# Back to the Future for Consistency-based Trajectory Tracking

**James Kurien**
NASA Ames Research Center
MS 269-3, Moffett Field, CA 94035
jkurien@arc.nasa.gov

**P. Pandurang Nayak**
PurpleYogi.com and RIACS,
201 Ravendale Drive, Mountain View, CA 94043.
nayak@purpleyogi.com

## Abstract

Given a model of a physical process and a sequence of commands and observations received over time, the task of an autonomous controller is to determine the likely states of the process and the actions required to move the process to a desired configuration. We introduce a representation and algorithms for incrementally generating approximate belief states for a restricted but relevant class of partially observable Markov decision processes with very large state spaces. The algorithm incrementally generates, rather than revises, an approximate belief state at any point by abstracting and summarizing segments of the likely trajectories of the process. This enables applications to efficiently maintain a partial belief state when it remains consistent with observations and revisit past assumptions about the process's evolution when the belief state is ruled out. The system presented has been implemented and results on examples from the domain of spacecraft control are presented.

## Introduction

Given a model of a physical system and a sequence of commands and observations received over time, the task of an autonomous controller is to determine the likely states of the system and the actions required to move the system to a desired configuration. Focusing on the state identification question, a *belief state* is a probability distribution over the possible states of a system. If the system has the Markov property, then the influence of a new command and observation upon the belief state can be integrated via Bayes' rule. The updated belief state is a sufficient statistic, capturing within a single distribution all knowledge about the current state of the system contained within a history of commands and observations. The controller then makes use of the updated belief state in selecting an action.

**Example 1** Consider the spacecraft propulsion subsystem of Figure 1. The helium tank pressurizes the two propellant tanks. When a propellant path to either engine is open, the pressurized tanks force fuel and oxidizer into the engine, producing thrust. Not shown are valve drivers that control the latch valves and a set of flow, pressure and acceleration sensors that provide partial observability. A model of a system specifies the modes of each component ( *e.g.,* a valve may be open, closed, stuck closed, and so on), behavior in each mode ( *e.g.,* a closed valve prevents flow), mode transitions ( *e.g.,* valves usually open when commanded, but stick
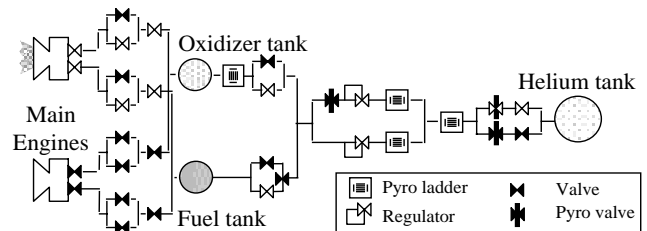
Figure 1: Propulsion system schematic.

closed with probability $p$) and connections between components ( *e.g.,* fuel flow into an engine is equal to the flow out of the attached fuel valve).

Consider the problem of determining the likelihood of the possible states of this subsystem. Unfortunately, computing a belief state in general requires enumeration of the state space. The propulsion subsystem has 38 components with an average of 3 states each. More complete spacecraft models capture 150 or more components averaging 4 states, yielding a state space of $2^{300}$ or more and making complete enumeration implausible. One alternative is to track an approximation whose computation does not require enumeration of the state space, ideally enumerating only the most likely portion of the belief state at each point in time. *Livingstone* (Williams & Nayak 1996) tracks $n$ approximately most likely states of the system by transitioning a small number of tracked states by the transitions that are most likely, given only the current observations. This approximation is extremely efficient and well suited to the problem of tracking the internal state of a machine, where the likelihood of the nominal or expected transition dominates, and immediate observations often rule out the nominal trajectory when a failure occurs. The task then becomes one of diagnosing the most likely system transition, chosen from combinations of component transitions, that would be consistent with the unexpected observations. Using this technique, *Livingstone* is able to perform approximate state identification and reconfiguration of systems with hundreds of state variables. It has been applied to the control of a number of systems within NASA and is an integral part of the Remote Agent architecture demonstrated in-flight on the Deep Space 1 spacecraft in 1999 (Muscettola *et al.* 1998; Bernard *et al.* 1998). Unfortunately, the true trajectory may not be among the most likely given only the current observations. Consider the following example.
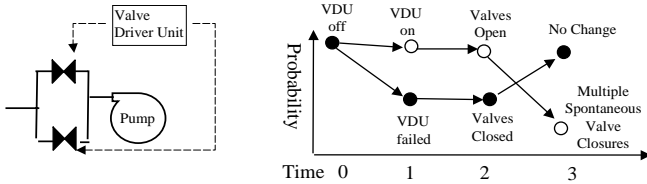
Figure 2: Evolution of a Valve Driver Unit and Valves

**Example 2** Figure 2 illustrates a small system and two possible trajectories. The pump pressurizes the system and the valves, if open, allow a fluid flow. The valve driver unit (VDU) commands the two valves in parallel via the data bus represented by dashed lines. The graph to the right represents the probability of two possible trajectories. The filled circles represent the true state of the system. At time 0 the VDU is off, the valves are closed and the pump is off. At time 0 the VDU is commanded on. For the sake of illustration, consider an approximate belief state of size 1. The state wherein the VDU is on is placed into the belief state. The true state wherein the VDU is failed is discarded. At time 1, the VDU is commanded to open its valves. Since the only state in the belief state assumes the VDU is on, the single state in the updated belief state has the VDU on and all valves open. In the true, untracked state the valves are closed, as they never received a command. At time 2, the pump is turned on. Pressure is observed at the outlet of the pump, yet no flow is observed downstream of the valves. Failure of the pump alone has zero probability, given the observations. Failure of the VDU in the current time step has no effect on the valves. Thus, the most likely next state consistent with the observations requires that all valves spontaneously and independently shut. Regardless of the number of valves and the unlikeliness of spontaneous closure, this transition must be taken if it exists. If it does not exist, the belief state approximation becomes empty.

In general, as the true state evolves, the tracked subset of states may need to undergo arbitrarily unlikely transitions in order to remain consistent with the observations. While only one trajectory is tracked in this example, for any fraction of the trajectories that are tracked, an example can be constructed wherein the actual state of the system falls outside the tracked fraction and the error in the approximation may become arbitrarily large. We propose an alternative to committing to a subset of the current belief state or maintaining an approximation of the entire belief state. We propose to maintain the information necessary to begin incrementally generating the current belief state in best-first order at any point in time. Since we do not update the entire belief state, we do not have a sufficient statistic, so a history must be maintained. We introduce a variable to represent every state variable, command and observation at every point in time and an algorithm for incrementally generating the exact belief state at any point. Duplicating the entire set of variables at each point in the history seems impractical except for short duration tasks. We apply two approximations motivated by our experience modeling physical systems for

*Livingstone.* The first duplicates only a small number of carefully selected variables at each time point. This approximation is conservative in that does not eliminate any feasible trajectories but may admit certain infeasible trajectories. These may be eliminated by future observations. The second limits the length of the history that is maintained by absorbing older variables into a single variable that grossly approximates them. This allows an approximate belief state to be generated at any point in time from a constant number of variables. The variables represent an exact model of system evolution over the recent past, an approximate model over the intermediate past, and a gross summarization over the more distant past. This allows assignment of the most likely past transitions to be revisited as new observations become available. The fewest variables, and thus the least flexibility, are allocated to segments of the system trajectory that have remained consistent with the system's observed evolution for the longest time.

In the following sections of the paper, we start by giving the complete history representation followed by a simple, exact, and intractable algorithm for enumerating the belief state. We introduce optimizations and approximations in order to gain tractability while maintaining the ability to revise assessments of past system evolution. We introduce a software system *L2* (for *Livingstone2*) that embodies these ideas. Finally we present the results of running *L2* on scenarios developed while applying *Livingstone* within NASA.

## Transition Systems

We wish to represent the possible histories of a system composed of non-deterministic, concurrent automata given the commands issued to the automata and their output. We create a structure that allows incremental, best-first enumeration of all possible trajectories by extending the formalism of *Livingstone*. In order to compactly represent the trajectories, we add a set of transition variables that represent the non-deterministic transitions each automaton may make at each time step. Each assignment to a transition variable has a likelihood representing the prior probability of the corresponding non-deterministic transition occurring. One trajectory of the system is thus an assignment to each transition variable, and given the appropriate independence assumptions, the set of trajectories can be incrementally enumerated in order of likelihood. In order to capture the feasible behaviors of the automata, we introduce a set of formulae $\mathcal{M}_\Sigma$ describing the input/output mapping of the automata in each state, and a set of formulae $\mathcal{M}_\mathcal{T}$ for describing the feasible transitions of the automata.

**Definition 1** A *transition system* $\mathcal{S}$ is a tuple $\langle \Pi, \mathcal{T}, \mathcal{D}, \mathcal{C}, \ \mathcal{M}_\Sigma, \mathcal{M}_\mathcal{T} \rangle$, where

- $\Pi$ is a set of *state variables* representing the state of each automaton. Let $n$ denote the number of automata and $m$ denote the number of discrete, synchronous time steps over which the state is to be tracked. $\Pi$ then contains $m \times n$ variables. $\Pi_t$ will denote the set of state variables representing the state of the system at time step $t$. Each state variable $y$ ranges over a finite domain denoted $\delta(y)$.

The temporal variable representing the occurrence of variable $y$ at time step $t$ is denoted $y_t$.

- $\mathcal{T}$ is a set of *transition variables*. The transition variable that represents the transition of state variable $y$ from time $t$ to $t+1$ is denoted $\tau_{y,t}$. Each value in the domain of $\tau_{y,t}$ is assigned a probability.
- $\mathcal{D}$ is a finite set of *dependent variables*.
- $\mathcal{C}$ is a finite set of *command variables*.
- State $s_t$ is an assignment to $\Pi_t \cup \mathcal{T}_t \cup \mathcal{D}_t \cup \mathcal{C}_t$
- $\mathcal{M}_\Sigma$ is a propositional formula over $\Pi_t$ and $\mathcal{D}_t$ that specifies the feasible subset of the state space. A state is feasible if it makes an assignment to $\Pi_t \cup \mathcal{D}_t$ that is consistent with $\mathcal{M}_\Sigma$.
- $\mathcal{M}_\mathcal{T}$ is a propositional formula over $\Pi_t$, $\mathcal{D}_t$, $\mathcal{C}_t$, $\mathcal{T}_t$ and $\Pi_{t+1}$ that specifies the feasible sequences of states. $\mathcal{M}_\mathcal{T}$ is a conjunction of transition formulae modeling possible evolutions of $y_t$ to $y_{t+1}$ of the form

$$\phi_t \wedge (\tau_{y,t} = \tau^*) \Rightarrow y_{t+1} = y^*$$

where $\phi_t$ is a propositional formula over $\Pi_t \cup \mathcal{D}_t \cup \mathcal{C}_t$, and $\tau^*$, representing a choice among the non-deterministic transitions of $y$, is in $\delta(\tau_{y,t})$. The sequence $s_i, s_{i+1}$ is feasible if the assignment made by $s_i \cup s_{i+1}$ is consistent with $\mathcal{M}_\mathcal{T}$.

**Example 3** We introduce a transition system to model a VDU and two valves. The variables corresponding to the VDU consist of a state variable $vdu$ representing the mode (*on*, *off*, or *failed*), the transition variable $\tau_{vdu}$, a command variable $cmdin$ representing commands to the VDU or its associated valves (*on*, *off*, *open*, *close*, *none*), and a dependent variable $cmdout$ representing the command the VDU passes on to its valves (*open*, *close*, or *none*). The feasible states of the VDU are specified by the formulae

$$
\begin{aligned}
vdu = on \quad \Rightarrow \quad & (cmdin = open \Rightarrow cmdout = open) \\
& \wedge (cmdin = close \Rightarrow cmdout = close) \\
& \wedge ((cmdin \neq open \wedge cmdin \neq close) \\
& \quad \Rightarrow cmdout = none) \\
vdu = off \quad \Rightarrow \quad & cmdout = none \\
vdu = failed \quad \Rightarrow \quad & cmdout = none
\end{aligned}
$$

together with formulae like $(vdu \neq on) \vee (vdu \neq off) \vee (vdu \neq failed), \ldots$ that assert that variables have unique values. The time step subscript is omitted, indicating that all clauses refer to variables within the same time step. The valves $v1$ and $v2$ each have a state variable of domain (*open*, *closed*, or *stuck*), a transition variable $\tau_{vi}$ and a dependent variable $flow_{vi}$ of domain (*zero*, *nonzero*). The feasible states of the $v1$ are specified by the formula below. The feasible states of $v2$ are specified similarly.

$$
\begin{aligned}
v1 = open \quad &\Rightarrow \quad flow_{v1} = nonzero \\
v1 = closed \quad &\Rightarrow \quad flow_{v1} = zero \\
v1 = stuck \quad &\Rightarrow \quad flow_{v1} = zero
\end{aligned}
$$

$\mathcal{M}_\mathcal{T}$ for $\tau_{vdu}$ is as follows.

$$
\begin{aligned}
\tau_{vdu,t} = nominal \Rightarrow & \\
vdu_t = off \wedge cmdin_t = on \quad &\Rightarrow \quad vdu_{t+1} = on \\
vdu_t = off \wedge cmdin \neq on \quad &\Rightarrow \quad vdu_{t+1} = off \\
vdu_t = on \wedge cmdin_t = off \quad &\Rightarrow \quad vdu_{t+1} = off \\
vdu_t = on \wedge cmdin_t \neq off \quad &\Rightarrow \quad vdu_{t+1} = on \\
vdu_t = failed \quad &\Rightarrow \quad vdu_{t+1} = failed \\
\tau_{vdu,t} = fail \Rightarrow vdu_{t+1} = failed &
\end{aligned}
$$

$\mathcal{M}_\mathcal{T}$ for $\tau_{v1}$ is shown below. $\tau_{v2}$ is as $\tau_{v1}$.

$$
\begin{aligned}
\tau_{v1,t} = nominal \Rightarrow & \\
v1_t = closed \wedge cmdout_t = open \quad &\Rightarrow \quad v1_{t+1} = open \\
v1_t = closed \wedge cmdout_t \neq open \quad &\Rightarrow \quad v1_{t+1} = closed \\
v1_t = open \wedge cmdout_t = closed \quad &\Rightarrow \quad v1_{t+1} = closed \\
v1_t = open \wedge cmdout_t \neq close \quad &\Rightarrow \quad v1_{t+1} = open \\
v1_t = stuck \quad &\Rightarrow \quad v1_{t+1} = stuck \\
\tau_{v1,t} = stick \Rightarrow v1_{,t+1} = stuck &
\end{aligned}
$$

## Infinitesimals

In order to complete the transition system model shown in Example 3, we require the probability of each $\tau_{y,t}$ assignment, representing the prior probability of each possible component transition. Experience with *Livingstone* suggests that an order of magnitude probability scale is sufficient for two reasons. First, the internal behavior of a machine is usually far less stochastic than its interaction with its environment. There is an expected or nominal behavior that a component will exhibit for a given state and input. Failures are one or more orders of magnitude less likely. Second, precise estimates for these priors are often either inaccessible or unknown. In the case of spacecraft, the components may be unique or they may be destined for a new operating environment. However, the relative plausibility of each failure mode during operation can be elicited quite easily. In this work, we formalize and capitalize on these characteristics of the priors by making use of infinitesimals (Goldszmidt & Pearl 1992) to model the relative likelihoods of failures.

An infinitesimal probability is represented by an infinitesimally small constant raised to an exponent referred to as the *rank*. The rank can be considered the degree of unbelievability. Intuitively, one would not consider a rank 2 infinitesimal believable unless all rank 0 and rank 1 possibilities had been eliminated. Composition of infinitesimals has many desirable properties. If $A$ and $B$ are independent events, then

$$Rank(AB) = Rank(A) + Rank(B)$$
$$Rank(A \vee B) = min(Rank(A), Rank(B))$$

Thus an outcome that can occur through multiple independent events has rank $i$ if one event has rank $i$ and the remaining events, even if arbitrarily many, have ranks of $i$ or more. This property is key. It allows us to consider only the most likely trajectories leading to a state: if a sequence of events of rank $i$ ends in state $s_j$, then an arbitrary number of higher rank (i.e. less likely) trajectories leading to $s_j$ will not change its rank. Similarly, if state $s_j$ is reached by a trajectory of rank $i$, and no trajectory of rank $i$ or less reaches $s_k$, then $s_j$ is more likely than $s_k$. We need not consider the possibility that a vast number of unlikely trajectories lead to $s_k$ and together increase its likelihood above that of $s_j$. We frame our algorithms in terms most likely trajectories, knowing the direct correspondence to most likely states given the infinitesimal interpretation of the priors.

## Trajectory Identification

**Definition 2** A *trajectory* for $\mathcal{S}$ is a sequence of states $s_0, s_1, \ldots s_m$ such that for all $t$, $0 < t < m$, $s_t$ is consistent with $\mathcal{M}_\Sigma$ and for all $t$, $0 < t < (m-1)$, $s_t \cup s_{t+1}$ is consistent with $\mathcal{M}_\mathcal{T}$.
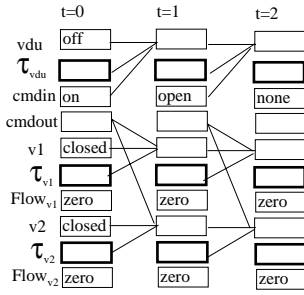
Figure 3: Evolution of the VDU/valve system

Figure 4: Two evolutions of the system

Consider the problem of determining the state of a physical process modeled by a transition system $\mathcal{S}$ at each point in a trajectory $s_0 \ldots s_m$. The subset of the dependent variables $\mathcal{D}$ whose assignment corresponds to a measurement from the process will be referred to as the observations, $\mathcal{O}$. We are given an assignment for the initial state, $\Pi_0$. In addition we are given assignments to commands $\mathcal{C}_t$ and observations $\mathcal{O}_t$ for all $0 < t < m$. The task is to choose assignments to $\tau_{y,t}$ for all $y$ and $t$ so as to ensure consistency with $\mathcal{M}_\Sigma$ and $\mathcal{M}_\mathcal{T}$ and maximize the likelihood of the trajectory. That is to say, given a starting state, a set of commands and a set of observations, we must find the most likely sequence of transitions such that each state is consistent with the state model $\mathcal{M}_\Sigma$ and the transitions are consistent with the transition model $\mathcal{M}_\mathcal{T}$. We define trajectory likelihood to be $\sum_{t=0}^{m} \sum_{y=1}^{n} Rank(\tau_{y,t})$. This definition makes the assumption that the likelihood of assignments to $\tau_{y,t}$ are independent of $\tau_{x,t}$. This is a common assumption and has been an adequate approximation in practice. Note that this assumption does not effect the handling of single failures that manifest themselves at multiple points throughout the system ( *e.g.,* a power failure causing all lights to go out).

## A Simple Tracking Algorithm

The transition-system formulation suggests an intuitive procedure to begin enumerating the belief state at any point. The transition system is initialized with $\mathcal{M}_\Sigma$ and a copy of all variables, representing the initial state. At time step $t$, we introduce a copy of $\mathcal{M}_\Sigma$ and a copy of all variables, representing the next state of the system, as well as a copy of $\mathcal{M}_\mathcal{T}$ representing the constraints between the current and next states. We assign $\mathcal{C}_t$ and $\mathcal{O}_{t+1}$ according to how the system was commanded and the observations that resulted.

**Example 4** Figure 3 illustrates a trajectory-tracking problem of length three for the model of Example 2. Each box represents an assignment. The command is $cmdin$ and the observations are $flow_{v1}$ and $flow_{v2}$. These variables are assigned by the problem, as is the start state. The highlighted $\tau_{y,t}$ assignments must be chosen. The remaining variables will be constrained based upon these assignments. The arcs represent constraints from $\mathcal{M}_\mathcal{T}$. Constraints from $\mathcal{M}_\Sigma$ are not shown. For all $\tau_{y,t}$ we will assume $Rank(\tau_{y,t} = nominal) = 0$ and $Rank(\tau_{y,t} \neq nominal) = 1$.
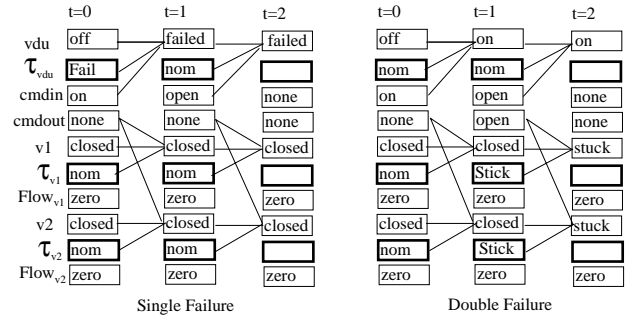
Trajectories may be enumerated in order by enumerating assignments to all $\tau_{y,t}$ in order of the sum of the ranks, then testing for consistency with $\mathcal{M}_\mathcal{T}$ and $\mathcal{M}_\Sigma$. Conflict-directed, best-first search, or *CBFS* (Dressler & Struss 1992; de Kleer & Williams 1989; Williams & Nayak 1996) greatly focuses this process by using conflicts. In this context, a conflict is a partial assignment to $\mathcal{T}$ and $\mathcal{O}$ that is inconsistent. When a candidate solution is found to be inconsistent, the conflict is recorded in a database, $ConflictDB$. No further candidates that contain a known conflict are generated.

**Example 5** Figure 4 illustrates the two lowest cost solutions to the above problem would be found by *CBFS*. They represent a single failure of rank 1 at time 1 and a double failure of rank 2 at time 2, respectively.

While applying CBFS to the full transition system exactly enumerates the most likely trajectories, and thus states, in order, problem size is a significant issue. Let $p$ denote the number of propositions needed to represent each possible value of each variable in $\mathcal{T} \cup \Pi \cup \mathcal{C} \cup \mathcal{D} \cup \mathcal{O}$. These propositions are constrained by a copy of $\mathcal{M}_\mathcal{T}$ and $\mathcal{M}_\Sigma$ at each time step. Testing consistency of an $m$-step candidate trajectory is a consistency problem of $m \times p$ propositions and $m \times |\mathcal{M}_\mathcal{T} \cup \mathcal{M}_\Sigma|$ clauses. For the Deep Space 1 model, this is $m \times 4041$ propositions and $m \times 13,503$ clauses. The remainder of this paper discusses methods that reduce the size of the consistency problem to be solved, eventually deriving a method that allows a constant problem size.

## Problem Size Reduction

In this section, we reduce the structure needed to represent the evolution of the system at a time point from a complete copy of the system model to a small number of variables and clauses. Intuitively, when a command is issued to the system, only a small number of components participate in transmitting that command through the system or transitioning in response to the command. Consider Figure 5. The squares represent state variables, the lines sets of constraints from $\mathcal{M}_\mathcal{T}$. As of time 7, the valves, pump and VDU have not been commanded nor have they interacted with other components by passing a command. If we did not detect a failure of any of these components, we can represent the possibility that they remained idle or failed in a localized and unobservable way with a single set of variables and con-
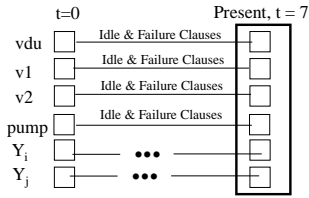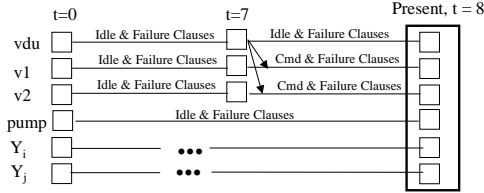
Figure 5: Evolution before commanding the valves



Figure 6: Evolution upon commanding the valves

straints as illustrated. At time 7 we command the valves on. We require variables $v1_8$ and $v2_8$ to represent the new states of the valves. $\mathcal{M}_\mathcal{T}$ suggests $vdu_7$, $v1_7$ and $v2_7$ will interact with $v1_8$ and $v2_8$. These variables, along with necessary transition variables $\tau_{vdu,7}$, $\tau_{v1,7}$ and, $\tau_{v2,7}$, are introduced to the system with the appropriate clauses from $\mathcal{M}_\mathcal{T}$. For each other variable $y$, the variable representing $y_7$ is adequate to represent $y_8$. Figure 6 illustrates this process. In order to derive a well-founded algorithm from these intuitions, we first place a natural restriction on $\mathcal{M}_\mathcal{T}$ that does not impact correctness. Second we introduce an approximation involving $\mathcal{M}_\Sigma$ that, importantly, does not rule out consistent trajectories. Instead, some trajectories that are not consistent with past observations may be admitted, with the possibility that future observations will eliminate them. These problem modifications avoid replication of many variables in $\Pi$ and $\mathcal{D}$, as well as corresponding constraints from $\mathcal{M}_\mathcal{T}$ and $\mathcal{M}_\Sigma$.

### Restricting $\mathcal{M}_\mathcal{T}$

We restrict $\mathcal{M}_\mathcal{T}$ as do *Livingstone* and *Burton* (Williams & Nayak 1997): a component moves to a failure state with equal probability from any state, and except for failures a component that does not receive a command idles in its current state. $\mathcal{M}_\mathcal{T}$ is limited to the forms:

$$(\tau_{y,t} = \tau_{failure}) \Rightarrow y_{t+1} = y_{failure}$$
$$(\mathcal{C}_{y,t} = \mathcal{C}^*) \wedge \phi_t \wedge (\tau_{y,t} = nominal) \Rightarrow y_{t+1} = y^*$$
$$(\mathcal{C}_{y,t} = idle) \wedge (\tau_{y,t} = nominal) \Rightarrow y_{t+1} = y_t$$

where $\phi_t$ is a propositional formula over $\Pi_t \cup \mathcal{D}_t$, $\mathcal{C}^* \in \delta(\mathcal{C}_{y,t})$, $nominal \in \delta(\tau_{y,t})$ and $\tau_{fail} \in \delta(\tau_{y,t})$. Formulae of the first form model failures while formulae of the second form model nominal, commanded transitions. Formula of the third form are frame axioms that encode our assumption that devices that do receive a command remain in their current state. We replace $\phi_t$ with implicant $\pi_t$, an equivalent formula involving only $\Pi_t$. Intuitively $\phi_t$ is a formula involving $\mathcal{D}$ that, given $\mathcal{M}_\Sigma$ and an assignment to $\Pi$, allows us to infer if $\mathcal{C}_{y,t}$ propagates through a set of components to component $y$. To form $\pi_t$, we replace each assignment to $\mathcal{D}_t$

with a set of assignments from $\Pi_t$ that imply the $\mathcal{D}_t$ assignment under $\mathcal{M}_\Sigma$. We expect that for the type of clauses $\mathcal{M}_\mathcal{T}$ contains, growth in $\pi_t$ will be proportional to the length of the component chain that transmits $\mathcal{C}_{y,t}$, which ranged from 1 to 5 in (Bernard *et al.* 1998). Our experience supports this hypothesis. This growth is offset as non-idle, non-failure clauses take the following form which is independent of $\mathcal{D}$.

$$(\mathcal{C}_{y,t} = \mathcal{C}^*) \wedge \pi_t \wedge (\tau_{y,t} = nominal) \Rightarrow y_{t+1} = y^*$$

Given a $\mathcal{C}_{y,t}$ which is not idle, in order to determine consistency with $\mathcal{M}_\mathcal{T}$ we now need only introduce $\mathcal{C}_{y,t}$, $\tau_{y,t}$ and those select members of $\Pi_t$ that appear in $\pi_t$.

### Eliminating intermediate observations

$\mathcal{M}_\Sigma$ remains, and requires introduction of all variables in $\Pi_t$ and $\mathcal{D}_t$ in order to check consistency against $\mathcal{O}_t$. We proceed by eliminating all variables $\mathcal{O}_t$ for values of $t$ sufficiently far in the past. That is to say, transition choices are only constrained by consistency between the trajectories they imply and recent observations. As the system evolves, variables representing older observations and the copies of $\mathcal{M}_\Sigma$ that constrain them are unneeded. For the portions of the trajectory where $\mathcal{M}_\Sigma$ is not introduced, we need not introduce $\mathcal{D}$ and need only introduce the limited portion of $\Pi_t$ required by $\mathcal{M}_\mathcal{T}$. This is of course an approximation. It is now possible to choose transition assignments that are inconsistent with the discarded observations, resulting in an "imposter" trajectory. This approximation has several important features. First, it is a conservative approximation in that no consistent trajectories are eliminated. Second, all trajectories are checked against new observations, and imposters are eliminated as soon as they fail to describe the on-going evolution of the system. Finally if conflicts are recorded in $ConflictDB$, no partial assignment to $\mathcal{T}$ that was discovered to be in conflict with the observations will be reconsidered, even after observations are discarded. Thus we can only admit an imposter in the case where a transition choice is in conflict with an observation, but the choice is not considered until after the conflicting observation has been discarded.

### Selective Model Extension

Based upon these restrictions, the procedure *extend* introduces into time step $t$ only the small fraction of the model involved with the evolution of the system due to the command $\mathcal{C}_{y,t} = \mathcal{C}^*$. The resulting problem size per time step is proportional to $| \pi_t |$. This hinges upon Theorem 1. For the purpose of discussion we will assume that for each time step $t$ there exists only one $y$ for which $\mathcal{C}_{y,t} \neq idle$. The proofs can be extended to parallel commanding.

**Theorem 1** *Assume $\mathcal{C}_{y,t} = \mathcal{C}^*$, $\mathcal{C}^* \neq idle$, and for all $x \neq y$, $\mathcal{C}_{x,t} = idle$. Consider the formula of $\mathcal{M}_\mathcal{T}$*

$$(\mathcal{C}_{y,t} = \mathcal{C}^*) \wedge \pi_t \wedge (\tau_{y,t} = nominal) \Rightarrow y_{y+1} = y^*$$

*For all state variables $x_t$, $x \neq y$, if $x_t \notin \pi_t$, then an equivalent consistency problem is formed by replacing $x_t$, $\tau_{x,t}$ and all formulae of $\mathcal{M}_\mathcal{T}$involving these variables with a constraint between $x_{t-1}$ and $x_{t+1}$.*

Space precludes inclusion of the complete proof. Intuitively, there are no witnesses to the value of $x_t$ except for $x_{t-1}$ and $x_{t+1}$, which can be constrained directly. If $x_t$ is as described, then the only clauses involving $x_t$ are of the form:

$$(\mathcal{C}_{x,t-1} = \mathcal{C}^*) \wedge \phi_{t-1} \wedge (\tau_{x,t-1} = nominal) \quad \Rightarrow x_t = x^*$$
$$(\mathcal{C}_{x,t-1} = idle) \wedge (\tau_{x,t-1} = nominal) \quad \Rightarrow x_t = x_{t-1}$$
$$(\tau_{x,t-1} = \tau_{fail}) \quad \Rightarrow x_t = x_{fail}$$
$$(\mathcal{C}_{x,t} = idle) \wedge (\tau_{x,t} = nominal) \quad \Rightarrow x_{t+1} = x_t$$
$$(\tau_{x,t} = \tau_{fail}) \quad \Rightarrow x_{t+1} = x_{fail}$$

The variable $x_t$ can only impact the consistency of the system via the assignments to $\tau_{x,t-1}$ and $\tau_{x,t}$. Given the independence assumptions, assigning failures to both is indistinguishable from and less likely than assigning $\tau_{x,t-1} = nominal$ and $\tau_{x,t}$ to a failure, while assigning a failure to one is equivalent to assigning a failure to the other. Thus we need only consider $\tau_{x,t-1} = \tau_{x,t} = nominal$ and $\tau_{x,t-1} = nominal$, $\tau_{x,t} = \tau_{fail}$. In the nominal case, $x_t$ is equivalent to $x_{t+1}$ and can be eliminated. In the failure case, the assignment to $x_t$ has no impact on $x_{t+1}$ and can be eliminated. The above formula are rendered equivalent to the following reduced set:

$$(\mathcal{C}_{x,t-1} = \mathcal{C}^*) \wedge \pi_{t-1} \wedge (\tau_{x,t-1} = nominal) \quad \Rightarrow x_{t+1} = x^*$$
$$(\mathcal{C}_{x,t-1} = idle) \wedge (\tau_{x,t-1} = nominal) \quad \Rightarrow x_{t+1} = x_{t-1}$$
$$(\tau_{x,t-1} = \tau_{fail}) \quad \Rightarrow x_{t+1} = x_{fail}$$

In fact, at time $t$ we will know whether or not $\mathcal{C}_{x,t-1} = idle$, and therefore we need only introduce one of the first two formulae. The $extend$ procedure repeatedly applies Theorem 1 to avoid introducing a variable or constraints for $x_t$ when there have been no witnesses to $x_t$ and it is possible to constrain $x_{t+1}$ directly from $x_{t-1}$. When a command is introduced, the compiled $\mathcal{M}_{\mathcal{T}}$ determines what clauses should be added to constrain the nominal transition of $y_t$ under $\mathcal{C}_{y,t}$. State variables appearing in the introduced clauses are added, along with constraints representing their idle or failure transitions. By reducing the number of variables and clauses introduced at each time step, we reduce the consistency problem involved in checking a trajectory to a number of variables proportional to $m \times | \pi_t |$. The number of clauses is proportional to $m \times (| \pi_t | + k)$ where $k$ is the number of failure values per $\tau_y$ domain.

## Conflict Coverage Search

The strengths of efficiently tracking a partial belief state are merged with the flexibility of incrementally enumerating belief states in the *CoverTrack* procedure of Figure 7. $TSet$ is a superset of all consistent trajectories of rank $\gamma$, as returned by a previous call to *CoverTrack*. As described above, *extend* adds to the transition system the variables needed to represent the outcomes of the current command. All trajectories are augmented by the new transition variables, which are assigned nominal transition, and checked for consistency. Any inconsistent trajectory requires additional failures above rank $\gamma$, and is discarded as relatively implausible. The survivors are a superset of all consistent trajectories of rank $\gamma$. If this set is not empty, it is returned. Otherwise, the most likely trajectory has a rank greater than $\gamma$. The *GenerateCover* algorithm generates all assignments to $\mathcal{T}$ of

```
proc CoverTrack(cmd, obs, TSet, ConflictDB, γ) {
    /*Extend the system adding Π_t to Π, 𝒯_t to 𝒯*/
    extend(Π,𝒯, cmd);
    /*Extend trajectories at current γ */
    Assign 𝒯_t to nominal, 0 rank assignment.
    for trajectory in TSet
        trajectory = trajectory ∪𝒯_t;
    /*Check trajectories for consistency, up γ if needed*/
    Assign 𝒪 according to obs received;
    Survivors = ∅;
    loop{
        for trajectory in TSet {
            conflict=checkConsistency(trajectory);
            if (conflict) then
                push(conflict, ConflictDB);
            else
                push(trajectory,survivors); }
        if(survivors) then return survivors;
        /*Ran out of trajectories. Find more at next rank*/
        γ = γ + 1;
        TSet=GenerateCover(𝒯,ConflictDB,γ);
    }
}
```

Figure 7: Conflict Coverage Tracking Procedure

a given rank that cover all known conflicts. A conflict is covered if at least one of the variables in the conflict is assigned to an assignment that does not appear in the conflict. Intuitively, we leave the $\tau_{y,t}$ at their zero rank values, introducing reassignment only to avoid conflicts, with a total cost of $\gamma$. This is the NP-hard *hitting set* problem. The contents of $ConflictDB$ and $\gamma$ will determine whether this problem is tractable. Because of the loss of observations at past time points, *GenerateCover* returns superset of all consistent rank $\gamma$ trajectories. If at least one trajectory is consistent with the current observations, it is returned. If not, $\gamma$ is increased.

## Finite Horizons

While selective extension reduces the variables per time step, we still require an unbounded number of variables over time. We avoid this requirement by summarizing sets of $\tau_{y,t}$ variables beyond a horizon $h$ in the past into a single variable $\tau_h$. This horizon is fixed relative to the present, so at time step $m$, only the $\mathcal{T}$ variables $\tau_{y,m-h}$ through $\tau_{y,m}$ are required. Consider Figure 6 extended to some time step $m$. The variables $\tau_{y,0}$ through $\tau_{y,m}$ have been introduced to represent choices in the system's evolution. When tracking the system, we incrementally generate the few most likely consistent assignments to all $\tau_{y,t}$, representing the most likely consistent trajectories. Note that each $m$-step trajectory contains an assignment to $\tau_{y,0}$ and $\tau_{y,1}$ that appears among the most likely given a potentially large amount of information gained from time steps 0 through $m$. Each such assignment also induces an assignment upon $\Pi_2$, for example $y_2 = y^*$. Intuitively, we replace each of $l$ likely assignments to $\tau_{y,0}$ and $\tau_{y,1}$ with an assignment $\tau_h = choice_l$ that has the same rank. We may then replace $\mathcal{M}_{\mathcal{T}2}$ with $l$ clauses of the form $\tau_h = choice_l \Rightarrow y_2 = y^*$. The summary variable $\tau_h$ restricts choices for the initial portion of the trajectory to the partial
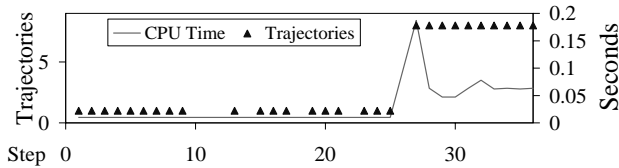
Figure 8: ISPP - Independent failures at steps 27, 32 and 33

trajectories that appeared most likely after being extended for some time. By summarizing the $l$ most likely assignments to $\tau_h$ and all $\tau_{y,2}$ into a new variable $\tau_{h'}$ when the front of the transition system is extended, we can maintain a fixed sized problem representation. Since our search algorithms are continually considering the most likely choices for $\mathcal{T}$ and inferring the implications upon $\Pi$ in order to determine consistency, this is a relatively low cost approximation to compute.

## Results

*L2* has been implemented in C++ in a modular form that allows alternative search and consistency procedures to be plugged into the transition system framework. The tests described below were performed using *CoverTrack* and propositional consistency as determined by an LTMS, running under Windows NT on a 550Mhz Pentium III. Observations were kept for one step only. No horizon was used.

*L2* correctly tracks the canonical scenarios known to confound *Livingstone*. Consider Example 2. When the pump is turned on, *Livingstone* finds two conflicts in the current mode assignments: valve v1 cannot be open, and valve v2 cannot be open. It thus fails both valves. *L2* finds the following sets of devices that could not have both transitioned nominally: {VDU, v1}, {VDU,v2}. The lowest cost covering is to fail the VDU at time step 0. Many more interesting scenarios have been demonstrated. If the VDU is failed and v1 is commanded open, the trajectory wherein v1 is stuck will be tracked if that is more likely than a VDU failure. If v2 is later commanded and no flow results, the v1 failure is dropped and the trajectory where the VDU failed in the past is found. If the VDU is resettable, the trajectory wherein the VDU has failed in a resettable manner is first tracked when multiple valves fail to open. If the VDU is reset and the valves again fail to open, *L2* may find a trajectory where the valves were stuck all along, one that replaces the past resettable VDU failure with a permanent failure, or both, depending upon the ranks of the various failures.

Longer runs on more complex models written by *Livingstone* users rather than the authors were also performed. The ISPP model has 59 components and represents a chemical processor designed to produce rocket fuel from the Martian atmosphere. Its *flow failure* requires far more time for diagnosis under *Livingstone* than any other scenario we have encountered. Figure 8 illustrates a 33 step track of the model, approximating one day's worth of commands. On the $27^{th}$ step, the flow failure becomes observable. On steps 32 and 33 simpler, unrelated failures occur. Figure 8 illustrates a second tracking run of the same model. Note
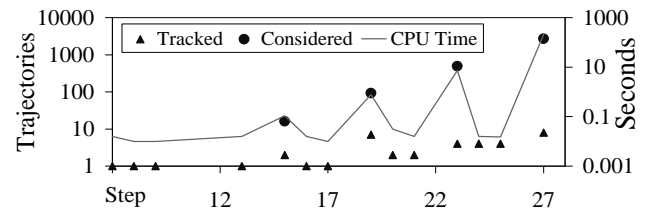


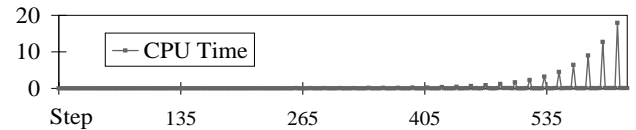Figure 9: ISPP - 4 Identical failures over 27 Steps



Figure 10: CB - 39 Identical failures over 618 Steps

that the time axis is logarithmic. On step 15 the flow failure is introduced. Repair actions are taken and the failure is immediately reintroduced, until a total of four identical failures have occurred. The CB model of 24 electrical components connected in series and parallel was tracked in runs of 618 steps. Figure 10 illustrates a run wherein every 16 steps the same set of devices is turned on, a device fails and is reset, and the devices are turned off. The device fails a total of 39 times. Additional runs were made on both models.

Our results suggest the following. *Model growth per time step is small.* ISPP begins at 2933 clauses and grows by an average of 36 clauses per time step. CB begins at 1126 clauses and grows by an average of 44 clauses per time step. *Tracking time steps where no failure occurs takes a very small amount of CPU time.* Note that in Figure 8 the steps before the first failure occurs require a negligible amount of CPU time. The nominal steps after the failure take slightly more time, as 8 trajectories are being tracked, but the cost is still negligible. *Keeping a history does not induce an unreasonable cost when diagnosing a single failure.* When the nominal trajectory is ruled out we have a single, long conflict and $\gamma = 1$, leading to a simple coverage problem. The CPU time for a single CB failure scenario is below clock resolution whether 15 or over 600 nominal steps precede the failure. In Figure 9, *L2* finds the eight trajectories that explain the failure that became observable on step 27 in 0.19 seconds. *Because of the accumulation of conflicts, tracking the system through $k$ failures spread over time can be an easier problem than diagnosing a single failure of cardinality $k$.* Consider the run of Figure 8. On step 27, the flow failure occurs, causing the large spike in CPU time. Eight trajectories result and are tracked until step 32. On step 32, a simpler, unrelated failure occurs, and none of the 8 trajectories is consistent when extended by the nominal transition. Note that *L2* must now rediagnose the entire history of the system including the flow failure. It does so in just 0.08 seconds, less than half of the time required to diagnose the flow failure alone. The key to this behavior is the conflicts. On step 27, the nominal trajectory is ruled out and $ConflictDB$ contains a single conflict. *GenerateCover* returns 28 candidate

trajectories, 20 of which are ruled out, adding another 15 candidates to $ConflictDB$. Calling $GenerateCover$ with the same $\gamma$ on these conflicts almost immediately returns just the 8 consistent candidates. On step 32, the 8 diagnoses are ruled out by conflicts resulting from the simple failure. Since the conflicts from the simple failure involve none of the variables from the flow failure conflicts, the problem decomposes into two subproblems, one of which has previously been solved and memoized by the conflicts. *For these classes of problems,* L2 *has adequate performance.* As a practical matter, each test above executes several times faster than *Livingstone*'s single diagnosis of the flow failure. On the above problems, the additional work performed is dominated by the size and speed benefits of porting from Lisp. *Unfortunately, tracking $k$ related failures over time can also be as computationally intensive as diagnosing a cardinality $k$ failure.* Figure 9 illustrates a sequence of failures where the conflict coverage problem does not decompose. At each time peak, the flow failure has occurred again. The conflicts generated by the fourth failure involve exactly the devices involved in the first three failures. As a result, the time required to solve the hitting set problem and the number of inconsistent trajectories considered rises dramatically. At the fourth failure, 2694 candidates are returned in 174 seconds. An additional 33 seconds are spent determining all but 8 of them are inconsistent. Figure 10 clearly shows the exponential growth of tracking time as the number of failures involving the same device grows.

## Future Work

Interleaving consistency checking and conflict coverage may significantly cut down on the number of candidates returned by *GenerateCover*. We have not yet run tests with a horizon. A fixed horizon limits the search that can be done and cuts off consideration of overlapping conflicts beyond the horizon. A more interesting approach is to iteratively deepen the horizon as time allows or uncertainty requires. A recency bias may be a practical heuristic. Only exploring each device's history up to the last point it was considered to have failed should reduce the explosion in possible trajectories. Selectively reintroducing observations and small portions of $\mathcal{M}_\Sigma$ at past time points should also clamp the growth of trajectories. We are currently investigating these and other extensions to $L2$. The resulting system will be evaluated on Earth-bound testbeds representing an interferometer and a Mars propellant plant. In addition, it will be flown as an experiment on the X-34 rocket plane in 2001 and the X-37 orbital vehicle in 2002.

## Related Work

The problem described is a partially observable Markov decision process with focus placed upon belief revision. (Friedman & Halpern 1999) provides an excellent synthesis of the literature in belief revision and belief update. The authors describe a general, plausibility-based temporal logic framework that can be used to describe revision methods such as $L2$. There also exists a large body of work concerning approximate belief update. (Boyen & Koller 1998)

for example provides an approximate, factored belief state with a bounded error that can be updated without enumerating the state space. Unfortunately, the systems we consider have inadequate mixing rates to apply this approximation. $L2$ differs from this work and the other approximations of which the authors are aware in that it uses history to compensate for not having a sufficient statistic.

## Conclusions

This paper presents incremental belief state generation as an alternative to belief revision. The described approximations create a family of representations that track an exact model for a number of steps, then track a reduced model, then summarize over the most likely initial trajectories. The uniform nature of the three abstractions allows a single, simple search to be employed. *CoverTrack* combines partial belief state propagation with the flexibility of the transition system representation. It is highly efficient when failures are sufficiently independent. We are investigating methods to improve performance when multiple failures involving the same components cause a highly unconstrained search.

## Acknowledgements

## References

Bernard, D. E.; Dorais, G. A.; Fry, C.; Jr., E. B. G.; Kanefsky, B.; Kurien, J.; Millar, W.; Muscettola, N.; Nayak, P. P.; Pell, B.; Rajan, K.; Rouquette, N.; Smith, B.; and Williams, B. C. 1998. Design of the remote agent experiment for spacecraft autonomy. In *Procs. IEEE Aerospace*.

Boyen, X., and Koller, D. 1998. Tractable inference for complex stochastic processes. 33–42.

de Kleer, J., and Williams, B. C. 1989. Diagnosis with behavioral modes. In *Procs. IJCAI-89*, 1324–1330.

Dressler, O., and Struss, P. 1992. Back to defaults: Characterizing and computing diagnoses as coherent assumption sets. In *Procs. ECAI-92*.

Friedman, N., and Halpern, J. Y. 1999. Modeling belief in dynamic systems part ii: Revision and update. *JAIR* 10:117–167.

Goldszmidt, M., and Pearl, J. 1992. Rank-based systems: A simple approach to belief revision, belief update, and reasoning about evidence and actions. In *Procs. KR-92*, 661–672.

Muscettola, N.; Nayak, P. P.; Pell, B.; and Williams, B. C. 1998. Remote Agent: To boldly go where no AI system has gone before. *Artificial Intelligence* 103:5–47.

Struss, P. 1997. Fundamentals of model-based diagnosis of dynamic systems. In *Procs. IJCAI-97*. 480–485.

Williams, B. C., and Nayak, P. P. 1996. A model-based approach to reactive self-configuring systems. In *Procs. AAAI-96*, 971–978.

Williams, B. C., and Nayak, P. P. 1997. A reactive planner for a model-based executive. In *Procs. IJCAI-97*.