexibility while maintaining cost efficiency comparable to highly automated mass production. A main cost driver in automated production is the human workforce needed for setup steps, the development of processes, and quality assurance. These high labor costs can typically only be amortized by very large lot sizes. For small lot sizes as found in prototype and highly customized production, human workers are still unchallenged in flexibility and cost by automated systems. Therefore, to facilitate the emergence of mass customization, levels of flexibility similar to the flexibility of human workers must be reached at prices only highly automated systems can achieve.

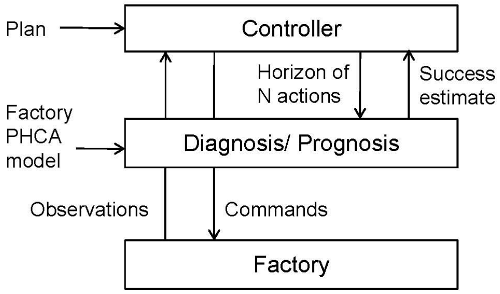

Figure 1: Model-based plan assessment.

The German research cluster ”Cognition for Technical Systems” (CoTeSys) (Beetz *et al.*, 2007) was founded to understand human cognition and make its performance accessible for technical systems. Future technical systems are expected to act robustly under high uncertainty, reliably handle unexpected events, quickly adapt to changing tasks and own capabilities. A key technology for the realization of such systems is automated planning combined with self-diagnosis and self-assessment. These capabilities can allow the system to plan its own actions, and also react to failures and adapt the behavior to changing circumstances.

From the point of view of planning, production systems are a relatively rigid environment, where the necessary steps to manufacture a product can be anticipated well ahead. However, from a diagnosis point of view, production systems are typically equipped with only few sensors, so it cannot be reliably observed whether an individual manufacturing step went indeed as planned; instead, this becomes only gradually more certain during execution of the production plan. Therefore, in the presence of faults or other unforeseen events – which become more likely in individualized production – the question arises whether plans that are ready for execution or already being executed will indeed succeed, and whether it is necessary to revise a plan or even switch to another plan.

To address this problem, we propose in this paper a model-based capability that estimates the suc-

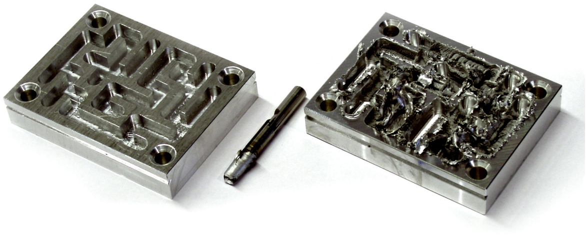

Figure 2: Effects of cutter deterioration until breakage in machining. Image © Prof. Shea TUM PE

cess probability of production plans in execution (figure 1). We assume that a planner provides plans given a system model. A plan is a sequence of actions where each action is executed at its corresponding start time. Whenever the system produces an observation, it is forwarded to a module that performs simultaneous plan tracking and plan prognostic using probabilistic hierarchical constraint automata (PHCA) models (Williams *et al.*, 2001) of the system. We propose a formulation of this problem as a soft constraint optimization problem (Schiex *et al.*, 1995) over a window of N time steps that extends both into the past and the future, and present a fast but approximate solution method that enumerates only k most likely system trajectories. The resulting success or failure prognosis can then be used to autonomously react in different ways depending on the probability estimate; for instance, continue with plan execution, discard the plan, or augment the plan by adding observation-gathering actions to gain further information (Kuhn *et al.*, 2008).

In the remainder of the paper, we first motivate the approach informally with an example from an automated metal machining process, and then present our algorithmic solution and experimental results.

2 METAL MACHINING AND ASSEMBLY EXAMPLE

As part of the CoTeSys cognitive factory test-bed, we set up a customized and extended Flexible Manufacturing System (FMS) based on the iCim3000 from Festo AG (see figure 5). The system consists of a conveyor transport and three stations: storage, machining (milling and turning), and assembly. We built a simplified model of this manufacturing system (see figure 4) which consists only of the machining and the assembly station and allows to track system behavior over time, including unlikely component faults. In particular, the machining station can transition to a "cutter blunt" composite location, where vibrations are caused during operation due to a blunt cutter. A blunt cutter is very likely to break, leading to flawed products (see figure 2). The assembly station model contains a composite location which models occasional vibrations. A sensor at the assembly station can detect these vibrations, yielding binary signals "vibration occurred" and "no vibration occurred". However, the signal is ambiguous, since the sensor cannot differentiate between the two possible causes.

Two products are produced using a single production plan $\mathcal{P}_{prod}$: a toy maze consisting of an alloy base plate and an acrylic glass cover, and an alloy part of a robotic arm (see figure 3). $\mathcal{P}_{prod}$ consists of these steps: (1) cut maze into base plate (one time step), (2) assemble base plate and cover (one time step), (3,4,5,6) cut robot arm part (one to four time steps). The plan takes two to six time steps (starting at $t = 0$). The plan is considered successful if both products are flawless. In our example, only a broken cutter causes the machined product to be flawed, in all other cases the production plan will succeed. Now consider the following scenario: after the second plan step (assembling the maze base plate and its cover at $t = 2$) a vibration is observed. Due to sensor ambiguity it remains unclear whether the plan is unaffected (vibration within assembly) or whether it might fail in the future due to a broken cutter (vibration caused by a blunt cutter), and the question for the planner is: How likely is it that the current plan will still succeed? Our new capability allows to compute this likelihood, taking into account past observations and future plan steps.

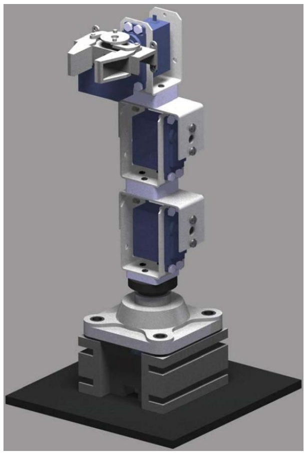

Figure 3: The robotic arm product. Image © Prof. Shea TUM PE

3 MODELING SYSTEM BEHAVIOR WITH PHCA

Probabilistic hierarchical constraint automata (PHCA) were introduced in (Williams *et al.*, 2001) as a compact encoding of hidden markov models (HMMs). These automata have the required expressivity to uniformly model both probabilistic hardware behavior (e.g., likelihood of component failures) and complex software behavior (such as high level control programs).

Definition 1 (PHCA)
A PHCA is a tuple $\langle \Sigma, P_{\Xi}, \Pi, O, Cmd, \mathcal{C}, P_T \rangle$, where:

- Σ is a set of locations, partitioned into primitive locations Σ_p and composite locations Σ_c. Each composite location denotes a hierarchical, constraint automaton. A location may be marked or unmarked. A marked location represents an active execution branch.
- $P_{\Xi}(\Xi_i)$ denotes the probability that $\Xi_i \subseteq \Sigma$ is the set of start locations (initial state). Each compos-

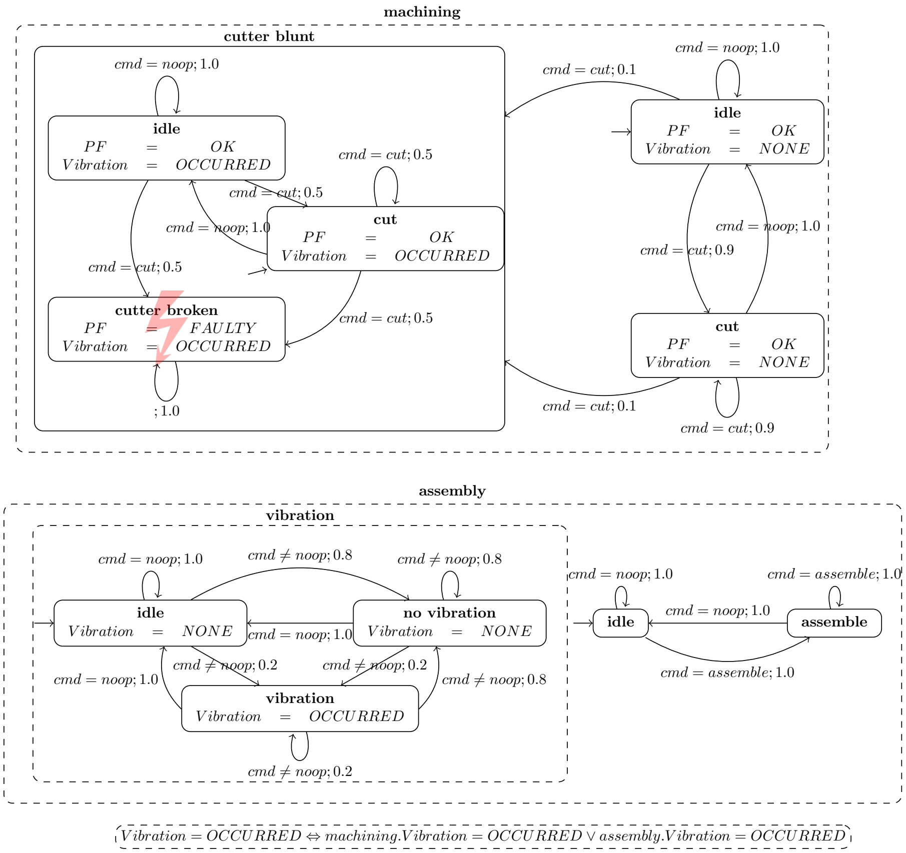

Figure 4: Simplified PHCA of the manufacturing system. The machining and assembly station are modeled as parallel running composite locations (indicated by dashed borders). Variables appearing within a location are local to this location, i.e. $machining.cmd$ refers globally to the command variable cmd within composite location $machining$. Note: "noop" means "no operation".

ite location $l_i \in \Sigma_c$ may have a set of start locations that are marked when l_i is marked.

- Π is a set of variables with finite domains. $\mathcal{C}[\Pi]$ is the set of all finite domain constraints over Π.
- $O \subseteq \Pi$ is the set of observable variables.
- $Cmd \subseteq \Pi$ is the set of command variables.
- $\mathcal{C} : \Sigma \rightarrow \mathcal{C}[\Pi]$ associates with each location $l_i \in \Sigma$ a finite domain constraint $\mathcal{C}(l_i)$.
- $P_T(l_i)$, for each $l_i \in \Sigma_p$, is a probability distribution over a set of transition functions $T(l_i) : \Sigma_p^{(t)} \times \mathcal{C}[\Pi]^{(t)} \rightarrow 2^{\Sigma^{(t+1)}}$. Each transition function maps a marked location into a set of locations to be marked at the next time step, provided that the transition's guard constraint is entailed. We denote the set of all transitions as $\mathcal{T}$, and the guard of a transition $\tau \in \mathcal{T}$ as $\mathcal{G}(\tau)$, where function $\mathcal{G} : \mathcal{T} \rightarrow \mathcal{C}[\Pi]$ maps transitions to their guards.

Definition 2 (PHCA State) The state of a PHCA at time t is a set of marked locations called a marking $m^{(t)} \subset \Sigma$.

The example PHCA shown in figure 4 illustrates the PHCA definition. The main factory components $machining$ and $assembly$ are encoded as top level composite locations. A dashed border indicates that locations may be marked at the same time, which means they can run in parallel. There is a third top level location at the bottom of figure 4 whose behavior constraint encodes that an observed vibration is caused by one of the two components or both. Primitive locations are for example $machining.idle$ and $machining.cut$, which encode the machining station being in an idle state and working on a piece. An example for an observable variable is $Vibration$, which encodes whether a vibration has occurred or not. The dependent variables $machining.Vibration$ and $assembly.Vibration$ encode for each component whether it caused a vibration. A command variable is, e.g., $machining.cmd$. It occurs in the guard constraint for transition $idle \rightarrow cut$ within composite location $machining$: $machining.cmd = cut$. Transition guards have the general form <guard constraint>;<transition probability>. The guard constraint is a logical constraint over PHCA variables, usually an assignment to command variables. The transition is non-deterministic: Given the guard is satisfied, it is taken with probability 0.9. The remaining possibility (completing the conditional probability

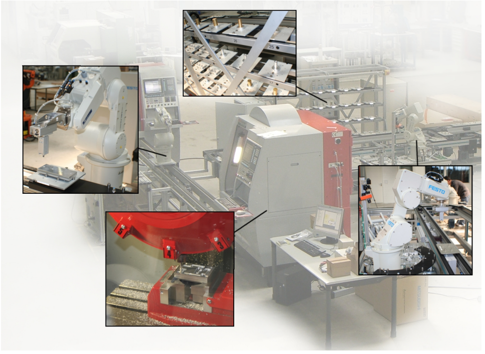

Figure 5: The hardware setup for experimentation, showing storage, transport, robot and machining components.

distribution) is the transition from $idle$ to the composite location $cutter\,blunt$, which has the same guard and is taken with probability 0.1.

4 PLAN ASSESSMENT AS CONSTRAINT OPTIMIZATION OVER PHCA MODELS

Plan assessment requires tracking of the system's plan-induced evolution; in our case, it means tracking the evolution of PHCA markings. Previous work (Mikaelian *et al.*, 2005) introduced an encoding of PHCA as soft constraints (Schiex *et al.*, 1995), and casted the problem of tracking PHCA markings as a soft constraint optimization problem whose solutions are the most probable trajectories (sequences of markings) within a window of an N time steps. In the following, we recap this encoding and show how the problem of tracking plans is formulated as constraint optimization problem based on an encoding of a PHCA model, available observations, and the production plan as soft constraints.

Encoding PHCA Models as Soft Constraints

The PHCA model is encoded as variables and constraints of a probabilistic variant of a constraint optimization problem (COP), which is defined as follows:

Definition 3 (Constraint Optimization Problem) A *Probabilistic Constraint Optimization Problem* (COP) $\mathcal{R}$ is a triple (X, D, C) where $X = \{X_1, ..., X_n\}$ is a set of variables with corresponding set of finite domains $D = \{D_1, \ldots, D_n\}$, and $C = \{C_1, \ldots, C_r\}$ is a set of constraints (S_i, F_i) with scope $S_i = \{X_{i1}, \ldots, X_{im}\} \subseteq X$ and a constraint function $F_i : D_{i1} \times \ldots \times D_{im} \rightarrow [0, 1]$. The constraint function maps partial assignments of variables in S_i to a probability value in $[0, 1]$. Given variables of interest (solution variables) $Y \subseteq X$, a solution to the COP is an assignment to Y that has an extension to all variables X that maximizes the global probability value in terms of the functions F_i.

The PHCA model encoding as a probabilistic COP consists of:

- Set of variables $X_\Sigma^{(t)} \cup \Pi^{(t)} \cup X_{Exec}^{(t)}$ for $t = 0..N$, where $X_\Sigma^{(t)} = \{L_1^{(t)}, ..., L_{|\Sigma|}^{(t)}\}$ is a set of variables that correspond to PHCA locations $l_i \in \Sigma$, $\Pi^{(t)}$ is the set of PHCA variables at time t, and $X_{Exec}^{(t)} = \{E_1^{(t)}, ..., E_n^{(t)}\}$ is a set of auxiliary variables used for encoding the execution semantics of the PHCA within an N-step time window.
- Set of finite, discrete-valued domains $D_{X_\Sigma} \cup D_\Pi \cup D_{X_{Exec}}$, where $D_{X_\Sigma} = \{\{Marked, Unmarked\}\}$ contains the single domain for variables in X_Σ, D_Π is the set of domains for PHCA variables Π, and D_{Exec} is a set of domains for variables X_{Exec}.
- Set of logical (hard) constraints $R \subseteq C$ that include the behavioral constraints associated with locations within the PHCA, as well as the encoding of the PHCA execution semantics.
- Set of soft-constraints which encode all probabilistic features, such as the probability distribution P_Ξ of PHCA start states and probabilities associated with PHCA transitions P_T.

Hard constraints such as behavioral PHCA constraints are represented by a soft constraint function F mapping (partial) variable assignments disallowed by the constraint to 0.0 and allowed assignments, or models, to 1.0. The optimal solutions to the COP are assignments to solution variables $X_\Sigma^{(t)}$ for $\{t, \ldots, t+N\}$, representing the most probable PHCA state trajectories. To avoid confusion, we refer to the behavioral and guard constraints of a PHCA as PHCA constraints, and COP (soft and hard) constraints simply as constraints.

Executing a PHCA, given a marking $m^{(t)}$, means to identify possible target locations to be marked at $t+1$, probabilistically choose transitions and check consistency of observations and commands with transition guards as well as behavior of the targets. Also, it involves checking for interdependences encoded in behavior PHCA constraints, e.g., that a vibration occurs if and only if a vibration occurs in machining or in assembly. Finally, targets have to be marked correctly regarding, among other things, the hierarchical structure of a PHCA and initial marking.

These execution semantics are encoded as COP constraints for single time points, consisting of consistency and marking constraints, and for transitions between time points. The COP consists of N copies of these constraints, corresponding to the N time steps of the time window. Variables belonging to time step t are marked by superscript $^{(t)}$. Marking constraints are less important here, therefore we focus on consistency and transition constraints.

PHCA constraints are local to locations (behavior) or transitions (guards), i.e., if inconsistent, they render a specific location or transition impossible. In contrast, COP constraints always globally refer to the complete model. If inconsistent, no COP solution and therefore no PHCA trajectory exists. This means PHCA constraints cannot be mapped directly to COP constraints. The solution are consistency constraints: they explicitly encode consistency of behavior and guards by connecting the PHCA constraints with auxiliary variables $Behavior_L^{(t)}, Guard_\tau^{(t)} \in X_{Exec}$ for locations L and transitions τ at time t:

Behavioral Consistency: ($\forall t \in \{0..N\}, \forall L \in \Sigma : Behavior_L^{(t)} = Consistent \Leftrightarrow \mathcal{C}(L)^{(t)}$)
Transition Guard Consistency: ($\forall\, t \in \{0..N-1\}, \forall \tau \in \mathcal{T} : Guard_\tau^{(t)} = Consistent \Leftrightarrow \mathcal{G}(\tau)^{(t)}$)

Transition choice constraints encode, for a given location, that a single outgoing transition may be probabilistically enabled at time t. All transitions are assigned auxiliary variables $\{T^{(t)} | t \in \{0..N\}\}$ with domain $\{Enabled, Disabled\}$, encoding whether a transition T is possible in between t and $t+1$, regardless of guard satisfaction.

Probabilistic Transition Choice:[1] ($\forall t \in \{0..N-1\}, \forall P \in \Sigma_p : (\exists \tau \in \{T | Source(T) = P\} \Rightarrow [P^{(t)} = Marked \Leftrightarrow (\exists T \in \{T | Source(T) = P\} : T^{(t)} = Enabled \wedge (\forall T' \in (\{T | Source(T) = P\} - \{T\}) : T'^{(t)} = Disabled))] \bigwedge [P^{(t)} = Unmarked \Leftrightarrow (\forall T \in \{T | Source(T) = P\} : T^{(t)} = Disabled)]))$

The probability distribution over all possible transitions is represented by the following soft constraint function F_T with scope $S_T = \{P^{(t)}\} \cup \{T_i^{(t)} | Source(T_i) = P\}$, mapping each model M of the transition choice constraint to probability values:

$$F_T(M) = \begin{cases} Prob(T_i) & if(\exists T_i^{(t)} : T_i^{(t)} = Enabled) \\ 1.0 & otherwise \end{cases}$$

If a transition is enabled with some probability > 0, it's guard must be satisfied. This is encoded through transition consistency constraints, which specify allowed assignments to variables $T^{(t)}$ and $Guard_\tau^{(t)}$.

For a more in depth discussion of the COP encoding of PHCAs we refer to (Mikaelian *et al.*, 2005).

Encoding Plans as Constraints

We consider a plan $\mathcal{P}$ and its goal G. A plan is a sequence of action and start time pairs $\mathcal{P} = ((a, 0), (a, 1), \ldots, (a, n))$. The starting times here are simply represented by indices of time steps. An action is an assignment to command variables $Cmd^{(t)} \subseteq \Pi^{(t)}$ for the corresponding start time t, referred to by $a^{(t)}$. For example $a_{cut}^{(t)}$ and $a_{assemble}^{(t)}$ are assignments $machining.cmd^{(t)} = cut \wedge assembly.cmd^{(t)} = noop$ and $assembly.cmd^{(t)} = assemble \wedge machining.cmd^{(t)} = noop$. $\mathcal{P}$ is then mapped to the following logical constraint: $\forall t \in \{0..N\} : a^{(t)}$.

The plan's goal G is to produce a flawless product. We encode this informal description as a logical constraint $G \equiv \forall PF^{(t_{end})} \in RelevantFeatures(\mathcal{P}) : PF^{(t_{end})} = OK$ over product feature variables $PF^{(t)} \in \{OK, FAULTY\}$ at the end of the execution, t_{end}. $RelevantFeatures()$ is a function mapping a production plan to all product feature variables which define the product. Each system component is responsible for a product feature in the sense that if it fails, the product feature is not present ($PF^{(t)} = FAULTY$). In our example, there is only a single product feature PF, which is absent if the cutter is broken. The goal constraint for the above mentioned plan (three time steps long) is accordingly $PF^{(3)} = OK$.

Encoding Observations as Constraints

Observations made during the plan execution (such as the occurred vibration at $t = 2$) are added as soft-constraints over observable variables in the PHCA. These constraints are very similar to soft-constraints over command variables resulting from production plans. An observation at time t is encoded as an assignment to a corresponding observable variable: $obsVar^{(t)} = obsValue$. In our example, a vibration occurs at $t = 2$, resulting in the assignment $Vibration^{(2)} = OCCURRED$. These assignments can be directly expressed as soft-constraint function.

4.1 Solving Constraints to Enumerate Most Likely System Trajectories

The three described soft constraint encodings (PHCA model, plan, observations) form a COP that captures the probabilistic behavior of the system over a horizon of N time steps. The model encoding can be done offline, while the plan and the observations have to be encoded and added to the COP online. The effect of adding the plan and observations constraints is that they render certain PHCA trajectories impossible (zero probability). For example, the observation of a vibration renders impossible the trajectory which doesn't entail an observed vibration. The goal constraint, however, is *not* added to the COP, since adding this constraint would render all non-goal-achieving trajectories impossible (we need these failure trajectories for normalization in computing the plan's success probability, as shown in the next section).

For a given plan $\mathcal{P}$ and available observations, we then enumerate the k best solutions to the COP. These correspond to the system's most likely execution trajectories, or diagnoses, within the N-step horizon. An execution trajectory is a sequence of markings for each time step, encoded as assignment to location variables. These are the variables of interest for our COP. For example, Table 1 shows the most likely execution trajectory of the example PHCA, given production plan $\mathcal{P}_{prod} = (a_{cut}, a_{assemble}, a_{cut})$ and observation $Vibration^{(2)} = OCCURRED$.

Technically, the k-best enumeration is done by translating the generated COP (as part of the compilation step) into the weighted CSP format as used by the soft constraint solver toolbar (Bouveret *et al.*, 2004). In the online step, we used a modified version of toolbar that implements mini-bucket elimination to generate a search heuristic for the problem. The heuristic is used by a subsequent A* search to enumerate the k-best solutions. This approach is described in more detail in (Kask and Dechter, 1999).

4.2 Combining Plan Tracking and Prognosis

In the previous section, we described a method to track plan execution within an N-step time window based on a system model and observations. To assess a plan's probability of success, we require not only to track past system behavior, but also to predict its evolution

[1] Where $\{T | Source(T) = P\}$ is short for $\{T \in \mathcal{T} | Source(T) = P\}$.

Table 1: Most probable PHCA trajectory for production plan $\mathcal{P}_{prod} = (a_{cut}, a_{assemble}, a_{cut})$, given a vibration occurred at $t = 2$. A shown variable $X_L^{(t)}$ indicates a marking of location L at time t.

time	marking
0	$assembly.vibration.idle_L^{(0)}$, $assembly.idle_L^{(0)}$, $machining.idle_L^{(0)}$
1	$assembly.vibration.idle_L^{(1)}$, $assembly.idle_L^{(1)}$, $machining.cut_L^{(1)}$
2	$assembly.vibration.vibration_L^{(2)}$, $assembly.assemble_L^{(2)}$, $machining.idle_L^{(2)}$
3	$assembly.vibration.idle_L^{(3)}$, $assembly.idle_L^{(3)}$, $machining.cut_L^{(3)}$

in the future. In principle, this could be accomplished in two separate steps: first, assess the system's state given the past behavior, and then predict its future behavior given this belief state and the plan. However, this two-step approach leads to a problem. Computing a belief state (complete set of diagnoses) is intractable, thus it must be replaced by some approximation (such as considering only k most likely diagnoses (Kurien and Nayak, 2000)). But if a plan uses a certain component intensely, then this component's failure probability is relevant for assessing this plan, even if it is very low and therefore would not appear in the approximation. In other words, the plan to be assessed determines which parts of the belief state (diagnoses) are relevant.

To address this mutual dependency, we propose a method that performs diagnosis and plan assessment *simultaneously*, by framing it as a single optimization problem. The key idea is as follows: The optimization problem formulation is independent of where the present time point is within the N-step time window. We therefore choose it such that the time window covers the remaining future plan actions as well as the past behavior. Now solutions to the COP are system trajectories which start in the past and end in the future. We then compute a plan's success probability by summing over trajectories that achieve the goal. Again due to complexity reasons, we approximate the success probability by generating only the k most probable trajectories. But since we have only a single optimization problem now, we don't have to prematurely cut off unlikely hypotheses and have only one source of error, compared to approximating the belief state and predicting the plan's evolution based on this estimate.

4.3 Approximating the Plan Success Probability

We denote the set of all trajectories as Θ and the set of the k-best trajectories as Θ^*. A trajectory is considered successful if it entails the plan's goal constraint. We define $SUCCESS := \{\theta \in \Theta | \forall s \in \mathcal{R}_{sol}, s \downarrow_Y = \theta : F_G(s) = \text{true}\}$, where $\mathcal{R}_{sol}$ is the set of all solutions to the probabilistic constraint optimization problem, $s \downarrow_Y$ their projection on marking variables, and $F_G(s)$ is the goal constraint. $SUCCESS^*$ is the set of successful trajectories among Θ^*. The exact success probability is computed as

$$\begin{aligned} P(SUCCESS|Obs,\mathcal{P}) &= \\ \sum_{\theta\in SUCCESS} P(\theta|Obs,\mathcal{P}) &= \\ \sum_{\theta\in SUCCESS} \frac{P(\theta,Obs,\mathcal{P})}{P(Obs,\mathcal{P})} &= \\ \sum_{\theta\in SUCCESS} \frac{P(\theta,Obs,\mathcal{P})}{\sum_{\theta\in\Theta} P(\theta,Obs,\mathcal{P})} &= \\ \frac{\sum_{\theta\in SUCCESS} P(\theta,Obs,\mathcal{P})}{\sum_{\theta\in\Theta} P(\theta,Obs,\mathcal{P})} & \end{aligned}$$

The approximate success probability $P^*(SUCCESS^*|Obs,\mathcal{P})$ is computed the same way, only $SUCCESS$ is replaced with $SUCCESS^*$ and Θ with Θ^*. $E(k) := |P(SUCCESS|Obs,\mathcal{P}) - P^*(SUCCESS^*|Obs,\mathcal{P})|$ is the error of the above k-best approximation. It converges to zero as k goes to infinity. Also, $E(k) = 0$ if $P(SUCCESS|Obs,\mathcal{P})$ is 0 or 1. However, as the example in figure 6 shows, $E(k)$ does in general not decrease monotonically with increasing k. Therefore, the question is if any non-trivial bounds can be formulated for $E(k)$. We cannot yet answer this question. However, we suppose that bounds derived for *submodularity optimization* in (Krause and Guestrin, 2007) can be applied to our case. In particular, we conjecture that the trajectory enumeration for plan assessment has the property of *diminishing returns*, that is, the more trajectories are enumerated, the less is learned about the plan's success probability. Submodular objective functions generalize the concept of diminishing returns; if a submodular, non-decreasing objective function $F(k)$ can be derived from $E(k)$, and our best-first A* algorithm can be cast as an instance of the greedy algorithmic scheme in (Krause and Guestrin, 2007), then these bounds also hold for our algorithm.

4.4 Algorithm for Plan Evaluation

Plans are generated by the planner and then advanced until they are finished or new observations are available. In the latter case the currently executed plan is evaluated using Algorithm 1. It first computes the k-best solutions to the COP using an external solver (toolbar in our case). This results in the k most probable trajectories. Then, using these trajectories, it approximates the success probability of plan $\mathcal{P}$ and finally compares the probability against the two thresholds ω_{success} and ω_{fail}. Now we have to address one of three cases: (1) The probability is above ω_{success}, i.e. the plan will probably succeed, (2) the probability is below ω_{fail}, i.e. the plan will probably fail or (3) the probability is in between both thresholds, which means the case cannot be decided. In the first case we simply continue execution. In the second case we have to adapt the plan to the new situation. This is done by REPLAN($\mathcal{P}$, Θ^*), which modifies the future actions of $\mathcal{P}$ taking into account the diagnostic information contained in Θ^*. The third case indicates that not enough information about the system's current state is available. As a reaction, the procedure

ReplanPervasiveDiagnosis($\mathcal{P}$, Θ^*) implements a recently developed method called *pervasive diagnosis* (Kuhn *et al.*, 2008). It addresses this problem by augmenting a plan with information gathering actions (we do not detail the procedures RePlan and ReplanPervasiveDiagnosis as they are beyond this paper's scope).

Algorithm 1

```
procedure EvaluatePlan($\mathcal{R} = (X, D, C)$, $Obs$, $\mathcal{P}$)
    $\mathcal{R}$' ← add constraints over $Obs$ and $\mathcal{P}$ to $\mathcal{R}$
    $\Theta^*$ ← $k$-best solutions of $\mathcal{R}$' for $Y$
    $p \leftarrow P^*(SUCCESS^*|Obs, \mathcal{P})$
    if $p > \omega_{\text{success}}$ then return
    else if $p < \omega_{\text{fail}}$ then
        stop execution of $\mathcal{P}$
        RePlan ($\mathcal{P}$, $\Theta^*$)
    else
        stop execution of $\mathcal{P}$
        ReplanPervasiveDiagnosis($\mathcal{P}$, $\Theta^*$)
    end if
end procedure
```

5 EXPERIMENTAL RESULTS

We ran experiments for five small variations of our example scenario, where $\mathcal{P}_{prod}$ uses the machining station zero to four times. The time window size N accordingly ranges from 2 to 6, problem sizes range from 240 to 640 variables and 240 to 670 constraints. Figure 6 shows the success probabilities for different $\mathcal{P}_{prod}$ and k. Table 2 shows the runtime in seconds and the peak memory consumption in megabytes for computing success probabilities in the planning scenarios, additionally ranging over different values for the mini-bucket parameter i. As expected, with increasing use of the machining stationing station, $P^*(SUCCESS^*|Obs, \mathcal{P}_{prod})$ decreases. Also, runtime increases for larger time windows. The effect of approximation (choosing lower k) is that $P^*(SUCCESS^*|Obs, \mathcal{P}_{prod})$ increasingly deviates from the exact solution. In our example, the approximation tends to be optimistic. In general, however, we think that $P^*(SUCCESS^*|Obs, \mathcal{P}_{prod})$ can be pessimistic, if success trajectories are pruned first when decreasing k. Increasing k hardly seems to affect the runtime, especially if the mini-bucket search heuristic is strong (bigger i-values). For weaker heuristics the influence increases slightly. Memory consumption is affected much stronger by k. Here also, a weaker search heuristic means stronger influence of k.

6 RELATED WORK

In probabilistic verification of model-based programs (Mahtab *et al.*, 2004), the problem is to determine the most likely circumstances under which a high-level control program drives the system towards a goal violating state. A plan can be understood as such a high level control program; so in general, this problem is similar to the plan assessment problem. However, our problem differs in that we are interested in the *set* of all goal achieving system trajectories, from which we derive the plan's success probability, while for the verification problem, only the single most probable goal violating trajectories are interesting. Therefore we have to go one step further, not only enumerating the trajectories, but also summing over them to compute the success probability.

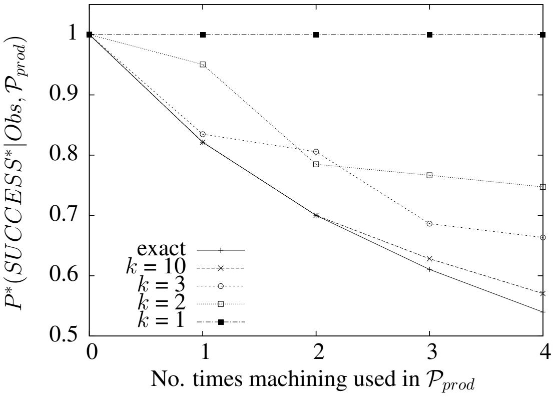

Figure 6: Approximate success probability (y-axis) of plan $\mathcal{P}_{prod}$ against varying usage of the machining station (x-axis) after the observation of a vibration at $t = 2$.

McDermott (McDermott, 1993) and Beetz's (Beetz, 2000) Reactive Plan Language (RPL) chooses a different approach to deal with system failures and uncertainty. It uses a hierarchical task decomposition, breaking down top level goals to a finer granularity recursively. The plan itself is not a sequence of actions but executable code. The language allows reasoning on and transformation of the plans. Heuristic routines attain the subgoals and cope with failures and unexpected events during the execution. A goal for finding a cup could e.g. look in the dishwasher after seeing that no cups are left in the cupboard. This approach is particularly promising in domains of high uncertainty, where classical planning fails. However, the RPL approach currently neglects explicit diagnosis techniques and relies on the observability of relevant environment states.

7 CONCLUSION AND FUTURE WORK

We presented a model-based method that combines diagnosis of past execution steps with prognosis of future execution steps of production plans, in order to allow the production system to autonomously react to failures and other unforeseen events. The method makes use of probabilistic constraint optimization to efficiently solve this combined diagnosis/prognosis problem. Preliminary results for a real-world machining scenario show it can indeed be used to guide the system away from plans that rely on suspect system components. Future work will concern the integration of the method into our overall planning/execution architecture, and its extension to multiple, simultaneously executed plans. We are also interested in exploiting the plan diagnosis/prognosis results in order

Table 2: Runtime in seconds / peak memory consumption in megabytes. (e) indicates that the exact success probability $P(SUCCESS|Obs, \mathcal{P}_{prod})$ could be computed with this configuration. (mem) indicates that A* ran out of memory (artificial cutoff at $>$ 1 GB, experiments were run on a Linux computer with a recent dual core 2.2 Ghz CPU with 2 GB RAM).

		No. times machining used in $\mathcal{P}_{prod}$ (window size N, #Variables, #Constraints)				
k	i	0 (2, 239, 242)	1 (3, 340, 349)	2 (4, 441, 456)	3 (5, 542, 563)	4 (6, 643, 670)
1	10	$<$ 0.1 / 1.8	0.1 / 6.8	0.1 / 19.0	(mem)	(mem)
	15	0.1 / 1.9	0.3 / 4.2	0.5 / 7.8	0.5 / 16.6	0.8 / 32.0
	20	0.1 / 1.9	0.5 / 5.2	3.7 / 20.1	6.5 / 34.5	9.5 / 50.7
2	10	$<$ 0.1 / 2.1	0.1 / 11.9	0.2 / 38.5	(mem)	(mem)
	15	0.1 / 2.2	0.3 / 5.4	0.5 / 9.7	0.6 / 28.0	0.8 / 52.0
	20	0.1 / 2.2	0.5 / 6.4	3.7 / 21.8	6.5 / 37.2	9.5 / 55.8
3	10	$<$ 0.1 / 2.3 (e)	0.1 / 11.9	0.2 / 40.1	(mem)	(mem)
	15	0.1 / 2.4 (e)	0.3 / 5.4	0.5 / 11.4	0.6 / 29.9	0.9 / 55.5
	20	0.1 / 2.4 (e)	0.5 / 6.4	3.7 / 23.5	6.6 / 38.3	9.5 / 57.4
4	10	(e)	0.1 12.5	0.2 / 40.1	(mem)	(mem)
	15	(e)	0.3 / 5.9	0.5 / 11.4	0.6 / 30.9	0.9 / 57.2
	20	(e)	0.5 / 6.9	3.7 / 23.5	6.6 / 39.3	9.5 / 59.1
5	10	(e)	0.1 / 13.1	0.2 / 40.7	(mem)	(mem)
	15	(e)	0.3 / 6.6	0.5 / 12.0	0.6 / 33.6	0.9 / 59.5
	20	(e)	0.5 / 7.6	3.7 / 24.0	6.6 / 42.8	9.5 / 63.9
10	10	(e)	0.1 / 14.0 (e)	0.2 / 43.4 (e)	(mem)	(mem)
	15	(e)	0.3 / 6.7 (e)	0.5 / 14.7 (e)	0.6 / 36.2	0.9 / 64.8
	20	(e)	0.6 / 7.7 (e)	3.8 / 26.6 (e)	6.6 / 45.8	9.6 / 68.9

to update the underlying system model, for instance, to automatically adapt to parameter drifts or wear of components.

ACKNOWLEDGMENTS

This work is supported by DFG and CoTeSys.

NOMENCLATURE

$a^{(t)}$	action to be executed at time t
COP $\mathcal{R}$	constraint optimization problem $\mathcal{R}$
G	a plan's goal
k	number of hypotheses/system trajectories
N	size of time window in time steps
ω_{success}	plan success threshold
ω_{fail}	plan failure threshold
$\mathcal{P}$	general production plan
$\mathcal{P}_{prod}$	example production plan
PHCA	probabilistic hierarchical constraint automata
Θ	set of all system trajectories
Θ^*	set of k best trajectories
Y	COP solution variables

REFERENCES

(Beetz *et al.*, 2007) Michael Beetz, Martin Buss, and Dirk Wollherr. Cognitive technical systems — what is the role of artificial intelligence? In *Proc. KI-2007*, pages 19–42, 2007.

(Beetz, 2000) Michael Beetz. *Concurrent Reactive Plans: Anticipating and Forestalling Execution Failures*, volume 1772 of *Lecture Notes in Artificial Intelligence*. Springer Publishers, 2000.

(Bouveret *et al.*, 2004) S. Bouveret, F. Heras, S.de Givry, J. Larrosa, M. Sanchez, and T. Schiex. Toolbar: a state-of-the-art platform for wcsp. www.inra.fr/mia/T/degivry/ToolBar.pdf, 2004.

(Kask and Dechter, 1999) Kalev Kask and Rina Dechter. Mini-bucket heuristics for improved search. In *Proc. UAI-1999*, pages 314–32, 1999.

(Krause and Guestrin, 2007) Andreas Krause and Carlos Guestrin. Near-optimal observation selection using submodular functions. In *Proc. AAAI-2007*, pages 1650–1654. AAAI Press, 2007.

(Kuhn *et al.*, 2008) Lukas Kuhn, Bob Price, Johan de Kleer, Minh Binh Do, and Rong Zhou. Pervasive diagnosis: The integration of diagnostic goals into production plans. In *Proc. AAAI-2008*, 2008.

(Kurien and Nayak, 2000) James Kurien and P. Pandurang Nayak. Back to the future for consistency-based trajectory tracking. In *Proc. AAAI-2000*, pages 370–377, 2000.

(Mahtab *et al.*, 2004) Tazeen Mahtab, Greg Sullivan, and Brian C. Williams. Automated Verification of Model-based Programs Under Uncertainty. In *Proceedings 4th International Conference on Intelligent Systems Design and Application*, 2004.

(McDermott, 1993) Drew McDermott. A reactive plan language. Technical report, Yale University, Computer Science Dept., 1993.

(Mikaelian *et al.*, 2005) Tsoline Mikaelian, Brian C. Williams, and Martin Sachenbacher. Model-based Monitoring and Diagnosis of Systems with Software-Extended Behavior. In *Proc. AAAI-05*, 2005.

(Schiex *et al.*, 1995) Thomas Schiex, Hélène Fargier, and Gerard Verfaillie. Valued constraint satisfaction problems: hard and easy problems. In *Proc. IJCAI-1995*, 1995.

(Williams *et al.*, 2001) Brian C. Williams, Seung Chung, and Vineet Gupta. Mode estimation of model-based programs: monitoring systems with complex behavior. In *Proc. IJCAI-2001*, pages 579–590, 2001.