Receding Time Horizon Self-Tracking and Assessment for Autonomous Manufacturing

Paul Maier, Martin Sachenbacher
Department of Informatics, Technische Universität München
Boltzmannstraße 3, 85748 Garching
Phone: +49 (0) 89 289 19595, E-Mail:{ maierpa,sachenba}@in.tum.de

Abstract

Next-generation production and manufacturing plants create individualized products by automatically deriving production schedules from design specifications. However, because planning and scheduling are computationally hard, they must typically be done off-line using a simplified system model, and are consequently unaware of on-line observations and potential component faults. This leads to a self-assessment problem we call *plan assessment*: Given behavior models and current observations of the plant's (possibly faulty) behavior, how likely is it that partially executed manufacturing plans will still succeed? A previously developed self-assessment capability generates k most probable system behaviors as solutions of a constraint optimization problem. However, it only allows single products and doesn't scale well with the schedule length. In this work, we extend the capability towards multiple products and generate k most probable system behaviors only within a small, fixed time window, which is then moved to cover schedules longer than the time window. Experiments show the feasibility of this approximative approach.

Keywords

self-assessment, constraint optimization, model-based reasoning

1 Introduction

Modern factory plants implement complex automation to realize highly autonomous mass production. Current research [BBW07, FVL+06] envisions and develops plants that implement planning and scheduling capabilities based on self-models in order to achieve automated mass production of individualized products, known as „mass customization“. In a typical scenario a factory plant automatically generates manufacturing process plans from given product specifications. The operations are scheduled based on models of the factory stations. Since scheduling and planning are computationally intensive they are usually performed offline, e.g. during the night for production the next day. For the same reason the mentioned models are typically focused: They omit, e.g., knowledge of potential failure behavior. This leads to a problem of evaluating manufacturing plans with respect to online observations, based on models focused on station behavior. In [MSR+10] we formalize this as the *plan assessment problem*: Given a model M_{assess}, the problem of plan assessment is to compute the success probability $P(\mathcal{G}_i|o^{0:t})$ for each manufacturing process plan $\mathcal{P}_i$, or good lower and upper bounds p_l and p_u: $p_l \leq P(\mathcal{G}_i|o^{0:t}) \leq p_u$. $\mathcal{G}_i$ is the event of product i being successfully produced. $P(\mathcal{G}_i|o^{0:t})$ or the bounds then allow to decide whether to a) continue with a plan, b) stop it because it probably won't succeed or c) gather more information. We describe a decision procedure in [MSR+10], whereas in this work, we focus on computing the success probability. Plan assessment 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.

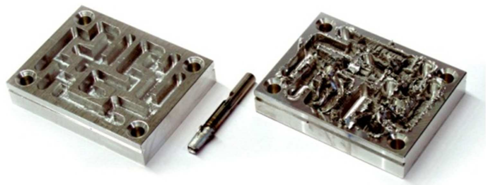

Figure 1-1: Effects of cutter deterioration until breakage in machining.

In earlier work [MSR+09, MSR+10] we proposed a combined diagnosis/prediction method as a solution to this problem, which approximated the success probability for a single product based on a number k of most probable system trajectories, spanning over the complete schedule length N_S. The problem of enumerating such trajectories is however exponential in N_S, and thus in our experiments we found the method did not scale beyond $N_S \approx 14$ time steps. To address this scalability issue, in this work we extend this method towards a receding time horizon scheme [KN00, MWS05], which slides a time window of fixed, small length N along the time line in order to cover realistically sized schedules. Most probable trajectories are only generated within this window, which renders the problem exponential only in $N \ll N_S$. This effectively eliminates one of two exponential dependencies, the second being the number of station and product states considered. Additionally, we extend this plan assessment method to enable plan assessment for multiple products simultaneously.

Closest to our work are verification methods such as probabilistic model checking [RKN+04], online verification [ASB07], or probabilistic verification [MSW04]. Plan assessment could possibly be formulated as a probabilistic model checking problem. However, model checking algorithms do not allow a step-wise approach like our time window shifting, i.e. they have to regard the whole schedule at once. Probabilistic verification [MSW04] deals only with single most likely behaviors, whereas we have to consider many goal-violating (and achieving) behaviors. And all of these methods usually focus on models of single systems such as cars [ASB07]. In contrast, we model a manufacturing facility and the products it processes. We see our self-assessment capability as a complementary block, which can support other self-X methods such as self-optimization [Gau05] in achieving the autonomy needed in the envisioned factories.

2 Example: Metal Machining and Assembly

Our factory test-bed − an iCim3000-based Festo Flexible Manufacturing System − consists of conveyor transports, storage, machining and assembly. It serves as basis for hypothetical example scenarios, where a scheduler schedules the manufacturing of toy mazes (Figure 1-1) and robot arms. 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 (in our abstract scenarios, we disregard transportation). For the robot arm, alloy parts for its brackets have to be machined. On its route through the factory a product might get flawed when worked on by faulty stations. Machining stations are suspicious candidates, because their cutter might blunt during operation and is then more likely to break. A broken cutter severely damages maze products (see Figure 1-1). Vibration sensors provide partial information about possible failures, which is, however, ambiguous: some components generate random vibrations, and thus not every vibration means that a component is faulty. We assume that, with some probability, vibrations in the assembly can trigger signals in sensors of machining stations. Here, a vibration is detected at $t_{\mathrm{vibration}}$, while the machining station is cutting a bracket for the robot and the maze is being assembled (see Figure 2-1). Is the vibration an indicator for a blunt cutter, and how does this possibility affect the plans? This scenario comprises one machining and one assembly station, with one maze and one robot being scheduled for manufacturing.

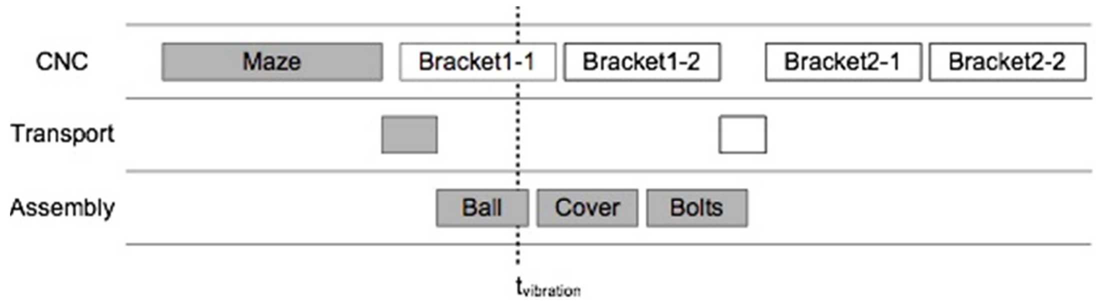

Figure 2-1: A visualisation of a schedule of a maze (dark) and a robot arm (bright).

3 Plan Assessment with Predicted System Behaviors

Computing $P(\mathcal{G}_i|o^{0:t})$ for a manufacturing plan $\mathcal{P}_i$ requires to predict whether $\mathcal{P}_i$ will reach a state that entails a successfully finished product. We use an approach that generates possible system state sequences, called *system trajectories*, over a time window of length N [KN00, MWS05]. We differentiate among trajectories that lead to goal-achieving states and those that lead to goal-violating states. We combine it with k-best approximation, enumerating only the k most probable state sequences within the window, and a receding horizon scheme that shifts the time window along the time line to cover larger schedules. We assume a given schedule $\mathcal{S}$ of manufacturing operations, i.e. a sequence of N_S 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 model that encodes factory station and product behavior of those stations and products occuring in $\mathcal{S}$. It also encodes possible observations. It defines a distribution $Pr(\mathbf{X}^{t+1} \mid \mathbf{X}^t)$ over possible state transitions given the current plant state and a distribution $Pr(\mathbf{O}^t \mid \mathbf{X}^t)$ over possible observations given the current state. $\mathbf{X}^t$ is a vector of variables X^t characterizing a system at time t, where $St^t(M_{\text{assess}})$ is the set of all possible (atomic) assignment vectors for $\mathbf{X}^t$, 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 can go beyond current time t, thereby *predicting system behavior*. Note that hereafter we deviate from standard notation and do *not* write vectors of values/variables bold face, as we mostly deal with vectors. So unless stated otherwise X^t, e.g., will refer to a vector of variables.

We model all products as finite state machines with two states. Thereby system states, with respect to some given modeled product i, can be partitioned into two sets $\mathcal{G}_i$ and $\overline{\mathcal{G}}_i$, respectively representing „product ok“ (goal states) and „product flawed“(goal-violating states). We model the interactions of each (unfinished) product with factory stations, following our underlying assumption that some $\mathcal{P}_i$ succeeds *as long as no station working it fails*. This simple modeling could be extended to cover a richer set of product states than {„product ok", „product flawed"}. $\mathcal{G}_i$ represents the success event, where success means that $\mathcal{P}_i$ generates a product according to its specification. *Success* of $\mathcal{P}_i$ is modeled as a constraint G_i encoding $X_{p_{\text{id}}.\text{ok}}^{(t_{\mathcal{P}_i})} == MARKED$, i.e. that the product identified by p_{id} must be in an ok-state at finishing time $t_{\mathcal{P}_i}$. The single variable $X_{p_{\text{id}}.\text{ok}}^{(t_{\mathcal{P}_i})}$ is part of the plan assessment model. Finally, a plan is *successful* if it entails G_i, i.e. if G_i is consistent with all system states possible after its execution. We define $\mathcal{G}_i$ as the set of all *goal-achieving* trajectories $\mathcal{G}_i = \{\theta \in St(M_{\text{assess}}) \mid \theta G_i\}$ and $\overline{\mathcal{G}}_i = \{\theta \in St(M_{\text{assess}}) \mid \theta G_i\}$. This leads to the following definition of success probability:

Definition 1. Plan Success Probability *Given a model* $\mathrm{M}_{\text{assess}}$ *, a sequence of observations* $\mathrm{o}^{0:\mathrm{t}}$ *from the start of the manufacturing process schedule up to current time* t *and a manufacturing plan* $\mathcal{P}_{\mathrm{i}}$ *, we define the probability that* $\mathcal{P}_{\mathrm{i}}$ *will succeed as*

$$P(\mathcal{G}_i|o^{0:t}) = \sum_{\theta\in\mathcal{G}_i} Pr(\theta \mid o^{0:t}) = \frac{\sum_{\theta\in\mathcal{G}_i} Pr(\theta, o^{0:t})}{Pr(o^{0:t})} = \frac{\sum_{\theta\in\mathcal{G}_i} Pr(\theta, o^{0:t})}{\sum_{\theta\in\Theta} Pr(\theta, o^{0:t})}$$

In most cases, it is infeasible to compute $P(\mathcal{G}_i|o^{0:t})$ exactly as it requires enumerating all trajectories $\Theta = St(M_{\text{assess}})$ to generate the complete distribution. Approximations can be computed based on a reduced set of trajectories $\Theta^* \subset St(M_{\text{assess}})$. In [MSR+09] we introduced an approach that enumerates only the k most probable trajectories:

$$P^k(\mathcal{G}_i(k)|o^{0:t}) = \frac{\sum_{\theta \in \mathcal{G}_i(k)} \Pr(\theta, o^{0:t})}{\sum_{\theta \in \Theta(k)} \Pr(\theta, o^{0:t})}.$$

$\Theta(k)$ is the set of k-best trajectories and $\mathcal{G}_i(k) \subseteq \Theta(k)$ the subset of goal-achieving trajectories. With increasing k the approximation eventually becomes the exact probability $Pr(\mathcal{G}_i|o^{0:t})$. Computing meaningful hard bounds p_l and p_u turnes out to be a challenge [MSR+10], and is still an open problem. In [MJW+10] we show how to compute statistical bounds based on *sampling* system trajectories rather than computing the k-best.

4 Model-Based Computing of Plan Success Probabilities

As modeling framework we use probabilistic hierarchical constraint automata (PHCA) [WCG01], a formalism tailored to suit embedded systems development and model-based tracking of complex, uncertain system behavior. In particular it allows modeling of probabilistic failures as well as parallel composition of components. A PHCA is a hierarchical automaton with a set of locations Σ, consisting of primitive and composite locations, a set of variables Π, a set of transitions $\mathcal{T}$ and a set $\mathcal{C}$ consisting of behavior constraints (propositional formulas) for locations and guard constraints for transitions. For each location $X_l \in \Sigma$ a transition function $P_T(X_l)$ defines distributions over subsets of outgoing transitions of X_l. Π consists of dependent, observable and commandable variables. The locations represent hidden modes of the modeled system. The state of a PHCA at time t is a set of marked locations called a marking $m^t \subset \Sigma$. To avoid ambiguity with "state" we will refer to markings rather than PHCA states. A sequence of such markings $\theta = (m^t, m^{t+1}, \ldots, m^{t+n})$ is called a PHCA trajectory. The execution semantics of a PHCA is defined in terms of possible evolutions of such markings. We model a plant as single PHCA, where station behavior and product states are modeled as composite locations. E.g., our example model consists of two composite locations for machining and assembly as well as two composite locations for the maze and the robot.

Most Probable State Sequences As Constraint Optimization

In [MWS05] we casted the problem of finding most probable sequences of PHCA markings within a window of N time steps as computing the best solutions to a combinatorial optimization problem. 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, a correct marking must be ensured regarding, e.g., the hierarchical structure of a PHCA and initial marking.

Given an N-step time window, these execution semantics are encoded as soft constraints [MRS06] for all time points in this time window and the transitions between them. This representation unfolds the PHCA over the N time steps and forms a constraint optimization problem (COP) $\mathcal{R} = (X, D, C)$ (similar to definitions in [MRS06]), where $X = \{X_1, \ldots, X_n\}$ is a set of variables with a corresponding set of finite domains $D = \{D_1, \ldots, D_n\}$, and $C = \{C_1, \ldots, C_q\}$ is a set of constraints (S_i, F_i) with scope $S_i = \{X_{i1}, \ldots, X_{im}\} \subseteq X$ and a constraint function $F_i : D_{i1} \times \ldots \times D_{im} \rightarrow [0,1]$ mapping partial assignments of variables in S_i to a probability value in [0,1]. For all time points $t = t_0 .. t_\mathrm{N}$, hard constraints in C (F_i evaluates to $\{0,1\}$ only) encode hierarchical structure as well as consistency of observations and commands with locations and transitions, while soft constraints in C encode probabilistic choice of initial locations at $t = t_0$ and probabilistic transitions. $\Pi^t \subseteq X$ encodes PHCA variables and auxiliary variables (needed to, e.g., encode hierarchical structure). A set of binary variables $Y = \{X_{L_1}^t, X_{L_2}^t, \ldots\} \subseteq X$, the solution variables of $\mathcal{R}$, encodes location markings. The k-best solutions to $\mathcal{R}$ are assignments to Y which, extended to all variables X, maximize the global probability value in terms of the functions F_i (i.e. the best solution has the largest probability value, the second best the second largest etc.). These assignments correspond to the most probable PHCA system trajectories. The size of $\mathcal{R}$ is essentially N times the PHCA size, since the encoding doesn't flatten the hierarchy (see [MWS05] for details).

To track multiple products, in addition to modeling each product with its own composite location we have to ensure that if at time t_j a product p_id is worked by component c_id, a link between the two is established. We do this by adding a hard constraint, called link constraint, that enforces equality of two helper variables. The schedule $\mathcal{S}$ determines which products and components are to be linked in the current time window, and according link constraints are added before trajectory enumeration (see Algorithm 4-1). For example, if product $p_\mathrm{id} = \mathrm{maze}$ and component $c_\mathrm{id} = \mathrm{machining}$ are to be linked, a constraint over helper variables $X_\mathrm{maze}, X_\mathrm{machining}$ with domain $D_\mathrm{product\,fault} = \{\mathrm{OK}, \mathrm{FAULTY}\}$ is added, enforcing $X_\mathrm{maze} = X_\mathrm{machining}$. X_maze, as part of the product composite location's transition guards, determines the maze's fault transition, while $X_\mathrm{machining}$, occuring in the component's location behavior constraints, explicitly identifies fault locations relevant for the product. Thereby equality of the two variables enforces that the maze becomes flawed if the machining's cutter is broken.

Plan Assessment Based on Time Window Filtering

Let the end time of a manufacturing plan $\mathcal{P}_i$ be $t_{\mathcal{P}_i}$. Typical schedules cannot be assessed with a single time window. Therefore, we shift a relatively small time window along the whole time line of a given schedule, until it covers $t_{\mathcal{P}_i}$. When we finally reach $t_{\mathcal{P}_i}$, we sum over goal-achieving trajectories among the k-best to compute $P(\mathcal{G}_i | o^{0:t})$. Before each shift trajectories of length N are generated, ranging over $t_0 .. t_N$. Then the time window is shifted one step forward and the distribution $Pr(X^{t_1} = s^{t_1}, O^{0:t} = o^{0:t})$ over the new initial states is computed recursively in terms of distributions over previous initial states at t_0 and over trajectories within $t_0..t_N$ (where $O^{0:t}$ is the vector of observation variables from the start of the schedule to t). Note the following notational conventions: we abbreviate $Pr(X^{t_1} = s^{t_1}, O^{0:t} = o^{0:t})$ to $Pr(s^{t_1}, o^{0:t})$. Start and end times of the time window are denoted as $t = t_0$ and $t = t_N$. $t = 0$ is the start time of the schedule and $t = t_c$ is the current time point, i.e. no observations are available after t_c. Unless stated otherwise, t refers to the current time point. The notations $t + x$ and $t - x$ indicate x time points ahead or behind t. For example, states s^0, s^1, s^{t_0}, s^{t_N}, s^t, $s^{t_{\mathcal{P}_i}}$ represent the first and second state relative to the schedule start, the first and last state within the time window, the current state and the state when product i is expected to finish. Observations are indexed analogously. Finally, $S^0, S^{t_0}, \ldots$ denote summation variable vectors for states.

Proposition 1. *Let* $\alpha = \frac{1}{\Pr(o^{t_1:t})}$. *Given observations* $o^{0:t}$ *we can compute the distribution over system states at* t_1 *recursively as*

$$Pr(s^{t_1}, o^{0:t}) = \alpha \sum_{S^{t_0}, S^{t_2}, \ldots, S^{t_N}} Pr(s^{t_1}, \ldots, S^{t_N}, o^{t_1:t} | S^{t_0})\, Pr(S^{t_0}, o^{0:t})$$

Proof. Observe that

$$Pr(s^{t_1}, o^{0:t}) = \textstyle\sum_{S^0, S^1, \ldots, S^{t_0}, S^{t_1}, S^{t_2}, \ldots, S^{t_N}} Pr(S^0, S^1, \ldots, S^{t_0}, S^{t_1}, s^{t_1}, S^{t_2}, \ldots, S^{t_N}, o^{0:t})$$

$$= \textstyle\sum_{S^0, S^1, \ldots, S^{t_0}, S^{t_1}, S^{t_2}, \ldots, S^{t_N}} Pr(s^{t_1}, S^{t_2}, \ldots, S^{t_N} | S^0, \ldots, S^{t_0}, o^{0:t})\, Pr(S^0, \ldots, S^{t_0}, o^{0:t})$$

Then with Markov property for the system states and the observations ($S^{t_1}, \ldots, S^{t_N}$ independent of $S^0, \ldots, S^{t-1}$ and $O^{0:t_0}$) we have

$$= \textstyle\sum_{S^0, S^1, \ldots, S^{t_0}, S^{t_1}, S^{t_2}, \ldots, S^{t_N}} Pr(s^{t_1}, S^{t_2}, \ldots, S^{t_N} | S^{t_0}, o^{t_1:t})\, Pr(S^0, \ldots, S^{t_0}, o^{0:t})$$

$$= \alpha \textstyle\sum_{S^0, S^1, \ldots, S^{t_0}, S^{t_1}, S^{t_2}, \ldots, S^{t_N}} Pr(s^{t_1}, S^{t_2}, \ldots, S^{t_N}, o^{t_1:t} | S^{t_0})\, Pr(S^0, \ldots, S^{t_0}, o^{0:t})$$

$$= \alpha \textstyle\sum_{S^{t_0}, S^{t_2}, \ldots, S^{t_N}} Pr(s^{t_1}, S^{t_2}, \ldots, S^{t_N}, o^{t_1:t} | S^{t_0}) \sum_{S^0, \ldots, S^{t-1}} Pr(S^0, \ldots, S^{t_0}, o^{0:t})$$

$$= \alpha \textstyle\sum_{S^{t_0}, S^{t_2}, \ldots, S^{t_N}} Pr(s^{t_1}, S^{t_2}, \ldots, S^{t_N}, o^{t_1:t} | S^{t_0})\, Pr(S^{t_0}, o^{0:t}) \blacksquare$$

This recursive term represents the exact computation, enumerating all trajectories. As this is typically infeasible, we approximate based on the k-best trajectories instead. Given these trajectories, we compute approximations for α and the distribution $Pr(X^{t_0}, o^{0:t})$ (t_0 relative to the new window). For every shift of the time window we add $Pr(X^{t_0}, o^{0:t})$ as constraint C_{SD} to the original constraint problem. A modified external solver then computes the k most probable trajectories along with their joint probabilities $Pr(s^{t_0:t_N}, o^{0:t}) = Pr(s^{t_1}, \ldots, s^{t_N}, o^{t_1:t} | s^{t_0})\, Pr(s^{t_0}, o^{0:t})$ for the next shift.

Algorithm 4-1 implements the Time Window Filtering (TWF). Consider the following example (illustrated in Figure 4-1), where we focus on three system states from M_{assess}: 1) a state were both maze and robot arm are "ok", 2) a state were the cutter broke and thus led to a flawed robot, but the maze is "ok" and 3) like 2, with the difference that here additionally vibrations in the assembly occured (not only caused by the blunt/broken cutter). These states are encoded as location variable assignments, e.g.

$X^t_{\text{maze.ok}} = M$. Note that for clarity the location domain $\{\text{MARKED}, \text{UNMARKED}\}$ is abbreviated to $\{\text{M}, \text{U}\}$ and the super script ^{t} is omitted in Figure 4-1.

```
1:  function TWF($W^N_{t_0}$, $o^{t_0:t}$, $\mathcal{S}$, $k$)
2:      $\Theta^* \leftarrow$ ESTIMATE($W^N_{t_0}$, $o^{t_0:t}$, $\mathcal{S}$, $k$)
3:      $W^N_{t_0+1} \leftarrow$ SHIFTTIMEWINDOW($W^N_{t_0}$, $\Theta^*$)
        return ($\Theta^*$, $W^N_{t_0+1}$)
4:  end function
5:  function ESTIMATE($W^N_{t_0}$, $o^{t_0:t}$, $\mathcal{S}$, $k$)
6:      $(\mathcal{R}, N, t_0) \leftarrow W^N_{t_0}$
7:      $C' \leftarrow \mathcal{R}.C \cup$ ASUNARYCONSTRAINTS($o^{t_0:t}$)
8:      $C' \leftarrow C' \cup$ PRODUCTCOMPONENTLINKS($\mathcal{S}$, $\mathcal{R}$)
9:      $\mathcal{R}' \leftarrow \langle \mathcal{R}.X, \mathcal{R}.D, C' \rangle$
10:     $\Theta^* \leftarrow$ SOLVE($\mathcal{R}'$, $k$)
        return $\Theta^*$
11: end function
12: function SHIFTTIMEWINDOW($W^N_{t_0}$,$\Theta^*$)
13:     $(\mathcal{R}, N, t_0) \leftarrow W^N_{t_0}$
14:     if $t_0 = 0$ then
15:         Remove constraints for initial location marking from $\mathcal{R}$
16:     end if
17:     $S_{\text{SD}} \leftarrow \{X^{t_1}_L \mid L$ is location marking variable$\}$
18:     $F_{\text{SD}} \leftarrow \{s^{t_1} \mapsto$ SUMTRAJECTORIESFORSTATE($s^{t_1}$, $N$) $\mid s^{t_1} \in$ TIMESTEPSTATES($\Theta^*$, 1) $\}$
19:     $C_{\text{SD}} \leftarrow (S_{\text{SD}}, F_{\text{SD}})$
20:     $\mathcal{R}.C \leftarrow \mathcal{R}.C \cup C_{\text{SD}}$                ▷ overrides previously added $C_{\text{SD}}$
        return $(\mathcal{R}, N, t_0 + 1)$
21: end function
22: function SUMTRAJECTORIESFORSTATE($s^{t_1}$, $N$)
        return $\alpha \sum_{S^{t_0}, S^{t_2}, \ldots, S^{t_N}} Pr(s^{t_1}, S^{t_2}, \ldots, S^{t_N}, o^{t_1:t} \mid S^{t_0})\, Pr(S^{t_0} \mid o^{0:t})$
23: end function
```

Algorithm 4-1: The Time Window Filtering algorithm. Function AsUnaryConstraints maps observations to unary constraints over observation variables, function TimeStepStates maps a set of trajectories to the set of states for the given time point. Function ProductComponentLinks generates product-component link constraints.

A time window of $N = 2$ is used and $k = 6$ trajectories are enumerated. We use the notation $W^N_t = (\mathcal{R}, N, t)$ for a time window of length N starting at time point t, with behaviors encoded in COP $\mathcal{R}$. Trajectories are numbered 1 (most probable) through 6 (least probable). At first, the time window W^2_t does not cover the product finishing time and thus success probabilities can't be computed. The time window is shifted (W^2_{t+1}), and in doing so probabilities of trajectories leading to the same initial state in the next window are summed up. They are normalized using the approximation $\alpha^* = \frac{1}{\sum_{\theta \in \Theta^*} Pr(\theta, o^{0:t})}$. This is done for all initial states, and thereby the approximate initial state distribution for the next time window is created. Now the finishing time is covered. Again, $k = 6$ trajectories are enumerated within the new time window. Only one of these trajectories is goal-achieving for the robot arm, while all of them lead to a properly finished maze. Summing over the respective goal-achieving and normalizing over all enumerated trajectories yields approximate success probabilities for the maze and the robot arm. Note that the approach readily deals with unknown future observations, which are simply handled as additional unkown variables.

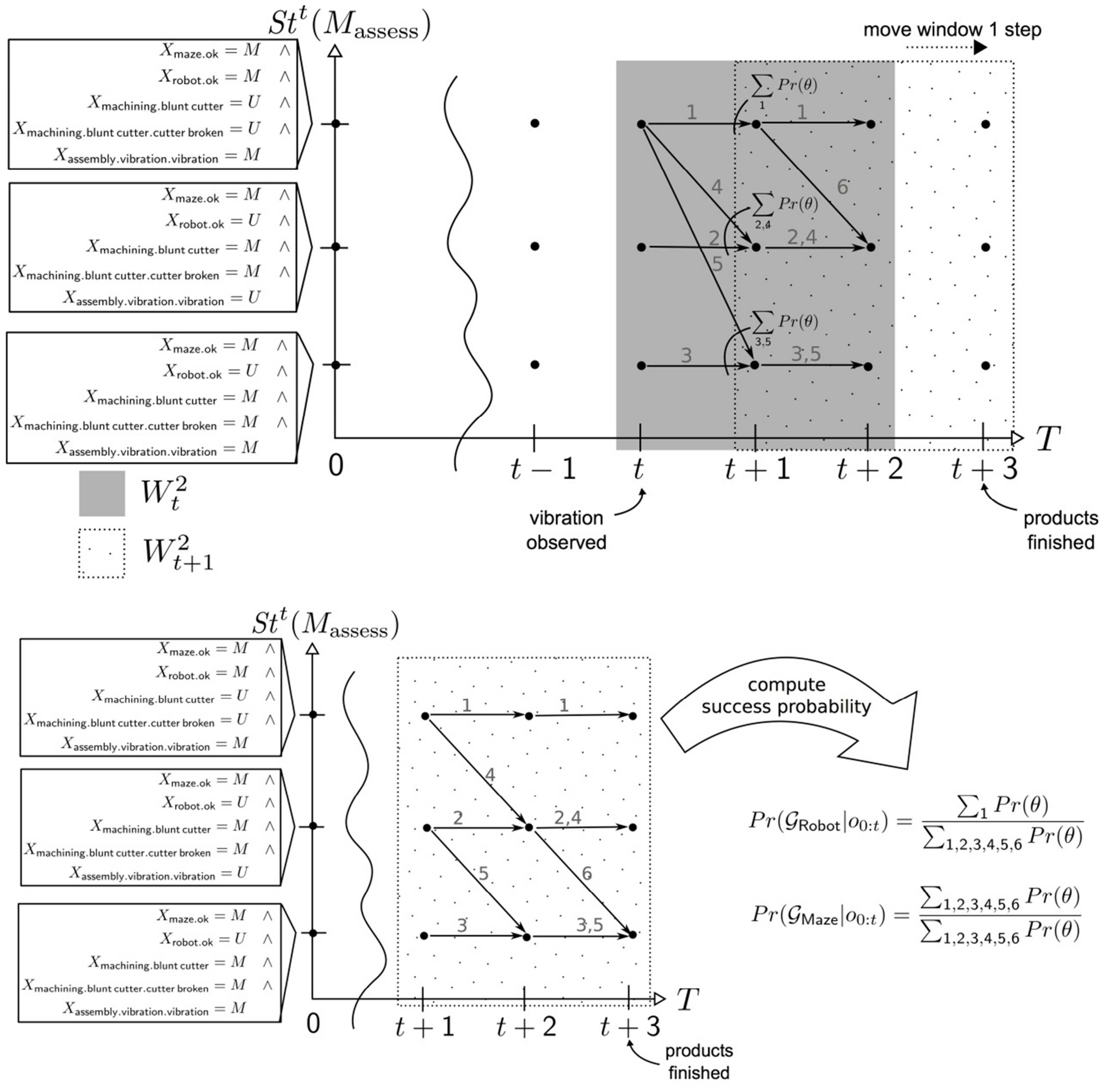
$St^t(M_{\text{assess}})$
$X_{\text{maze.ok}} = M \wedge$
$X_{\text{robot.ok}} = M \wedge$
$X_{\text{machining.blunt cutter}} = U \wedge$
$X_{\text{machining.blunt cutter.cutter broken}} = U \wedge$
$X_{\text{assembly.vibration.vibration}} = M$
$X_{\text{maze.ok}} = M \wedge$
$X_{\text{robot.ok}} = U \wedge$
$X_{\text{machining.blunt cutter}} = M \wedge$
$X_{\text{machining.blunt cutter.cutter broken}} = M \wedge$
$X_{\text{assembly.vibration.vibration}} = U$
$X_{\text{maze.ok}} = M \wedge$
$X_{\text{robot.ok}} = U \wedge$
$X_{\text{machining.blunt cutter}} = M \wedge$
$X_{\text{machining.blunt cutter.cutter broken}} = M \wedge$
$X_{\text{assembly.vibration.vibration}} = M$
move window 1 step
$\sum_1 Pr(\theta)$
$\sum_{2,4} Pr(\theta)$
$\sum_{3,5} Pr(\theta)$
1
2
3
4
5
6
2,4
3,5
0
$t-1$
t
$t+1$
$t+2$
$t+3$
T
vibration observed
products finished
W_t^2
W_{t+1}^2
compute success probability
$Pr(\mathcal{G}_{\text{Robot}}|o_{0:t}) = \frac{\sum_1 Pr(\theta)}{\sum_{1,2,3,4,5,6} Pr(\theta)}$
$Pr(\mathcal{G}_{\text{Maze}}|o_{0:t}) = \frac{\sum_{1,2,3,4,5,6} Pr(\theta)}{\sum_{1,2,3,4,5,6} Pr(\theta)}$

Figure 4-1: Above: Moving time window W_t^2 one step, generating time window W_{t+1}^2. Below: Computing success probabilities after $k = 6$ trajectories have been enumerated in the new time window W_{t+1}^2.

5 Results and Discussion

We implemented the constraint-based k-best enumeration of most probable trajectories as described in section 4 and ran experiments on an intel core 2 duo machine with 2.53 Ghz and 4GB of RAM, using a simple factory PHCA model (see Figure 5-1) for our scenario. Table 5-1 presents results for our scenario with $N_S = 9$, which can be solved without TWF with an $N = 9$ time window. It shows that plan assessment can indeed identify the robot arm to be jeopardized, given the observation of a vibration, while clarifying that the maze's success is not influenced. Intuitively, only products which further use suspicious components (machining) are likely to be flawed. We could solve instances of our scenario with schedule length' up to 14 time steps, which we generated

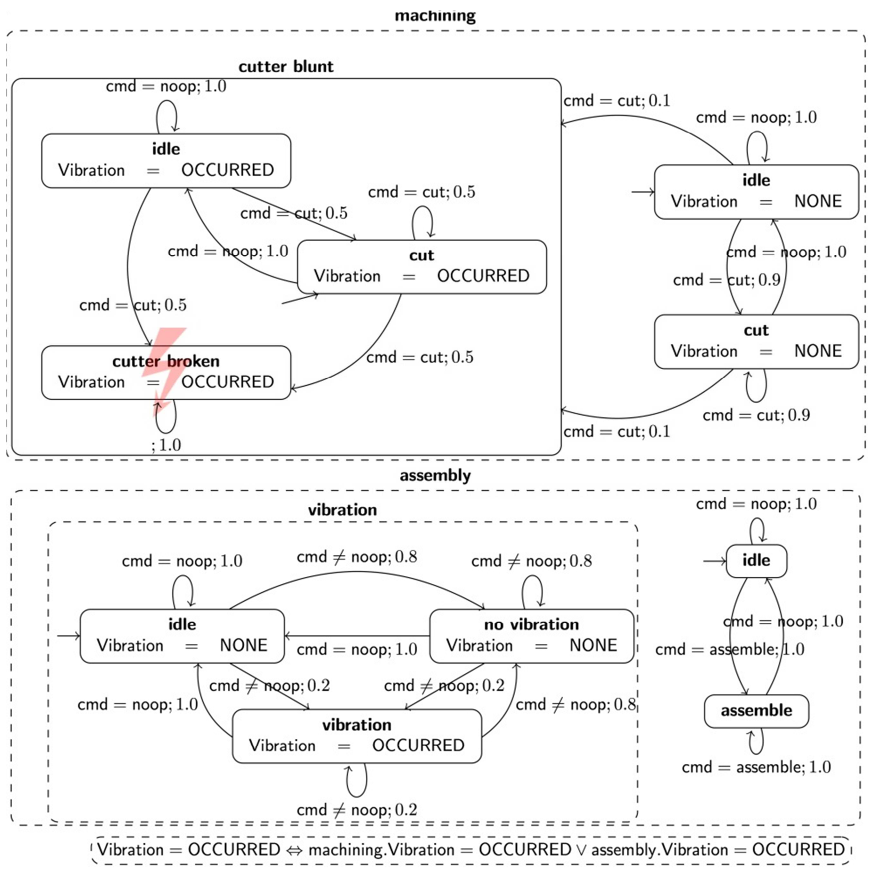

Figure 5-1: PHCA composite locations for maching and assembly stations.

Table 5-1: Success probabilities for the robot arm and the toy maze over different k with $N_S = 9$. The accoring COP has ≈ 1200 variables and ≈ 1250 constraints. First col.: exact solution. Last row: COP solving time.

	k					
	exact	10	5	4	3	2
$Pr(\mathcal{G}_{\text{Maze}} \mid \ldots)$	1.0	1.0	1.0	1.0	1.0	1.0
$Pr(\mathcal{G}_{\text{Robot}} \mid \ldots)$	0.53	0.65	0.74	0.81	0.78	0.74
time	3.0	2.0	1.8	1.8	1.8	1.8

by simply multiplying scheduled actions. Tweaking solver parameters might gain another one or two steps, but eventually the exponential dependency on schedule length exhausts resources (memory in this case). In a preliminary experiment using our novel TWF method we could solve a problem with $N_S = 29$, $N = 5$ in 240 seconds. We solved the problem by moving the time window along the whole schedule length, doing 24 shift operations, where each interleaved estimation step took 4-10 seconds. Table 5-2 shows how much the success probability for the robot deviates from the exact solution when varying k and N. Clearly, if k is big enough to enumerate all non-zero trajectories (> 80), TWF yields the exact solution independently of N. When choosing only one trajectory, increasing N seems to reduce the error.

Table 5-2: Comparing the success probability error (above) and runtime (in seconds, below) for the robot product for different number of trajectories k and time window sizes N.

k	$N = 1$	$N = 2$	$N = 3$	$N = 4$	$N = 5$	$N = 6$	$N = 7$	$N = 8$	$N = 9$
2	0.46	0.46	0.46	0.46	0.26	0.21	0.21	0.21	0.21
	39.77	36.14	34.44	35.84	50.82	57.56	57.14	47.50	27.65
3	0.42	0.46	0.46	0.46	0.46	0.25	0.25	0.25	0.25
	39.68	36.08	34.39	36.54	50.85	58.05	57.58	47.28	27.97
4	0.26	0.30	0.36	0.36	0.36	0.28	0.28	0.28	0.28
	39.56	35.96	34.35	36.02	50.81	58.62	57.34	48.88	27.94
5	0.25	0.31	0.34	0.32	0.26	0.21	0.21	0.21	0.21
	39.54	36.47	34.55	36.68	51.25	59.26	58.61	48.15	29.30
10	0.06	0.13	0.09	0.09	0.11	0.12	0.12	0.12	0.12
	40.92	36.09	34.89	36.49	53.37	59.68	59.11	48.45	28.34
25	0.00	0.02	0.03	0.06	0.06	0.05	0.05	0.05	0.05
	41.03	36.66	35.49	37.73	54.97	61.95	61.73	48.50	28.62
50	0.00	0.00	0.00	0.01	0.01	0.01	0.01	0.01	0.01
	40.32	37.19	37.69	39.19	54.46	60.72	60.04	50.79	29.34
100	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	40.89	39.26	37.53	39.23	54.87	61.56	60.76	50.56	29.98

The measured runtimes indicate feasibility within a day-to-day computation cycle. TWF allows to address bigger schedules and depends only linearly on the number of shifts. However, runtime increases considerably, especially when comparing the runtime for solving a problem with $N_S = 9$ (4s at worst) with the TWF performance. Partly this is a problem of our prototype implementation, which has some overhead. However, there are two other problems: First, computing the distribution over initial states for the next time window, even if only approximated, yields a big constraint over many location variables. It renders the trajectory enumeration step considerably harder and has the effect that above ≈ 30 locations in the assessment model our solver fails to even load the COP. Second, because the COP is changed with every shift in TWF, we did a preprocessing step on-line that usually can be taken off-line. The first problem could be alleviated with smaller time windows, and other solvers might be a remedy. However, we are not aware of any solvers that support k-best solution generation for constraint optimization. For the second problem it might suffice to modify the preprocessing step to allow off-line computing again. Further improvements could be to move the time window multiple steps at once, decreasing the number of estimations, and to only shift the time window to keep up with the schedule. Only if an assessment is requested the window is moved towards $t_{\mathcal{P}_i}$. All in all, we think that TWF makes a step towards a practical solution of the plan assessment problem, but a number of challanges remain.

6 Conclusion

Within an autonomous factory setting, we presented a novel method to solve the plan assessment problem, which asks whether a product presently being processed will be successfully produced. The method computes the product's success probability by tracking the k most probable factory behaviors and the product's states within a small, fixed-length time window. To cover large schedules the time window is moved along the time line, computing a distribution over initial states of the new window from enumerated trajectories in the current window. Our preliminary results indicate that our novel algorithm can indeed solve problems that a method that enumerates trajectories over the complete schedule length cannot solve.

References

[ASB07] Althoff, M.; Stursberg, O.; Buss, M.: Online Verification of Cognitive Car Decisions. *Proc. IV-2007*, 2007.

[BBW07] Beetz, M.; Buss, M.; Wollherr, D.: Cognitive Technical Systems --- What is the Role of Artificial Intelligence? *Proc. KI-2007*, pages 19--42, 2007.

[FVL+06] Ferrarini, L.; Veber, C.; Luder, A.; Peschke, J.; Kalogeras, A.; Gialelis, J.; Rode, J.; Wunsch, D.; Chapurlat, V.; e Informazione, D.E.: Control Architecture for Reconfigurable Manufacturing Systems: the PABADIS'PROMISE Approach. *Proc. ETFA-2006*, pages 545--552, 2006.

[KN00] Kurien, J.; Nayak, P. P.: Back to the Future for Consistency-Based Trajectory Tracking. *Proc. AAAI-2000*, pages 370--377, 2000. AAAI Press.

[MSW04] Mahtab, T.; Sullivan, G.; Williams, B. C.: Automated Verification of Model-based Programs Under Uncertainty. *Proc. ISDA-2004*, 2004.

[MJW+10] Maier, P.; Jain, D.; Waldherr, S.; Sachenbacher, M.: Plan Assessment for Autonomous Manufacturing as Bayesian Inference. *Proc. KI 2010*, 2010.

[MSR+10] Maier, P.; Sachenbacher, M.; Rühr, T.; Kuhn, L.: Automated plan assessment in cognitive manufacturing. *Adv. Eng. Informat.*, 2010. To appear.

[MSR+09] Maier, P.; Sachenbacher, M.; Rühr, T.; Kuhn, L.: Constraint-Based Integration of Plan Tracking and Prognosis for Autonomous Production. *Proc. KI-2009* in LNCS, Paderborn, Germany, 2009. Springer.

[MWS05] Mikaelian, T.; Williams, C. B.; Sachenbacher, M.: Model-based Monitoring and Diagnosis of Systems with Software-Extended Behavior. *Proc. AAAI-2005*, Pittsburgh, USA, 2005. AAAI Press.

[RKN+04] 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.

[WCG01] Williams, B. C.; Chung, S.; Gupta, V.: Mode Estimation of Model-based Programs: Monitoring Systems with Complex Behavior. *Proc IJCAI-2001*, pages 579--590, 2001.

[MRS06] Meseguer, P.; Rossi, F.; Schiex, T.: Handbook of constraint programming, chapter 9: Soft Constraints, pages 281–328. Foundations of Artificial Intelligence. Elsevier, 2006.

[Gau05] Jürgen Gausemeier. From mechatronics to self-optimizing concepts and structures in mechanical engineering: new approaches to design methodology. Int. J. Computer Integrated Manufacturing, 18(7):550–560, 2005.