Integrated Diagnosis and Plan Assessment for Autonomous Production Processes[*]

Paul Maier, Martin Sachenbacher, Thomas Rühr
Technische Universität München
85748 Garching, Germany
{maierpa,ruehr,sachenba}@in.tum.de

Lukas Kuhn
PARC
Palo Alto, 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 products, the industry struggles to implement production systems that demonstrate the necessary flexibility while maintaining cost efficiency comparable to highly automated mass production. The need for human workforce for the setup, the development of processes and quality assurance systems is a main cost driver in automated production. The high 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. 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.

The German research cluster "Cognition for Technical Systems" [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 typically have only few available sensors, and therefore it cannot be reliably observed whether an individual manufacturing step went indeed as planned. Instead, this becomes only gradually more certain while the production plans are being executed. Therefore, in the presence of faults or other unforeseen behavior, the question arises how likely it is that plans that are ready for execution or already being executed will 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 success probability of production plans in execution. We assume that a planner provides plans given a system model. A plan is a sequence of action and start time pairs where each action is executed at the corresponding start time. Whenever the system produces an observation, it will be forwarded to a module that performs simultaneous plan assessment 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. 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 ap-

[*]This work is supported by DFG and CoTeSys.

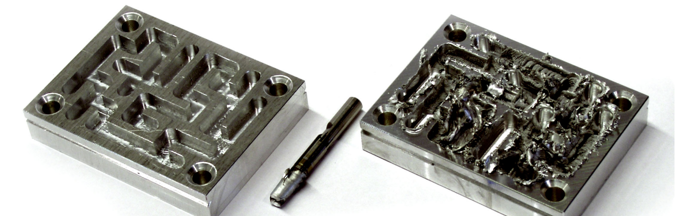

Figure 1: Effects of milling tool deterioration until breakage in machining. Image (c) Prof. Shea TUM PE

proach informally with an example, and then present our algorithmic solution and some experimental results.

2 Example: Metal Machining and Assembly

At TUM, a customized and extended Flexible Manufacturing System (FMS) based on the iCim3000 from Festo AG is installed for evaluation purposes (see figure 4). It resembles a typical setup for fully automated manufacturing and assembly of small products made from machined metal. The system consists of a conveyor transport and three stations: storage, machining (milling and turning), and assembly.

All products are transported on custom pallets which can be handled by the stations' robots and the pallet carriers of the palletized conveyor transport. For the handling of pallets and parts at the assembly and machining stations, Mitsubishi RV3SB robots are used. A linear three axis storage robot is used for handing over pallets between the storage buffers and pallet carriers on the conveyor. Machining capabilities are provided via a milling and a turning machine, which are loaded by a robot mounted on a linear axis enabling it to travel between the two CNC machines and the conveyor stopper position. The setup was recreated in a physical 3D simulation[1] in order to facilitate software development and testing.

For our model-based method, we need a model of this manufacturing system which allows to track system behavior over time, including unlikely component faults. The model is shown in figure 2. We model the setup using the probabilistic hierarchical constraint automata (PHCA) framework, which we explain in detail later. In our example we use a simplified setup consisting only of a milling and an assembly station. Both components have an idle state and a work state. The milling station model can transition to a "drill blunt" composite state, where its behavior is basically unchanged, but abrasions are caused during operation due to a blunt drill. Also, in this state it's very probable (probability 0.5) that the drill breaks, leading to a failure state. The assembly station model contains a composite state which models occasional abrasions, occurring in each time step with probability 0.2.

Two products are being produced using a single production plan $\mathcal{P}_{prod}$: (1) a toy maze consisting of an alloy base plate and an acrylic glass cover, held together by pins, and (2) an alloy part of a robot arm (see figure 3), which is used in manual assembly later. The production plan $\mathcal{P}_{prod}$ consists of these steps: (1) drill holes and maze into base plate, (2) assemble base plate and cover (3,4,5,6) mill robot arm part.

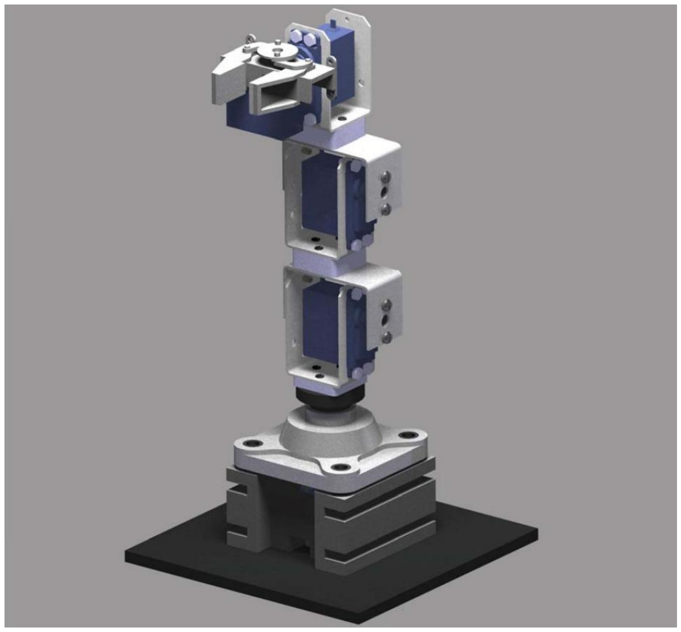

Figure 3: The robot arm product. (c) Prof. Shea TUM PE

All steps require a single time step, except for the milling of the robot arm part, which can take up to four time steps (yielding variations of $\mathcal{P}_{prod}$). Thus the plan takes 2 - 6 time steps (starting at $t = 0$ and ending at $t = 6$ at its latest, with the last step starting at $t = 5$). To keep the example simple, we did not model transportation events, such as conveyor belts or robot arms fetching parts from the storage shown in figure 4.

The plan is considered successful if both products are flawless. A product is flawless only if the product quality is ok, modeled through a model variable $PF^{(t)} \in \{OK, FAULTY\}$ (see figure 2). In the example, a product is flawed only if the drill of the milling station breaks ($PF^{(t)} = FAULTY$). In other words, as long as the drill doesn't break, the production plan will succeed.

A vibration sensor at the assembly station allows partial binary observations of the form "abrasion occurred" and "no abrasion occurred". Abrasions may occur in the milling station due to a blunt drill or in the assembly station due to rough assembly steps. The observation is partial because the sensor doesn't differentiate between these two causes. The drill going blunt is slightly less likely than an abrasion occurring in the assembly. Abrasions in the assembly don't affect the plan; a blunt drill however is likely to break and then ruin any milled products (see figure 1).

In our scenario, after the second plan step (assembling the maze base plate and its cover) at $t = 2$ an abrasion is observed ($Abrasion^{(2)} = OCCURRED$). Two hypotheses can explain this observation: (a) the abrasion occurred within the assembly, or (b) a blunt drill caused an abrasion in the milling station. Hypothesis (a) is slightly more likely than (b). How-

[1]Based on Gazebo, `http://playerstage.sourceforge.net/gazebo/gazebo.html`(March, 2009).

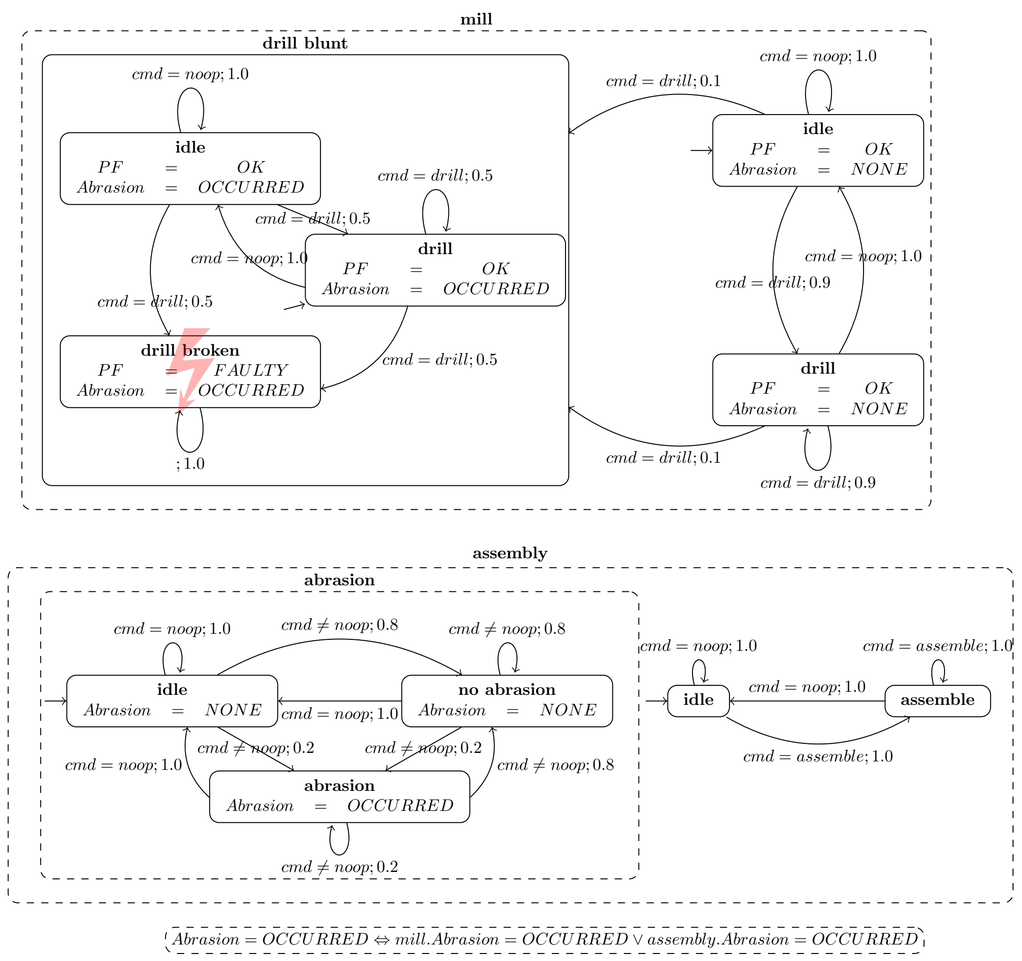

Figure 2: Simplified PHCA of the manufacturing system. Two components are modeled as parallel running complex locations (indicated by dashed borders): a milling station and an assembly station. Variables appearing within such a location are local to this location. $mill.cmd$ refers globally to the command variable cmd within complex location $mill$. The dependent variable $PF^{(t)} \in \{OK, FAULTY\}$ models the product feature. In this example, a broken drill deteriorates the product feature beyond usability ($PF^{(t)} = FAULTY$), which means that the production plan fails. A single observation is possible: whether an abrasion has occurred ($Abrasion^{(t)} = OCCURRED$) or not ($Abrasion^{(t)} = NONE$). An abrasion may be caused by a blunt drill in the milling station or within the assembly station.

ever, (b) contains the possibility of a fatal breakdown (drill breaks). The question for the planner now is: How likely is it that the current plan still succeeds?

In the following, we describe a method which computes this likelihood by estimating most likely states from past observations, while at the same time it assesses the remaining success probability by projecting the current production plan into the future. Before we describe our method in detail, we examine the necessary prerequisites.

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 $< \Sigma, P_\Xi, \Pi, O, Cmd, \mathcal{C}, P_T) >$, 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 composite 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

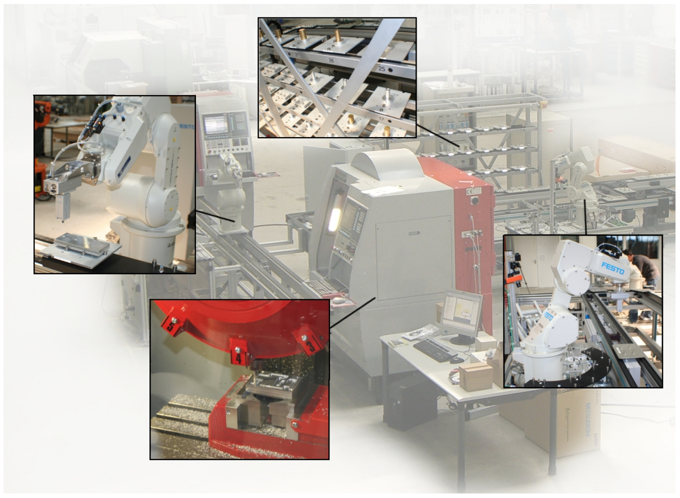

Figure 4: The hardware setup used for experimentation, showing storage, transport, robot and machining components.

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.
- $\mathcal{C} : \Sigma \to \mathcal{C}[\Pi]$ 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 $T : \Sigma_p^{(t)} \times \mathcal{C}[\Pi]^{(t)} \to 2^{\Sigma^{(t+1)}}$. 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 $\mathcal{G} : \mathcal{T} \to \mathcal{C}[\Pi]$ 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 2 illustrates the PHCA definition. The main factory components $mill$ 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 2 whose behavioral PHCA constraint encodes that an observed abrasion is caused by one of the two components or both. Behavioral and guard PHCA constraints express which observations and commands are consistent with which locations and transition guards. Primitive locations are for example $mill.idle$ and $mill.drill$, which encode the milling station being in an idle state and working on a piece. An example for an observable variable is $Abrasion$, which encodes whether an abrasion has occurred or not. The dependent variables $mill.Abrasion$ and $assembly.Abrasion$ encode for each component whether it caused an abrasion. A command variable is, e.g., $mill.cmd$. It occurs in the guard PHCA constraint for transition $idle \to drill$ within composite location $mill$: $mill.cmd = drill$. The transition is non-deterministic: Given the guard is satisfied, it is taken with probability 0.9. The remaining possibility (completing the conditional probability distribution) is the transition from $idle$ to the composite location $drill\,blunt$, which has the same guard and is taken with probability 0.1.

4 Plan Tracking as Constraint Optimization

Plan assessment requires tracking of the system's plan-induced evolution. In our case, it means tracking the evolution of PHCA markings. In previous work [Mikaelian *et al.*, 2005] we introduced an encoding of PHCA as soft constraints and casted the problem of tracking PHCA markings within an N-stage time window as a soft constraint optimization problem [Schiex *et al.*, 1995]. The solutions to this problem are the most probable trajectories (sequences of markings) within this time window. 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.

4.1 Encoding of PHCA Models as Probabilistic 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 $D = \{D_1, \dots, D_n\}$, and $C = \{C_1, \dots, C_n\}$ is a set of constraints (S_i, F_i) with scope $S_i = \{X_{i1}, \dots, X_{ik}\} \subseteq X$ and a constraint function $F_i : D_{i1} \times \dots \times D_{ik} \to [0, 1]$. 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 $X_\Sigma^{(t)} = \{L_1^{(t)}, ..., L_n^{(t)}\}$ 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 $D_{X_\Sigma} = \{Marked, Unmarked\}$ is the domain for each variable 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 interdependencies encoded in behavior PHCA constraints, e.g., that an abrasion occurs iff an abrasion occurs in the mill or the 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:[2] $(\forall t \in \{0..N-1\}, \forall P \in \Sigma_p : (\exists \tau \in \{T | Source(T) = P\} \Rightarrow [P^{(t)} = Marked \Leftrightarrow (\exists T \in \{T | Source(T) = P\} : T^{(t)} = Enabled \wedge (\forall T' \in (\{T | Source(T) = P\} - \{T\}) : T'^{(t)} = Disabled))] \bigwedge [P^{(t)} = Unmarked \Leftrightarrow (\forall T \in \{T | Source(T) = P\} : T^{(t)} = Disabled)]))$

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].

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

$mill.cmd^{(0)}$	$assembly.cmd^{(1)}$	$mill.cmd^{(2)}$	p
drill	*assemble*	*drill*	1.0

Table 1: Command constraint for plan $\mathcal{P}_{prod}$. Assignments mapped to 0.0 are omitted as well as *noop* commands.

4.2 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 $a_{drill}^{(t)}$ and $a_{assemble}^{(t)}$ are assignments $mill.cmd^t = drill \wedge assembly.cmd^t = noop$ and $assembly.cmd^t = assemble \wedge mill.cmd^t = noop$. $\mathcal{P}$ is then mapped to the following logical constraint: $\forall t \in \{0..N\} : a^{(t)}$.

As an example table 1 shows the command soft-constraint for plan $\mathcal{P}_{prod} = (a_{drill}, a_{assemble}, a_{drill})$ (with the second drill operation consisting of only a single drill step) from the example scenario.

Now we consider the plan's goal, which is generally to produce a flawless product. We encode this informal description as a logical constraint over product feature variables at the end of the execution, t_{end}: $G \equiv \forall PF^{(t_{end})} \in RelevantFeatures(\mathcal{P}) : PF^{(t_{end})} = OK$. $RelevantFeatures()$ 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 fails. In our example, there is only a single product feature PF, which is faulty if the drill is broken. The goal constraint for the above mentioned plan (three time steps long) is accordingly $PF^{(3)} = OK$.

$Abrasion^{(2)}$	p
$OCCURRED$	1.0

Table 2: Constraint for observation $Abrasion^{(2)} = OCCURRED$. Assignments mapped to 0.0 are omitted.

4.3 Encoding Observations as Constraints

Observations made during the plan execution (such as the occurred abrasion 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 basically encoded as an assignment to a corresponding observable variable: $obsVar^{(t)} = obsValue$. In our example, an abrasion occurs at $t = 2$, resulting in the assignment $Abrasion^{(2)} = OCCURRED$. These assignments can be directly expressed as soft-constraint function, as shown in table 2.

5 Solving Soft Constraints to Enumerate Most Probable System Trajectories

Together, 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 solutions of the COP are system state trajectories, or more precisely, PHCA marking trajectories, which can be used to compute a plan's success probability.

For a given plan $\mathcal{P}$ we enumerate the system's k most likely execution traces or trajectories as the k best solutions to the COP. An execution trajectory is simply 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 3 shows the most likely execution trajectory of the example PHCA, given production plan $\mathcal{P}_{prod} = (a_{drill}, a_{assemble}, a_{drill})$ and observation $Abrasion^{(2)} = OCCURRED$.

The effect of the plan actions and the observations is that these additional constraints render certain PHCA trajectories impossible (zero probability). For example, the observation of an abrasion renders impossible the trajectory which doesn't entail an observed abrasion. The goal constraint, however, is *not* added to the COP, since adding this constraint would render all non-goal-achieving or plan failure trajectories impossible. But these are needed for normalization in computing the success probability, as we will see shortly.

The k-best enumeration is done by translating the generated COP (as part of the compilation step) into the WCSP 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].

time	marking
0	$assembly.abrasion.idle_L^{(0)}$, $assembly.idle_L^{(0)}$, $mill.idle_L^{(0)}$
1	$assembly.abrasion.idle_L^{(1)}$, $assembly.idle_L^{(1)}$, $mill.drill_L^{(1)}$
2	$assembly.abrasion.abrasion_L^{(2)}$, $assembly.assemble_L^{(2)}$, $mill.idle_L^{(2)}$
3	$assembly.abrasion.idle_L^{(3)}$, $assembly.idle_L^{(3)}$, $mill.drill_L^{(3)}$

Table 3: Most probable PHCA trajectory for production plan $\mathcal{P}_{prod} = (a_{drill}, a_{assemble}, a_{drill})$, given an abrasion occurred at $t = 2$. A shown variable $X_L^{(t)}$ indicates a marking of location L at time t.

6 Combining Plan Tracking and Prognosis

In the previous sections, 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 analyze past behavior, but also to predict its evolution in the future. In principle, this could be accomplished in two steps: first, diagnose the system given the past behavior, and then predict its future behavior given these diagnoses and the plan. However, this two-step-approach leads to a problem. Computing a complete set of diagnoses (belief state) is intractable in general, and thus the first step must be replaced by some approximation (e.g., computing only k most likely diagnoses [Kurien and Nayak, 2000]). But if a plan uses a certain component intensely, then the failure probability of this component is relevant for assessing this plan, even if it is very low and therefore would not appear in the set of most likely diagnoses. That is, the plan to be assessed determines which diagnoses are relevant.

To address this problem, 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 COP formulation (see previous section) is independent of where the present time point is within the N-step time window. It can be chosen to be the last time point (window completely in the past, tracking only), the first time point (window completely in the future, prognosis only), or somewhere in the middle (tracking and prognosis combined). In our case, we simply shift the time window just as many time steps into the future as there are remaining future plan actions, and choose the present time point accordingly. Now solutions to the COP are trajectories which start in the past and end in the future.

We then compute a plan's success probability by summing over system trajectories that achieve the goal. We cannot do this exactly, again due to complexity reasons. Therefore, we approximate the success probability by generating only the k most probable ones. Since we have only a single optimization problem now, we only have one source of error, compared to two when explicitly computing the belief state and separately predicting the plan's evolution. Another advantage is that we can deal with delayed symptoms as described in [Kurien and Nayak, 2000], since we don't have to prematurely cut off unlikely hypotheses.

6.1 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^*.

6.2 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 don't detail the procedures REPLAN and REPLANPERVASIVEDIAGNOSIS here as it would exceed the scope of this paper.

Algorithm 1

```
 1: procedure EVALUATEPLAN($\mathcal{R} = (X, D, C)$, $Obs$, $\mathcal{P}$)
 2:     $\mathcal{R}$' ← add constraints over $Obs$ and $\mathcal{P}$ to $\mathcal{R}$
 3:     $\Theta^*$ ← $k$-best solutions of $\mathcal{R}$' for $Y$
 4:     $p \leftarrow P^*(SUCCESS^*|Obs, \mathcal{P})$
 5:     if $p > \omega_{\text{success}}$ then return
 6:     else if $p < \omega_{\text{fail}}$ then
 7:         stop execution of $\mathcal{P}$
 8:         REPLAN ($\mathcal{P}$, $\Theta^*$)
 9:     else
10:         stop execution of $\mathcal{P}$
11:         REPLANPERVASIVEDIAGNOSIS($\mathcal{P}$, $\Theta^*$)
12:     end if
13: end procedure
```

7 Experimental Results

We ran experiments on the compiled COP of the example PHCA and scenario described in section 2. The COP had 744 binary variables and 777 constraints. Generating the best 25 trajectories (which in our example includes all trajectories with nonzero probability, allowing to compute the exact success probability) in the online step took approximately five seconds[3].

First results indicate that the general plan assessment approach works as expected. Figure 5 shows the success probabilities of small variations of $\mathcal{P}_{prod}$ from our example. The variations make different use of the milling station (0 - 4 times). It can be seen that the more $\mathcal{P}_{prod}$ utilizes the drill, the less probable the plan is to succeed. Furthermore figure 5 shows the effect of approximation. Decreasing k in our k-best approach means that fewer trajectories are enumerated and thus the approximate success probability $P^*(SUCCESS^*|Obs, \mathcal{P})$ deviates from the exact success probability $P(SUCCESS|Obs, \mathcal{P})$ (solid line) [4].

8 Related Work

Similar to plan assessment is the problem of probabilistic verification of model-based programs [Mahtab *et al.*, 2004]. Given a high-level control program, a goal and a model of a fault-aware system, composed of software and hardware components (modeling the possibility of probabilistic failure), the problem is to determine the most likely circumstances under which the 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, these problems are similar: Predicting the behavior of a (non-deterministic) system (soft- and hardware) driven by a plan, given certain observations. 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. So basically, we have to

[3]On a machine with a recent 2.2GHz dualcore CPU and 2 GB of RAM.

[4]Computing the exact success probability was possible in our relatively small example.

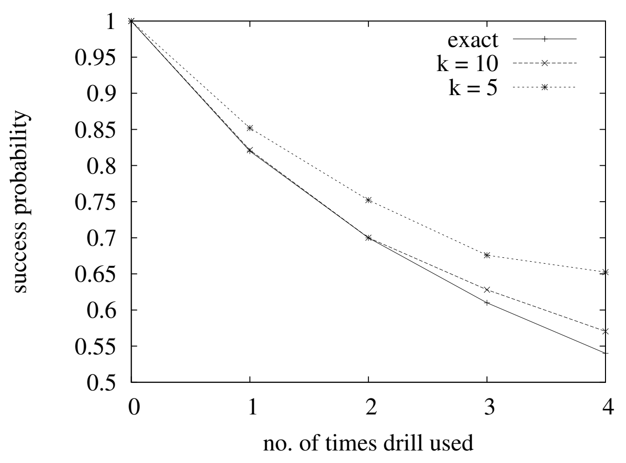

Figure 5: Approximate success probability (y-axis) of plan $\mathcal{P}_{prod}$ against varying usage of the milling station (0,1,2, …times, x-axis) after the observation at $t = 2$. The different plots show the approximation with different values for k.

go one step further, not only enumerating the trajectories, but also summing over them to compute the success probability.

McDermott [McDermott, 1993] and Beetz's [Beetz, 2000] Reactive Plan Language (RPL) chooses a different approach to deal with system failures and uncertainty. It employs a hierarchical task decomposition, breaking down top level goals to a finer granularity recursively. The plan itself is not an abstract sequence of symbols 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.

9 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 has been implemented, and 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 simultaneous plans. We are also interested in exploiting the plan diagnosis/prognosis results in order to update the underlying model, for instance, to automatically adapt to parameter drifts.

References

[Beetz *et al.*, 2007] Michael Beetz, Martin Buss, and Dirk Wollherr. Cognitive technical systems — what is the role of artificial intelligence? In J. Hertzberg, M. Beetz, and R. Englert, editors, *Proceedings of the 30th German Conference on Artificial Intelligence (KI-2007)*, pages 19–42, 2007. Invited paper.

[Beetz, 2000] Michael Beetz. *Concurrent Reactive Plans: Anticipating and Forestalling Execution Failures*, volume LNAI 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. http://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 *Proceedings of the 15th annual conference on uncertainty in artificial intelligence (uai-99)*, pages 314–32, San Francisco, CA, 1999. Morgan Kaufmann.

[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 Dieter Fox and Carla P. Gomes, editors, *Aaai*, pages 1306–1312. AAAI Press, 2008.

[Kurien and Nayak, 2000] James Kurien and P. Pandurang Nayak. Back to the future for consistency-based trajectory tracking. In *AAAI/IAAI*, pages 370–377, 2000.

[Mahtab *et al.*, 2004] Tazeen Mahtab, Greg Sullivan, and C. Brian Williams. Automated Verification of Model-based Programs Under Uncertainty. In *Proceedings of the 4th International Conference on Intelligent Systems Design and Application*, August 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 Chris Mellish, editor, *Ijcai'95: proceedings international joint conference on artificial intelligence*, Montreal, 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-01*, pages 579–590, 2001.