Fault Tolerant Planning: Toward Probabilistic Domain Models in Symbolic Non-Deterministic Planning

Rune M. Jensen RUNEJ@CS.CMU.EDU
Manuela M. Veloso MMV@CS.CMU.EDU
Randal E. Bryant BRYANT@CS.CMU.EDU
Computer Science Department, Carnegie Mellon University,
Pittsburgh, PA 15213-3891, USA

Abstract

Symbolic non-deterministic planning represents action effects as sets of possible next states. In this article, we move toward a more probabilistic uncertainty model by distinguishing between likely primary effects and unlikely secondary effects of actions. We consider the practically important case, where secondary effects are failures, and introduce n-fault tolerant plans that are robust for up to n faults occurring during plan execution. Fault tolerant plans are more restrictive than weak plans, but more relaxed than strong cyclic and strong plans. We show that optimal n-fault tolerant plans can be generated by the usual strong algorithm. However, due to non-local error states, it is often beneficial to decouple the planning for primary and secondary effects. We employ this approach for two specialized algorithms 1-FTP (blind) and 1-GFTP (guided) and demonstrate their advantages experimentally in significant real-world domains.

1. Introduction

MDP solving (e.g., Sutton & Barto, 1998) and Symbolic Non-Deterministic Planning (SNDP e.g., Cimatti, Pistore, Roveri, & Traverso, 2003) can be regarded as two alternative frameworks for solving planning problems with uncertain outcomes of actions. Both frameworks are attractive, but for quite different reasons. The main advantage of MDP solving is the high expressive power of the domain model: for each state in the MDP, the effect of an action is given by a probability distribution over next states. The framework, however, is challenged by a high complexity of solving MDPs. The main advantage of SNDP is its scalability. The domain model has less expressive power than an MDP. Action effects are modeled as sets of possible next states instead of probability distributions over these states. This allows powerful symbolic search methods based on Binary Decision Diagrams (BDDs, Bryant, 1986) to be applied. SNDP, however, is challenged by its coarse uncertainty model of action effects. The current solution classes are suitable when a pure disjunctive model of action effects is sufficient (e.g., for controlling worst-case behavior). However, when this is not the case, they often become too relaxed (weak plans) or too restrictive (strong cyclic and strong plans).

A large body of work in MDP solving addresses the scalability problem (e.g., Sutton & Barto, 1998; Tesauro, 1995). In particular, symbolic methods based on Algebraic Decision Diagrams (ADDs, Bahar, Frohm, Gaona, Hachtel, Macii, Pardo, & Somenzi, 1993) have been successfully applied to avoid explicitly enumerating states (Hoey, St-Aubin, & Hu, 1999; Feng & Hansen, 2002).

A dual effort in SNDP, where the uncertainty model of action effects is brought closer to its probabilistic nature, is still lacking. In this article, we take a first step in this direction by introducing a new class of fault tolerant non-deterministic plans. Our work is motivated by two observations:

1. Non-determinism in real-world domains is often caused by infrequent errors that make otherwise deterministic actions fail.

2. Normally, no actions are guaranteed to succeed.

Due to the first observation, we propose a new uncertainty model of action effects in SNDP that distinguishes between primary and secondary effects of actions. The primary effect models the usual deterministic behavior of the action, while the secondary effect models error effects. Due to the second observation, we introduce *n-fault tolerant plans* that are robust for up to n errors or faults occurring during plan execution. This definition of fault tolerance is closely connected to fault tolerance concepts in control theory and engineering. Every time we board a two engined aircraft, we enter a 1-fault tolerant system: a single engine failure is recoverable, but two engines failing may lead to an unrecoverable breakdown of the system.

An n-fault tolerant plan is not as restrictive as a strong plan that requires that the goal can be reached in a finite number of steps independent of the number of errors. In many cases, a strong plan does not exist because all possible errors must be taken into account. This is not the case for fault tolerant plans, and if errors are infrequent, they may still be very likely to succeed. A fault tolerant plan is also not as restrictive as a strong cyclic plan. An execution of a strong cyclic plan will never reach states not covered by the plan unless, it is a goal state. Thus, strong cyclic plans also have to take all error combinations into account. Weak plans, on the other hand, are more relaxed than fault tolerant plans. Fault tolerant plans, however, are almost always preferable to weak plans because they give no guarantees for *all* the possible outcomes of actions. For fault tolerant plans, any action may fail, but only a limited number of failures are recoverable.

One might suggest using a deterministic planning algorithm to generate n-fault tolerant plans. Consider for instance synthesizing a 1-fault tolerant plan in a domain, where there is a non-faulting plan of length k and at most f error states of any action. It is tempting to claim that a 1-fault tolerant plan then can be found using at most kf calls to a classical deterministic planning algorithm. This analysis, however, is flawed. It only holds for evaluating a given 1-fault tolerant plan. It neglects that many additional calls to the classical planning algorithm may be necessary in order to *find* a valid solution. Instead, we need an efficient approach for finding plans for many states simultaneously. This can be done with the BDD-based approach of SNDP.

The article contributes a range of unguided as well as guided algorithms for generating fault tolerant plans. We first observe that an n-fault tolerant planning problem can be reduced to a strong planning problem and solved with the strong planning algorithm. The resulting algorithm is called n-FTP$_S$. Since the performance of blind strong planning is limited, we also consider a guided version of n-FTP$_S$ called n-GFTP$_S$ using the approach introduced in Jensen, Veloso, and Bryant (2003). The n-GFTP$_S$ algorithm is efficient, when secondary effects are local in the state space, because they then will be covered by the search beam of n-GFTP$_S$. In practice, however, secondary effects may be permanent malfunctions that due to their impact on the domain cause a transition to a non-local state. To solve this problem, we decouple the planning for primary and secondary effects. We restrict our investigation to 1-fault tolerant planning and introduce two algorithms: 1-FTP and 1-GFTP using blind and guided search, respectively.

The algorithms have been implemented in the BIFROST search engine (Jensen, 2003b) and experimentally evaluated on a range of domains including three real-world domains: DS1 (Pecheur & Simmons, 2000), PRS (Thiébaux & Cordier, 2001), and SIDMAR (Fehnker, 1999). The purpose of the experiments is twofold. First, we want to characterize the performance of the four algorithms introduced in the article and investigate to what extend secondary effects of realistic actions transition to non-local states that require specialized heuristic algorithms like 1-GFTP to be handled efficiently. Second, we want to study concrete examples of fault tolerant planning to understand the advantages and limitations of this approach. With respect to the first objective, the experiments show that there is a natural connection between the existence of fault tolerant plans and the redundancy of the modeled system. Surprisingly, however, it turns out that even 1-fault plans require a fairly high level of redundancy since the plans must be able to recover from a failure happening at any point of execution. With respect to the second objective, the results from the SIDMAR domain shows that realistic failures of production machines are more efficiently solved by decoupling the guidance of the planning for primary and secondary effects as done by 1-GFTP.

The reminder of this article is organized as follows. In Section 2, we define the current three classes of non-deterministic plans and present a blind and guided BDD-based algorithm for synthesizing these plans. Section 3 introduces n-fault tolerant planning problems and defines valid and optimal n-fault tolerant plans in terms of induced non-deterministic plans. In Section 4, we introduce n-FTP$_S$ and n-GFTP$_S$ for generating n-fault tolerant plans via the blind and guided strong planning algorithm and the specialized blind and guided algorithms 1-FTP and 1-GFTP that decouples the planning for primary and secondary effects. The characteristics of fault tolerant plans and the performance of the developed algorithms are empirically evaluated in Section 5. A discussion of related work in AI and control theory is given in Section 6, and finally, in Section 7, we draw conclusions and suggest directions for future work.

2. Symbolic Non-deterministic Planning

A *non-deterministic planning domain* is a tuple $\langle S, Act, \rightarrow\rangle$, where S is a finite set of states, Act is a finite set of actions, and $\rightarrow \subseteq S \times Act \times S$ is a non-deterministic transition relation of action effects. Instead of $(s, a, s') \in \rightarrow$, we write $s \xrightarrow{a} s'$. The set of next states of an action a applied in state s is given by

$$\textsc{Next}(s,a) \equiv \{s' : s \xrightarrow{a} s'\}.$$

An action a is called *applicable* in state s iff $\textsc{Next}(s,a) \neq \emptyset$. The set of applicable actions in a state s is given by

$$\textsc{App}(s) \equiv \{a : \textsc{Next}(s,a) \neq \emptyset\}.$$

A *non-deterministic planning problem* is a tuple $\langle \mathcal{D}, s_0, G\rangle$, where $\mathcal{D}$ is a non-deterministic planning domain, $s_0 \in S$ is an initial state, and $G \subseteq S$ is a set of goal states. Let $\mathcal{D}$ be a non-deterministic planning domain. A *state-action pair* $\langle s, a\rangle$ of $\mathcal{D}$ is a state $s \in S$ associated with an applicable action $a \in \textsc{App}(s)$. A *non-deterministic plan* is a set of state-action pairs (SAs) defining a function from states to sets of actions relevant to apply in order to reach a goal state. States are assumed to be fully observable. An execution of a non-deterministic plan is an alternation between observing the current state and choosing an action to apply from the set of actions associated with the state. The set of states *covered* by a plan π is given by

$$\textsc{States}(\pi) \equiv \{s \,:\, \exists a \,.\, \langle s, a\rangle \in \pi\}.$$

The set of possible end states of a plan is given by

$$\textsc{Closure}(\pi) \equiv \{s' \notin \textsc{States}(\pi) \,:\, \exists \langle s, a\rangle \in \pi \,.\, s' \in \textsc{Next}(s, a)\,\}.$$

An execution of a *strong plan* is guaranteed to reach states covered by the plan until a goal state is reached after a finite number of steps. An execution of a *strong cyclic plan* is also guaranteed to reach states covered by the plan or a goal state. However, due to cycles, it may never reach a goal state. An execution of a *weak plan* may reach states not covered by the plan, it only guarantees that some execution exists that reaches the goal from the initial state.

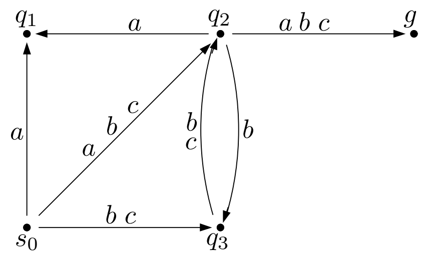

Figure 1: A non-deterministic planning domain $\langle S, Act, \rightarrow\rangle$ where $S = \{s_0, q_1, q_2, q_3, g\}$, $Act = \{a, b, c\}$ and $s \xrightarrow{x} s'$ holds iff there is a transition in the graph from s to s' labeled x.

Example 1 Consider a planning problem for the non-deterministic planning domain shown in Figure 1 where the initial state is s_0 and g is the only goal state. Let

$$\begin{aligned}\pi_w &= \{\langle s_0, a\rangle, \langle q_2, a\rangle\},\\ \pi_{sc} &= \{\langle s_0, b\rangle, \langle q_2, b\rangle, \langle q_3, b\rangle\},\\ \pi_s &= \{\langle s_0, c\rangle, \langle q_2, c\rangle, \langle q_3, c\rangle\}.\end{aligned}$$

We have, π_w is a valid weak plan, but not a valid strong cyclic or strong plan, π_{sc} is a valid weak and strong cyclic plan, but not a valid strong plan, and π_s is a valid weak, strong cyclic, and strong plan. ◇

Following Cimatti et al. (2003), we use CTL to formally define weak, strong cyclic, and strong plans. CTL specifies the behavior of a system represented by a *Kripke structure*. A Kripke structure is a pair $\mathcal{K} = \langle S, R \rangle$, where S is a finite set of states and $R \subseteq S \times S$ is a total transition relation. An *execution tree* is formed by designating a state in the Kripke structure as an initial state and then unwinding the structure into an infinite tree with the designated state as root.

We consider a subset of CTL formulas with two *path quantifiers* $\mathtt{A}$ ("for all execution paths") and $\mathtt{E}$ ("for some execution path") and one *temporal operator* $\mathtt{U}$ ("until") to describe properties of a path through the tree. Given a finite set of states S, the syntax of CTL formulas are inductively defined as follows

- Each element of 2^S is a formula,
- $\neg\psi$, $\mathtt{E}(\phi \,\mathtt{U}\, \psi)$, and $\mathtt{A}(\phi \,\mathtt{U}\, \psi)$ are formulas if ϕ and ψ are.

In the following inductive definition of the semantics of CTL, $\mathcal{K}, q \models \psi$ denotes that ψ holds on the execution tree of the Kripke structure $\mathcal{K} = \langle S, R \rangle$ rooted in the state q

- $\mathcal{K}, q_0 \models P$ iff $q_0 \in P$,
- $\mathcal{K}, q_0 \models \neg\psi$ iff $\mathcal{K}, q_0 \not\models \psi$,
- $\mathcal{K}, q_0 \models \mathtt{E}(\phi \,\mathtt{U}\, \psi)$ iff there exists a path $q_0 q_1 \cdots$ and $i \geq 0$ such that $\mathcal{K}, q_i \models \psi$ and, for all $0 \leq j < i$, $\mathcal{K}, q_j \models \phi$,
- $\mathcal{K}, q_0 \models \mathtt{A}(\phi \,\mathtt{U}\, \psi)$ iff for all paths $q_0 q_1 \cdots$ there exists $i \geq 0$ such that $\mathcal{K}, q_i \models \psi$ and, for all $0 \leq j < i$, $\mathcal{K}, q_j \models \phi$.

We will use three abbreviations $\mathtt{AF}\,\psi \equiv \mathtt{A}(S \,\mathtt{U}\, \psi)$, $\mathtt{EF}\,\psi \equiv \mathtt{E}(S \,\mathtt{U}\, \psi)$, $\mathtt{AG}\,\psi \equiv \neg\mathtt{EF}\neg\psi$. Since S is the complete set of states in the Kripke structure, the CTL formula S holds in any state. Thus, $\mathtt{AF}\,\psi$ means that for all execution paths a state, where ψ holds, will eventually be reached. Similarly, $\mathtt{EF}\,\psi$ means that there exists an execution path reaching a state, where ψ holds. Finally, $\mathtt{AG}\,\psi$ holds, if every state on any execution path satisfies ψ. We will often consider CTL formulas on sets of states. To simplify the presentation, we therefore introduce the short notation

$$\mathcal{K}, Q \models \psi \quad \equiv \quad \forall q \in Q \,.\, \mathcal{K}, q \models \psi.$$

The *execution model* of a plan π for the problem $\langle \mathcal{D}, s_0, G \rangle$ of the domain $\mathcal{D} = \langle S, Act, \rightarrow \rangle$ is a Kripke structure $\mathcal{M}(\pi) = \langle S, R \rangle$, where

- $S = \textsc{Closure}(\pi) \cup \textsc{States}(\pi) \cup G$,
- $\langle s, s' \rangle \in R$ iff $s \notin G$, $\exists a \,.\, \langle s, a \rangle \in \pi$ and $s \xrightarrow{a} s'$, or $s = s'$ and $s \in \textsc{Closure}(\pi) \cup G$.

Notice that all execution paths are infinite which is required in order to define solutions in CTL. If a state is reached that is not covered by the plan (e.g., a goal state or a dead end), the postfix of the execution path from this state is an infinite repetition of it. Given a problem $\mathcal{P} = \langle \mathcal{D}, s_0, G \rangle$ and a plan π for $\mathcal{D}$ we then have

- π is a weak plan iff $\mathcal{M}(\pi), s_0 \models \mathtt{EF}\, G$,
- π is a strong cyclic plan iff $\mathcal{M}(\pi), s_0 \models \mathtt{AGEF}\, G$,
- π is a strong plan iff $\mathcal{M}(\pi), s_0 \models \mathtt{AF}\, G$.

Example 2 Consider the planning problem defined in Example 1. As expected, we have

$$\begin{aligned} \mathcal{M}(\pi_w), s_0 &\models \mathtt{EF}\,\{g\}, \\ \mathcal{M}(\pi_{sc}), s_0 &\models \mathtt{AGEF}\,\{g\}, \\ \mathcal{M}(\pi_s), s_0 &\models \mathtt{AF}\,\{g\}. \end{aligned}$$

2.1 Blind Planning Algorithms

Weak, strong cyclic, and strong plans can be synthesized by a backward breadth-first search from the goal states to the initial states. The search algorithm is shown in Figure 2. The set operations can be efficiently implemented using BDDs. For a detailed description of this approach, we refer the reader to Jensen (2003a). In each iteration (l.2-7), a precomponent P_c of the plan is computed from the states C currently covered by the plan. If the precomponent is empty, a fixed point of P has been reached that does not cover the initial states and "no solution exists" is returned. Otherwise, the precomponent is added to the plan and the states in the precomponent are added to the set of covered states (l.6-7). The strong, strong cyclic, and weak planning algorithms only differ in the definition of the precomponent. Let $\textsc{PreImg}(C)$ denote the set of SAs, where the action applied in the state may lead into the set of states C

```
function NDP($s_0, G$)
1 $P \leftarrow \emptyset$; $C \leftarrow G$
2 while $s_0 \notin C$
3     $P_c \leftarrow \textsc{PreComp}(C)$
4     if $P_c = \emptyset$ then return "no solution exists"
5     else
6         $P \leftarrow P \cup P_c$
7         $C \leftarrow C \cup \textsc{States}(P_c)$
8 return $P$
```

Figure 2: A generic algorithm for synthesizing non-deterministic plans.

$$\textsc{PreImg}(C) \equiv \{\langle s, a\rangle \,:\, \textsc{Next}(s,a) \cap C \neq \emptyset\}.$$

The weak and strong precomponent are then defined by

$$\begin{aligned} \textsc{PreComp}_w(C) &\equiv \textsc{PreImg}(C) \setminus C \times \mathit{Act}, \\ \textsc{PreComp}_s(C) &\equiv (\textsc{PreImg}(C) \setminus \textsc{PreImg}(\overline{C})) \setminus C \times \mathit{Act}. \end{aligned}$$

The strong cyclic precomponent $\textsc{PreComp}_{sc}(C)$ can be computed by iteratively extending a set of candidate SAs and pruning it until a fixed point is reached. Let $\textsc{Weak}$, StrongCyclic, and Strong denote the NDP algorithm using PreComp$_w$, PreComp$_{sc}$, and PreComp$_s$, respectively. It can be shown that Weak, StrongCyclic, and Strong are correct Jensen (2003a). The algorithms return "no solution exists" iff no solution exists, otherwise they return a valid solution.

Due to the breadth-first search carried out by NDP, weak plans returned by Weak have minimum length best-case execution paths and strong plans returned by Strong have minimum length worst-case execution paths (Cimatti et al., 2003). Formally, for a non-deterministic planning domain $\mathcal{D}$ and a plan π of $\mathcal{D}$ let

$$\textsc{Exec}(s,\pi) \equiv \{q \,:\, q \text{ is a path of } \mathcal{M}(\pi) \text{ and } q_0 = s\}$$

denote the set of execution paths of π starting at s. Let the length of a path $q = q_0q_1\cdots$ with respect to a set of states C be defined by

$$|q|_C \equiv \begin{cases} i & : \text{ if } q_i \in C \text{ and } q_j \notin C \text{ for } 0 \le j < i \\ \infty & : \text{ otherwise.} \end{cases}$$

Let Min(s, C, π) and Max(s, C, π) denote the minimum and maximum length of an execution path from s to C of a plan π

$$\begin{aligned} \textsc{Min}(s,C,\pi) &\equiv \min_{q\in\textsc{Exec}(s,\pi)} |q|_C, \\ \textsc{Max}(s,C,\pi) &\equiv \max_{q\in\textsc{Exec}(s,\pi)} |q|_C. \end{aligned}$$

Similarly, let Π denote the set of all plans of $\mathcal{D}$ and let WDist(s, C) (weak distance) and SDist(s, C) (strong distance) denote the minimum of Min(s, C, π) and Max(s, C, π) for any plan $\pi \in \Pi$ of $\mathcal{D}$

$$\begin{aligned} \textsc{WDist}(s,C) &\equiv \min_{\pi\in\Pi} \textsc{Min}(s,C,\pi), \\ \textsc{SDist}(s,C) &\equiv \min_{\pi\in\Pi} \textsc{Max}(s,C,\pi). \end{aligned}$$

For a weak plan $\pi_w = \textsc{Weak}(s_0, G)$ and strong plan $\pi_s = \textsc{Strong}(s_0, G)$, we then have

$$\begin{aligned} \textsc{Min}(s_0,G,\pi_w) &= \textsc{WDist}(s_0,G), \\ \textsc{Max}(s_0,G,\pi_s) &= \textsc{SDist}(s_0,G). \end{aligned}$$

2.2 Guided Planning Algorithms

Pure heuristic non-deterministic planning algorithms can be realized by partitioning the set of state-action pairs of the precomponent according to a heuristic measure. A guided version of NDP called GNDP is shown in Figure 3. Similar to NDP, this algorithm can be efficiently implemented with BDDs using a technique called *non-deterministic state-set branching* (Jensen et al., 2003). The main difference between NDP and GNDP is that GNDP keeps partitioned precomponent and the set of states covered by the plan in the maps $\mathbf{P_c}$ and $\mathbf{C}$. The purpose of $\mathbf{C}$ is to partition the states with respect to the value of a heuristic function that for a state s estimates the minimum length of a path from s_0 to s. [1]

1. Initially the map entry, $\mathbf{C}[h_{goal}]$ is assigned to the goal states. To simplify the presentation, we assume that all goal states have identical h-value. A generalization of the algorithm is trivial.

```
function GNDP($s_0, G, h_g$)
1 $P \leftarrow \emptyset$; $\mathbf{C} \leftarrow \mathit{emptyMap}$; $\mathbf{C}[h_g] \leftarrow G$
2 while $s_0 \notin C$
3     $\mathbf{P_c} \leftarrow$ GPreComp($\mathbf{C}$)
4     if $|\mathbf{P_c}| = 0$ then return "no solution exists"
5     $P \leftarrow P \cup P_c$
6     for $k = 1$ to $|\mathbf{P_c}|$
7         $\mathbf{C}[h_k] \leftarrow \mathbf{C}[h_k] \cup$ States($\mathbf{P_c}[h_k]$)
8 return $P$
```

Figure 3: A generic guided algorithm for synthesizing non-deterministic plans.

The unbolded name of a map (e.g., P_c in Line 5) denotes the elements in the map. Thus, for a map $\mathbf{m} = \{e_1 \mapsto v_1, \ldots e_n \mapsto v_n\}$, we have $m = \{v_1, \ldots, v_n\}$. For weak and strong plans, the guided precomponent simply consists of the subset of SAs in the blind precomponent with lowest heuristic measure. Let GuidedWeak, GuidedStrongCyclic, and GuidedStrong denote GNDP using the guided weak, strong cyclic, and strong precomponent, respectively. These algorithms are correct, but GuidedWeak and GuidedStrong may not return optimal weak and strong plans due to the pure heuristic search. We refer the reader to (Jensen, 2003a) for details.

3. N-Fault Tolerant Planning Problems

A fault tolerant planning domain is a non-deterministic planning domain, where actions have primary and secondary effects. The primary effect is deterministic. However, since an action often can fail in many different ways, we allow the secondary effect to lead to one of several possible next states. Thus, secondary effects are non-deterministic.

Primary and secondary effects can either be represented explicitly in a planning domain via two separate effect descriptions or implicitly via a embedding in a non-deterministic planning domain extended with a fault counter. We call the explicit representation a *fault tolerant planning domain* while the implicit representation is called the *induced non-deterministic planning domain*. The explicit representation is suitable for planning algorithms that distinguish semantically between primary and secondary effects. This is particularly important for constructing guided planning algorithms that decouple the guidance of the planning for primary and secondary effects. The implicit representation, on the other hand, is useful for defining properties of fault tolerant plans and generating fault tolerant plans via the Strong algorithm.

Definition 1 (Fault Tolerant Planning Domain) *A fault tolerant planning domain is a tuple $\mathcal{D}^F = \langle S, Act, \rightarrow, \rightsquigarrow \rangle$ where S is a finite set of states, Act is a finite set of actions, $\rightarrow \subseteq S \times Act \times S$ is a deterministic transition relation of primary effects, and $\rightsquigarrow \subseteq S \times Act \times S$ is a non-deterministic transition relation of secondary effects. Instead of $(s, a, s') \in \rightarrow$ and $(s, a, s') \in \rightsquigarrow$, we write $s \xrightarrow{a} s'$ and $s \overset{a}{\rightsquigarrow} s'$, respectively.*

For a fault tolerant planning domain, the set of next states of an action a applied in state s is the union of next states reached by primary and secondary effects

$$\textsc{Next}^{\mathcal{F}}(s,a) \equiv \{s' \,:\, s \xrightarrow{a} s' \text{ or } s \overset{a}{\rightsquigarrow} s'\}.$$

As for non-deterministic domains, an action a is called *applicable* in state s iff $\textsc{Next}^{\mathcal{F}}(s,a) \neq \emptyset$. The set of applicable actions in a state s is given by

$$\textsc{App}^{\mathcal{F}}(s) \equiv \{a \,:\, \textsc{Next}^{\mathcal{F}}(s,a) \neq \emptyset\}.$$

A fault tolerant planning problem is given by an initial state, a set of goal states to reach, and an upper bound n on the number faults the plan must be able to handle during execution.

Definition 2 (N-Fault Tolerant Planning Problem) *An n-fault tolerant planning problem is a tuple $\mathcal{P}^{\mathcal{F}} = \langle \mathcal{D}^F, s_0, G, n\rangle$ where $\mathcal{D}^F$ is a fault tolerant planning domain, $s_0 \in S$ is an initial state, $G \subseteq S$ is a set of goal states, and $n : \mathbb{N}$ is an upper bound on the number of faults the plan must be able to recover from.*

An n-fault tolerant plan is a set of plans $F^0, \ldots, F^n$, where F^i is applied when i faults have occurred.[2]

Definition 3 (N-Fault Tolerant Plan) *An n-fault tolerant plan for the problem $\mathcal{P}^{\mathcal{F}} = \langle \mathcal{D}^{\mathcal{F}}, s_0, G, n\rangle$ is a tuple $\pi^{\mathcal{F}} = \langle F^0, \ldots, F^n\rangle$, where F^i is a a set of state action pairs of $\mathcal{D}^{\mathcal{F}}$.*

A *valid* n-fault tolerant plan guarantees that a goal state is reached in a finite number of steps from the initial state if no more than n faults occur during execution. An *optimal* n-fault tolerant plan is a valid fault tolerant plan with minimum worst case execution length. These properties can be defined formally in terms of the induced non-deterministic planning domain. The induced non-deterministic planning domain adds a fault counter f to the state description and models secondary effects only when $f \leq n$. In this way an optimal fault tolerant plan is a strong plan generated by the Strong algorithm.

Definition 4 (Induced Non-Deterministic Planning Domain) *Let $\mathcal{D}^{\mathcal{F}} = \langle S, Act, \rightarrow, \rightsquigarrow\rangle$ be a fault tolerant planning problem. The non-deterministic planning domain induced from $\mathcal{D}^{\mathcal{F}}$ is given by $\mathcal{D}^{\mathcal{I}}_n = \langle S^{\mathcal{I}}, Act^{\mathcal{I}}, \rightarrow^{\mathcal{I}}\rangle$, where*

- $S^{\mathcal{I}} = S \times \{0, \ldots, n\}$,
- $Act^{\mathcal{I}} = Act$,
- $\langle s, f\rangle \xrightarrow{a}{}^{\mathcal{I}} \langle s', f'\rangle$ *iff*
  - $s \xrightarrow{a} s'$ *and* $f' = f$, *or*
  - $s \overset{a}{\rightsquigarrow} s'$, $f < n$, *and* $f' = f + 1$.

Definition 5 (Induced Non-Deterministic Planning Problem) *Let $\mathcal{P}^{\mathcal{F}} = \langle \mathcal{D}^F, s_0, G, n\rangle$ be an n-fault tolerant planning problem. The non-deterministic planning problem induced from $\mathcal{P}^{\mathcal{F}}$ is given by $\mathcal{P}^{\mathcal{I}} = \langle \mathcal{D}^I_n, s^I_0, G^{\mathcal{I}}\rangle$, where*

2. These sub-plans can be compactly stored in a shared representation based on a multi-rooted BDD.

- $s_0^{\mathcal{I}} = \langle s_0, 0 \rangle$,
- $G^{\mathcal{I}} = G \times \{0, \ldots, n\}$.

Definition 6 (Induced Non-Deterministic Plan) *Let $\pi^{\mathcal{F}} = \langle F^0, \ldots, F^n \rangle$ be a fault tolerant plan. The non-deterministic plan induced from $\pi^{\mathcal{F}}$ is given by $\pi^{\mathcal{I}} = \cup_{i=0}^{n} \{ \langle \langle s, i \rangle, a \rangle : \langle s, a \rangle \in F_i \}$.*

Valid and optimal n-fault tolerant plans can now be formally defined in terms of their induced non-deterministic plans.

Definition 7 (Valid N-Fault Tolerant Plan) *An n-fault tolerant plan $\pi^{\mathcal{F}}$ is valid iff $\mathcal{M}(\pi^{\mathcal{I}}), s_0^{\mathcal{I}} \models \mathtt{AF}\, G^{\mathcal{I}}$.*

Definition 8 (Optimal N-Fault Tolerant Plan) *A valid n-fault tolerant plan $\pi^{\mathcal{F}}$ is optimal iff* $\text{Max}(s_0^{\mathcal{I}}, G^{\mathcal{I}}, \pi^{\mathcal{I}}) = \text{SDist}(s_0^{\mathcal{I}}, G^{\mathcal{I}})$

4. N-Fault Tolerant Planning Algorithms

Let n-FTP$_S$ denote the Strong algorithm applied to the non-deterministic planning problem induced from an n-fault tolerant planning problem. It follows directly from the definition of strong plans that n-FTP$_S$ is correct, i.e., that it returns a valid n-fault tolerant plan if such a plan it exists and otherwise returns failure. Moreover, due to the optimality of Strong, n-FTP$_S$ returns optimal n-fault tolerant plans. Since the performance of blind non-deterministic planning is limited, we also consider solving n-fault tolerant planning problems with the guided version of strong planning defined in previous section. Let n-GFTP$_S$ denote the GuidedStrong algorithm applied to the non-deterministic planning problem induced from an n-fault tolerant planning problem. Due to the pure heuristic search approach, n-GFTP$_S$ may return suboptimal solutions.

An *error state* is a state resulting from a secondary effect of an action. We call an error state *local*, if there exists a short path in the state space between the error state and the resulting state of the primary effect of the action. We may expect n-GFTP$_S$ to be efficient when error states are local because they then will be covered by the search beam of n-GFTP$_S$. In practice, however, secondary effects may be permanent malfunctions that due to their impact on the domain cause a transition to a non-local state. Indeed, in theory, the location of error states may be completely uncorrelated with the location of states of primary effect. To address this problem, we develop a specialized algorithm where the planning for primary and secondary effects is decoupled. We focus our investigation to 1-fault tolerant planning and introduce two algorithms: 1-FTP using blind search and 1-GFTP using guided search.

4.1 The 1-FTP algorithm

The 1-FTP algorithm is shown in Figure 4. The function PreImg$_f$ computes the preimage of secondary effects. 1-FTP returns the 1-fault tolerant plan $\langle F^0, F^1 \rangle$, where F^0 is robust to one fault while F^1 is a recovery plan.

```
function 1-FTP($s_0, G$)
1   $F^0 \leftarrow \emptyset$; $C^0 \leftarrow G$
2   $F^1 \leftarrow \emptyset$; $C^1 \leftarrow G$
3   while $s_0 \notin C^0$
4       $f_c^0 \leftarrow \textsc{PreImg}(C^0) \setminus C^0 \times Act$
5       $f^0 \leftarrow f_c^0 \setminus \textsc{PreImg}_f(\overline{C^1})$
6       while $f^0 = \emptyset$
7           $f^1 \leftarrow \textsc{PreImg}(C^1) \setminus C^1 \times Act$
8           if $f^1 = \emptyset$ then return "no solution exists"
9           $F^1 \leftarrow F^1 \cup f^1$
10          $C^1 \leftarrow C^1 \cup \textsc{States}(f^1)$
11          $f^0 \leftarrow f_c^0 \setminus \textsc{PreImg}_f(\overline{C^1})$
12      $F^0 \leftarrow F^0 \cup f^0$
13      $C^0 \leftarrow C^0 \cup \textsc{States}(f^0)$
14  return $\langle F^0, F^1 \rangle$
```

Figure 4: The 1-FTP algorithm.

1-FTP performs a backward search from the goal states that alternate between blindly expanding F^0 and F^1 such that failure states of F^0 always can be recovered by F^1. Initially F^0 and F^1 are assigned to empty plans (l.1-2). The variables C^0 and C^1 are states covered by the current plans in F^0 and F^1. They are initialized to the goal states since these states are covered by empty plans. In each iteration of the outer loop (l.3-13), F^0 is expanded with SAs in f^0 (l.12-13). First, a candidate f_c^0 is computed. It is the preimage of the states in F^0 pruned for SAs of states already covered by F^0 (l.4). The variable f^0 is assigned to f_c^0 restricted to SAs for which all error states are covered by the current recovery plan (l.5). If f^0 is empty the recovery plan is expanded in the inner loop until f^0 is nonempty (l.6-11). If the recovery plan at some point has reached a fixed point and f^0 is still empty, the algorithm terminates with failure, since in this case, no recovery plan exists (l.8).

Theorem 1 (Correctness of 1-FTP**)** *The 1-*FTP *planning algorithm is correct. The algorithm returns "no solution exists" iff no 1-fault tolerant plan exists, otherwise it returns a valid 1-fault tolerant plan.*

Proof. This follows from the soundness, completeness, and termination theorems of 1-FTP proved in Appendix A. □

Example 3 An example of the non-deterministic plans F^0 and F^1 returned by 1-FTP is shown in Figure 5 ◇

1-FTP expands both F^0 and F^1 blindly. An inherent strategy of the algorithm, though, is not to expand F^1 more than necessary to recover the faults of F^0. This is not the case for n-FTP$_S$ that does not distinguish states with different number of faults. The aggressive strategy of 1-FTP, however, makes it suboptimal as the example in Figure 6 shows. In

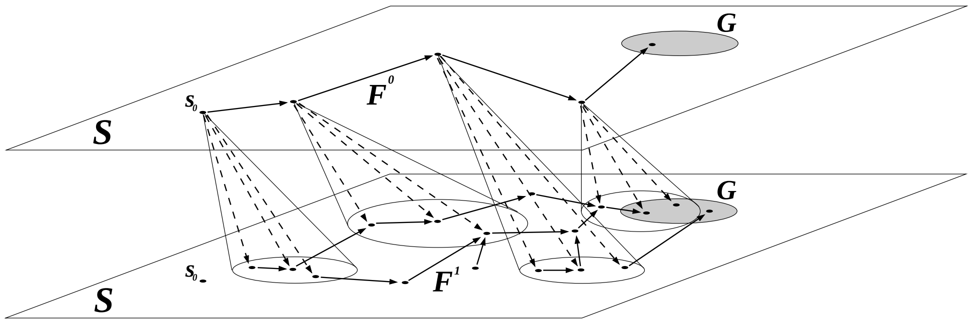

Figure 5: An example of a 1-fault tolerant plan $\langle F^0, F^1 \rangle$ returned by 1-FTP. Primary and secondary effects of actions are drawn with solid and dashed lines, respectively. In this example, we assume that F^0 forms a sequence of actions from the initial state to a goal state, while F^1 recovers all the possible faults of actions in F^0.

the first two iterations of the outer loop, $\langle p_2, b \rangle$ and $\langle p_1, b \rangle$ are added to F^0 and nothing is added to F^1. In the third iteration of the outer loop, F^1 is extended with $\langle p_2, b \rangle$ and $\langle q_2, a \rangle$ and F^0 is extended with $\langle q_2, a \rangle$. In the last two iterations of the outer loop, $\langle q_1, a \rangle$ and $\langle s_0, a \rangle$ are added to F^0. The resulting plan is

$$\begin{aligned} F^0 &= \{\langle s_0, a \rangle, \langle q_1, a \rangle, \langle q_2, a \rangle, \langle p_1, b \rangle, \langle p_2, b \rangle\} \\ F^1 &= \{\langle p_2, b \rangle, \langle q_2, a \rangle\}. \end{aligned}$$

The worst case length of this 1-fault tolerant plan is 4. However, a 1-fault tolerant plan

$$\begin{aligned} F^0 &= \{\langle s_0, b \rangle, \langle p_1, b \rangle, \langle p_2, b \rangle\} \\ F^1 &= \{\langle q_1, a \rangle, \langle q_2, a \rangle\} \end{aligned}$$

with worst case length of 3 exists.

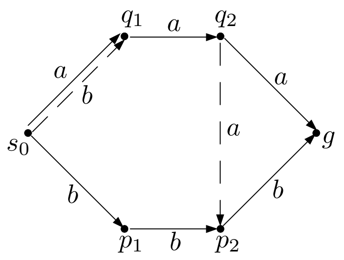

Figure 6: A problem with a single goal state g showing that 1-FTP may return suboptimal solutions. Dashed lines indicate secondary effects. Notice that action a and b only have secondary effects in q_2 and s_0, respectively. In all other states, the actions are assumed always to succeed.

4.2 The 1-GFTP Algorithm

Despite the different search strategies applied by 1-FTP and 1-FTP$_S$, they both perform blind search. A more interesting algorithm is a guided version of 1-FTP called 1-GFTP that similar to GNDP is based on non-deterministic state-set branching. The over all design goal of 1-GFTP is to guide the expansion of F^0 toward the initial state and guide the expansion of F^1 toward the failure states of F^0. However, this can be accomplished in many different ways. Below we evaluate three different strategies. For each algorithm, F^0 is guided in a pure heuristic manner toward the initial state using the approach employed by n-GFTP$_S$ while the guiding approach used for F^1 varies.

The first strategy is to assume that failure states are local and guide F^1 toward the initial state as well. The resulting algorithm is similar to 1-GFTP$_S$ and has poor performance. The problem is that the pure heuristic approach causes F^1 only to cover a narrow beam of states in the state space. Error states not within close distance to the primary effects tend not to be covered by F^1. The strategy can be improved by widening the beam by taking the search depth into account. However, this does not provide a satisfactory solution for non-local states.

The second strategy is ideal in the sense that it dynamically guides the expansion of F^1 toward error states of the precomponents of F^0. This can be done by using a specialized BDD operation that splits the precomponent of F^1 according to the Hamming distance to the error states. The complexity of this operation, however, is exponential in the size of the BDD representing the error states and the size of the BDD representing the precomponent of F^0. Due to the dynamic programming used by the BDD package, the average complexity may be much lower. However, this algorithm is often intractable in practice.

The third strategy is the one chosen for 1-GFTP. It expands F^1 blindly but then prunes SAs from the precomponent of F^1 not used to recover error states of F^0. Thus, it uses an indirect approach to guide the expansion of F^1. We expect this strategy to work well even if the *absolute position* of error states is non-local. However, the strategy assumes that the *relative position* of error states is local in the sense that the SAs in F^1 in expansion i of F^0 are relevant for recovering error states in expansion $i + 1$ of F^0. In addition, we still have an essential problem to solve: to expand F^0 or F^1. There are two extremes.

1. *Expand F^1 until first recovery of f^0*: compute a complete partitioned backward precomponent of F^0, expand F^1 until some partition in f^0 has recovered error states and add the partition with least h-value to F^0.

2. *Expand F^1 until best recovery of f^0*: compute a complete partitioned backward precomponent of F^0, expand F^1 until the partition of f^0 with lowest h-value has recovered error states and add this partition to F^0. If none of these error states can be recovered then consider the partition with second lowest h-value and so on.

It turns out that neither of these extremes work well in practice. The first is too conservative. It may add a partition with a high h-value even though a partition with a low h-value can be recovered given just a few more expansions of F^1. The second strategy is too greedy. It ignores the complexity of expanding F^1 in order to recover error states of the partition of f^0 with lowest h-value. Instead, we consider a mixed strategy: spend half of the last expansion time on recovering error states of the partition of f^0 with lowest h-value and, in case this is impossible, spend one fourth of the last expansion time on recovering error states of the partition of f^0 with second lowest h-value, and so on.

The 1-GFTP algorithm is shown in Figure 7. The keys in maps are sorted ascendingly.

```
function 1-GFTP($s_0, G$)
1  $F^0 \leftarrow \emptyset$; $\mathbf{C}^0[h_g] \leftarrow G$
2  $F^1 \leftarrow \emptyset$; $C^1 \leftarrow G$
3  $t \leftarrow \epsilon$
4  while $s_0 \notin C^0$
5     $t_s \leftarrow t_{CPU}$
6     $\mathbf{PC} \leftarrow$ PreCompFTP($\mathbf{C}^0$)
7     $f^0 \leftarrow \emptyset$; $f_c^0 \leftarrow \emptyset$
8     $\mathbf{f}_c^1 \leftarrow$ emptyMap
9     $i \leftarrow 0$
10    while $f^0 = \emptyset \wedge i < |\mathbf{PC}|$
11       $i \leftarrow i + 1$; $t \leftarrow t/2$
12       $f_c^0 \leftarrow f_c^0 \cup \mathbf{PC}[i]$
13       $\langle \mathbf{f}_c^1, f^0 \rangle \leftarrow$ ExpandTimed($f_c^0, \mathbf{f}_c^1, C^1, t$)
14    if $f^0 = \emptyset$ then
15       $\langle \mathbf{f}_c^1, f^0 \rangle \leftarrow$ ExpandTimed($f_c^0, \mathbf{f}_c^1, C^1, \infty$)
16    $t \leftarrow t_{CPU} - t_s$
17    if $f^0 = \emptyset$ then return "no solution exists"
18    $f^1 \leftarrow$ PruneUnused($\mathbf{f}_c^1, f^0$)
19    $F^1 \leftarrow F^1 \cup f^1$; $C^1 \leftarrow C^1 \cup$ States($f^1$)
20    $F^0 \leftarrow F^0 \cup f^0$
21    for $i = 1$ to $|\mathbf{PC}|$
22       $\mathbf{C}^0[h_i] \leftarrow \mathbf{C}^0[h_i] \cup$ States($f^0 \cap \mathbf{PC}[h_i]$)
23 return $\langle F^0, F^1 \rangle$
```

Figure 7: The 1-GFTP algorithm.

The instantiation of F^0 and F^1 of 1-GFTP is similar to 1-FTP except that the states in C^0 are partitioned with respect to their associated h-value. Initially the map entry, $\mathbf{C}^0[h_{goal}]$ is assigned to the goal states which are assumed to have identical h-value. [3] The variable t stores the duration of the previous expansion. Initially, it is given a small value ϵ. In each iteration of the main loop (l.4-22), the precomponents f^0 and f^1 are computed and added to F^0 and F^1. First, the start time t_s is logged by reading the current time t_{CPU} (l.5). Then a map $\mathbf{PC}$ holding a complete partitioned precomponent candidate of F^0 is computed by PreCompFTP [4] (l.6). For each entry in $\mathbf{C}^0$, PreCompFTP inserts the sub-precomponent in $\mathbf{PC}$ of each partition of a partitioning of the transition relation of primary effects. We assume that this partitioning has m subrelations $\rightarrow_1, \ldots, \rightarrow_m$ where the transitions in sub-relation by $\rightarrow_j$ are associated with a change δh_j of the h-value (in forward direction). The inner loop (l.10-13) of 1-GFTP expands the two candidates f_c^0

3. As for GNDP, a generalization of the algorithm is trivial.
4. Recall that the unbolded name of a map (e.g., C^0 in l.4) denotes the elements in the map.

and f_c^1 for f^0 and f^1. In each iteration, a partition of the partitioned precomponent **PC** is added to f_c^0 (l.12).[5] The function ExpandTimed expands f_c^1. In iteration i, the time out bound of the expansion is $t/2^i$. The states *recovS* is the union of the states in f_c^1 and F^1. Hence, $\text{PreImg}_f(\overline{recovS})$ is the set of SAs with error states outside of *recovS* which are subtracted from the SAs in f_c^0 (l.5 and l.11). ExpandTimed returns early if

1. a precomponent f^0 in the candidate f_c^0 is found where all error states are recovered, or
2. f_c^1 has reached a fixed point.

The sub-precomponent added to f_c^1 in iteration i of ExpandTimed is stored in the map entry $\mathbf{f}_c^1[i]$ in order to later prune SAs not used for recovery.

```
function PreCompFTP($\mathbf{C}^0$)
1  $\mathbf{PC} \leftarrow emptyMap$
2  for $i = 1$ to $|\mathbf{C}^0|$
3     for $j = 1$ to $m$
4        $SA \leftarrow \text{PreImg}_j(\mathbf{C}^0[h_i]) \setminus C^0 \times Act$
5        $\mathbf{PC}[h_i - \delta h_j] \leftarrow \mathbf{PC}[h_i - \delta h_j] \cup SA$
6  return PC
```

Eventually f_c^0 may contain all the SAs in **PC** without any of these being recoverable. In this case 1-GFTP expands f_c^1 (l.15) untimed.

```
function ExpandTimed($f_c^0, \mathbf{f}_c^1, C^1, t$)
1  $t_s \leftarrow t_{CPU}$
2  $Oldf_c^1 \leftarrow \perp$
3  $i \leftarrow |\mathbf{f}_c^1|$
4  $recovS \leftarrow \text{States}(f_c^1) \cup C^1$
5  $f^0 \leftarrow f_c^0 \setminus \text{PreImg}_f(\overline{recovS})$
6  while $f^0 = \emptyset \wedge Oldf_c^1 \neq f_c^1 \wedge t_{CPU} - t_s < t$
7     $Oldf_c^1 \leftarrow f_c^1$
8     $i \leftarrow i + 1$
9     $\mathbf{f}_c^1[i] \leftarrow \text{PreImg}(recovS) \setminus recovS \times Act$
10    $recovS \leftarrow \text{States}(f_c^1) \cup C^1$
11    $f^0 \leftarrow f_c^0 \setminus \text{PreImg}_f(\overline{recovS})$
12 return $\langle \mathbf{f}_c^1, f^0 \rangle$
```

If f_c^1 has reached a fixed point but no recoverable precomponent f^0 exists, no 1-fault tolerant plan exists and 1-GFTP returns with failure (l.17). Otherwise, f_c^1 is pruned for SAs of states not used to recover the SAs in f^0 (l.18). This pruning is computed by PruneUnused that traverses backward through the sub-precomponents of $\mathbf{f}_c^1$ and marks states that either are error states of SAs in f^0, or states needed to recover error states.

5. Recall that **PC** is traversed ascendingly such that the partition with lowest h-value is added first.

```
function PruneUnused($\mathbf{f}_c^1, f^0$)
1 $err \leftarrow \mathrm{SAimg}_f(f^0)$
2 $img \leftarrow \emptyset$;
3 for $i = |\mathbf{f}_c^1|$ to 1
4     $\mathbf{f}_c^1[i] \leftarrow \mathbf{f}_c^1[i] \cap \big((err \cup img) \times Act\big)$
5     $img \leftarrow \mathrm{SAimg}(\mathbf{f}_c^1[i])$
6 return $f_c^1$
```

The function $\mathrm{SAimg}(\pi)$ and $\mathrm{SAimg}_f(\pi)$ computes the image states of a set of SAs π for primary and secondary effects respectively.

$$\mathrm{SAimg}(\pi) \equiv \{s' : \exists \langle s, a\rangle \in \pi \,.\, s \overset{a}{\rightarrow} s'\} \quad (1)$$
$$\mathrm{SAimg}_f(\pi) \equiv \{s' : \exists \langle s, a\rangle \in \pi \,.\, s \overset{a}{\rightsquigarrow} s'\} \quad (2)$$

The updating of F^0 and F^1 of 1-GFTP (l.19-22) is similar to 1-FTP, except that $\mathbf{C}^0$ is updated by iterating over **PC** and picking SAs in f^0. Notice that in this iteration h_i refers to the keys of **PC**.

Theorem 2 (Correctness of 1-GFTP**)** *The 1-*GFTP *planning algorithm is correct. The algorithm returns "no solution exists" iff no 1-fault tolerant plan exists, otherwise it returns a valid 1-fault tolerant plan.*

Proof. This follows from the soundness, completeness, and termination theorems of 1-GFTP proved in Appendix A. □

The specialized algorithms can be generalized to n faults by adding more recovery plans $F^n, F^{n-1}, \ldots, F^0$. For n-GFTP all of these recovery plans would be indirectly guided by the expansion of F^n. The algorithm is illustrated in Figure 8.

5. Experimental Evaluation

The experimental evaluation has two major objectives: to get a better intuition about the nature of fault tolerant plans and to compare the performance of the developed algorithms. 1-FTP, 1-GFTP, 1-FTP$_S$, and 1-GFTP$_S$ have been implemented in the BDD-based BIFROST 0.7 search engine (Jensen, 2003b). The domains are defined in an extended version of the Non-deterministic Agent Domain Language (NADL) (Jensen & Veloso, 2000) called NADL$^+$.

All experiments have been executed on a Redhat Linux 7.1 PC with kernel 2.4.16, 500 MHz Pentium III CPU, 512 KB L2 cache and 512 MB RAM. Since the number of allocated BDD nodes in the unique table (n) and the number of allocated BDD nodes in the operator caches (c) of the BuDDy[6] BDD package (Lind-Nielsen, 1999) may cause an exponential performance difference of BIFROST, we state the settings of these parameters for each experiment. In general, the sizes of the operator caches and the unique table are adjusted

6. Comparison experiments with the CUDD package (Somenzi, 1996) has not shown a significant performance difference (Jensen, 2002).

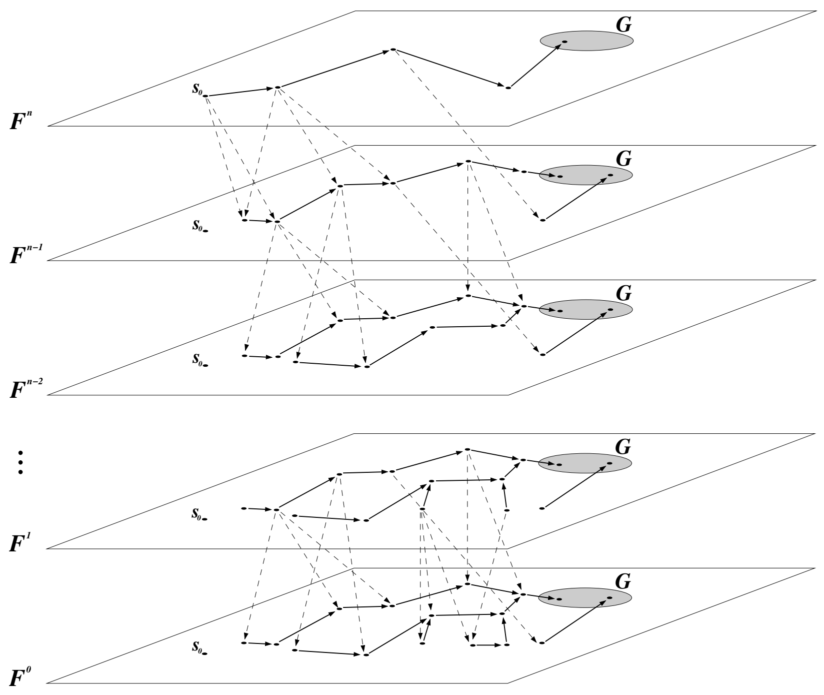

Figure 8: An example of $F^n, \ldots, F^0$ produced by a specialized n-fault tolerant planning algorithm. Primary and secondary effects of actions are drawn with solid and dashed lines, respectively.

to fit the memory requirements of the most demanding algorithm in an experiment. Thus, performance differences between algorithms are not due to relative differences in cache misses or page faults.

5.1 Unguided Search

We first focus on unguided search and study four fault tolerant planning domains. Two of these, DS1 and PSR, are models of real-world domains.

DS1

DS1 is based on an SMV encoding (Pecheur & Simmons, 2000) of the Livingstone model (Williams & Nayak, 1996) used by the Remote Agent for NASA's Deep Space One probe. The Livingstone model describes the electrical system of the spacecraft. It consists of a system bus and a number of units connected to the bus. These units include a power distribution subsystem, a Ion Propulsion System (IPS), Propulsion Drive Electronics (PDE), a Reaction Control System (RCS), Attitude Control System (ACS), Star Tracker Unit (SRU), and a MICAS camera. We recast the SMV encoding as a fault tolerant planning problem. Each bus-command is an action. The primary effect of the command is the changes it causes on the electrical system given that all units work correct. The secondary effect of an action is that one of the two faults F2 and F4 considered in the Remote Agent Experiment happens (Muscettola, Nayak, Pell, & Williams, 1998).

F2 : camera or pasm switch is recoverably stuck on/off.

F4 : an x-z thruster valve is permanently stuck closed.

In addition to these two faults, the Remote Agent Experiment considered two other malfunctions. We are not modeling these, since no 1-fault tolerant plan exists when taking all four faults into account. The following simplifications have been made in the NADL$^+$ model of the SMV description

1. we assume that the state of components is known,
2. attitude errors are assumed to be deterministically computable,
3. relative thrust is assumed to be low or nominal if a valve is stuck otherwise nominal,
4. redundant state variables in the SMV model have been removed. [7]

The NADL$^+$ encoding of the domain has 84 Boolean state variables. We consider generating a 1-fault tolerant plan from an initial state where the IPS is in standby mode, the MICAS camera is "off", and the pasm switch is "on". The goal is to reach a state where the IPS is in thrusting mode, the MICAS camera is "on", and the pasm switch is "off". The BDD package parameters are $n = 1\text{M}$ and $c = 100\text{K}$. The threshold for merging partitions of a transition relation partitioning is 5000. The total size of the transition relation is 104881 and is computed in 0.42 seconds. The size of the solution is 535 and the total CPU time is 1.15 seconds.

The experiment shows that a BDD encoding is very efficient for the kind of constraints modeled by DS1. Despite a fairly large and dense model, a transition relation is fast to compute. In addition, a 1-fault tolerant plan for a non-trivial problem in this domain is small and can be generated in less than a second. The experiment demonstrates that BDD-based fault tolerant planning is mature to be applied on significant real-world problems.

Regarding the nature of fault tolerant plans, the DS1 experiment shows that even 1-fault tolerant plans require significant redundancy of the controlled system. No 1-fault tolerant plan exists for the problem if all of the original four failures are taken into account. This result is encouraging since it shows that fault tolerant plans are substantially stricter than weak plans and the existence of fault tolerant plans is connected with the redundancy characteristics of the modeled system in a natural way.

PSR

The Power Supply Restoration domain (PSR) is a network of electric lines connected via switching devices (SDs), and fed via circuit-breakers (CBs). Switching devices and circuit breakers can either be open or closed. A circuit-breaker supplies power, when it is closed, and a switching device stops the power propagation if it is open. Consumers may be located

7. An automatic approach for removing redundant state variables doing has been developed in (Yang, Simmons, Bryant, & O'Hallaron, 1999).

on any line and are supplied only when the line is supplied. We assume that each closed circuit-breaker forms a feeder. A feeder is a tree consisting of closed switching devices and lines reachable downstream from the circuit breaker. The leafs are open switching devices and dead end lines. The "simple" PSR domain investigated in (Bertoli, Cimatti, Slanley, & Thiébaux, 2002) is shown in Figure 9. In the depicted configuration, it only has a single feeder rooted in CB_2. In the original definition of PSR domains, each unit in

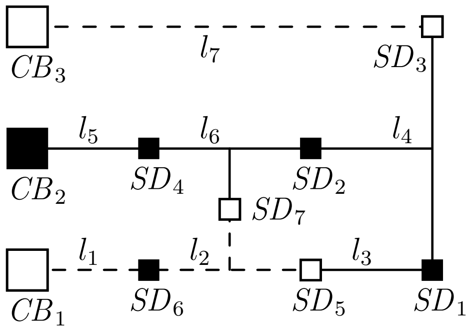

Figure 9: The "simple" PSR domain studied in (Bertoli et al., 2002). A filled box denote that the associated circuit-breaker or switching device is closed. Supplied and unsupplied lines are drawn solid and dashed, respectively.

the system may fail. Lines may short circuit, and switches may get stuck in one of their two positions. In addition, states are assumed only to be partially observable. We consider a simplified version of the domain where states are fully observable and lines do not fail. The actions of the simplified domain is to open and close switching devises and circuit breakers. The primary effect of actions is that they open and close their associated units. The secondary effect is that the units break permanently and get stuck in their current position. A specialized linear version of the domain shown in Figure 10 with n ranging from 5 to 35.

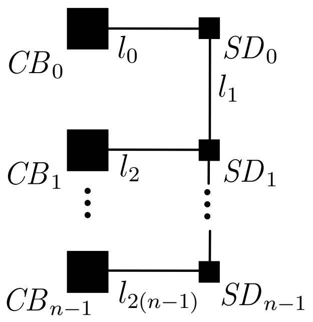

Figure 10: The linear PSR networks used for experiments.

We compare the performance of 1-FTP and 1-FTP$_S$. In the initial state, all switches are open and the goal is to feed all lines. 1-FTP and 1-FTP$_S$ solve the simple network in 6.8 and 11.25 seconds, respectively (0.98 seconds is used on memory allocation, $n = 1$M and

$c = 700\text{K}$). The results of the linear network are shown in Figure 11. The BDD package

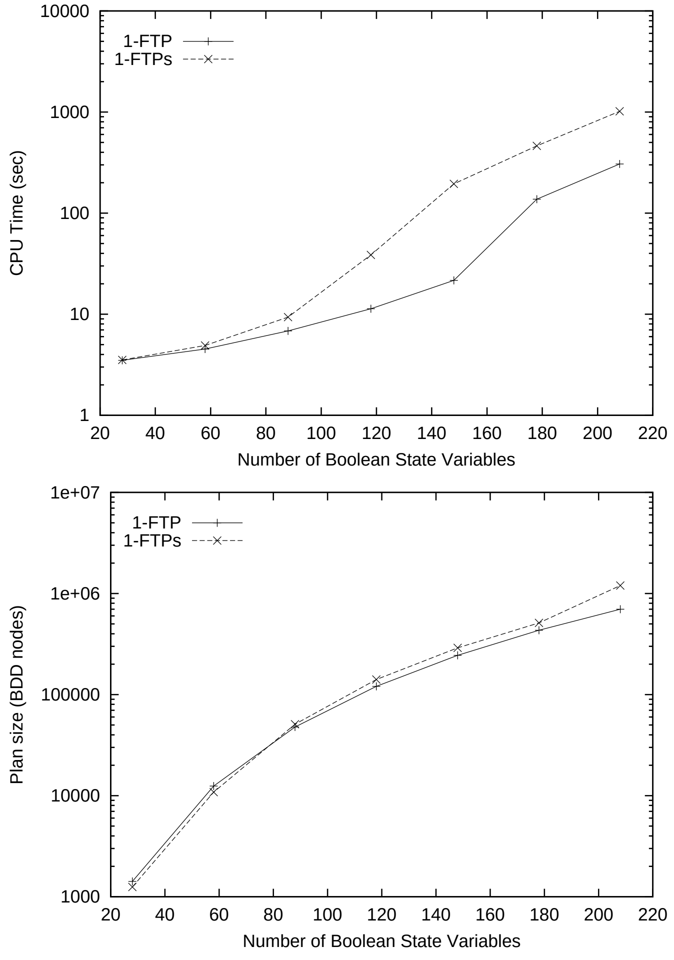

Figure 11: Results of the PSR problems.

parameters are $n = 15M$ and $c = 500K$ and 3.38 seconds are used on memory allocation. 1-FTP performs significantly better than 1-FTP$_S$ on this problem. Interestingly, the performance difference is not reflected by the plan sizes. However, this may be an artifact caused by the fact that the plan size for 1-FTP is a sum of the size of two BDDs, while the plan size for 1-FTP$_S$ is the size of a single BDD. Similarly to the DS1 domain, 1-fault tolerance imposes a strong constraint on the PSR domain. For most configurations, where a few units already have failed, no 1-fault tolerant plan exists.

Power Plant

The power plant domain is shown in Figure 12 and originates in (Jensen & Veloso, 2000). The task is to execute the control actions in order to bring the plant from some bad state, where the plant is unsafe or not working properly, to some good state, where the plant satisfies its safety and activity requirements. A single reactor R is surrounded by four

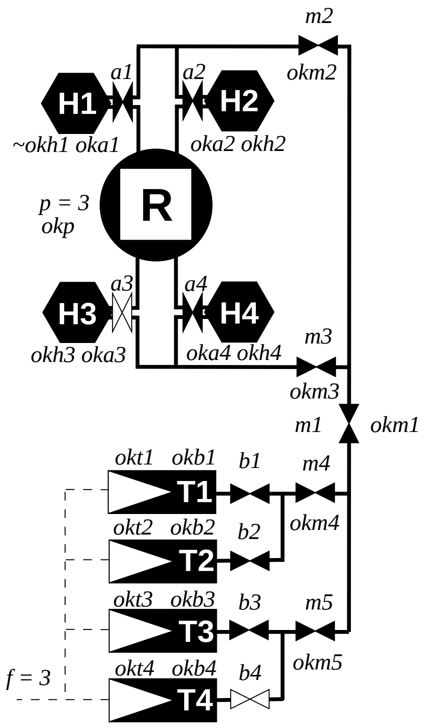

Figure 12: The power plant domain. An open valve is drawn solid and allows water or steam to flow through it. In the depicted state, a failure of heat exchanger 1 is assumed just to have happened.

heat exchangers H1, H2, H3 and H4. The heat exchangers produce high pressure steam to the four electricity generating turbines T1, T2, T3 and T4. The heat exchangers can fail and leak radioactive substances from the internal water loop to the external steam loop. If this happens, the blocking valve ($a1$, $a2$, $a3$ or $a4$) of the heat exchanger must be closed. However, these valves can fail too, in which case the valves $m2$, $m3$ or $m1$ are used. Similarly, if turbines fail, they must be shut down by closing one of the valves $b1$, $b2$, $b3$ or $b4$, or $m4$, $m5$ and $m1$. The energy production p of the plant can either be 0,1,2,3 or 4 units of energy per time unit. The production must be adjusted to fit the demand f, if possible. A heat exchanger can only transfer enough energy to a single turbine, and a single turbine can only produce one unit of energy per time unit. The initial state is shown in Figure 12. A failure of heat exchanger 1 is a assumed to have just happened.

We compare the performance of 1-FTP and 1-FTP$_S$ in two versions of the domain. The first considers controlling a single power plant. The second considers controlling two power plants simultaneously. The results are shown in Figure 13. In both experiments, the parameters of the BDD package are $n = 15M$ and $c = 500K$. The time spent on memory allocation is 3.4 seconds. 1-FTP has a slightly better performance than 1-FTP$_S$. However, both algorithms suffer from a large growth rate of the BDDs representing the frontier of the backward search. Again, 1-fault tolerant plans turns out to be surprisingly

	1-FTP		1-FTP$_S$	
size	t_{total}	$\|sol\|$	t_{total}	$\|sol\|$
40	6.1	65K	8.7	62K
80	157.8	1.2M	189.4	1.5M

Figure 13: Results of the power plant experiment. The total CPU time and plan size is given by t_{total} and $|sol|$, respectively. The size of the problem is the number of Boolean state variables.

strict. Even though the system is highly redundant, 1-fault tolerant plans only exist for simple malfunctions like the one investigated in this experiment.

Beam Walk

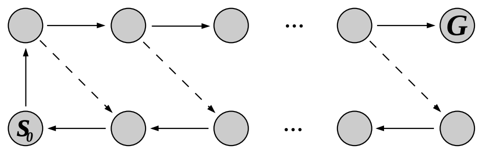

Figure 14: The Beam Walk domain. Solid edges denote primary effects of the move action, while dashed edges denote secondary effects.

The Beam Walk domain was introduced in (Cimatti, Roveri, & Traverso, 1998) and consists of a robot walking on a beam. The primary effect of the move action is that the robot moves one step forward on the beam. The secondary effect is that it falls down from the beam. The domain is shown in Figure 14. The Beam Walk domain represents a worst case scenario for 1-FTP and 1-FTP$_S$ since a fault in the last step to reach the goal causes a transition to the state furthest away from the goal. Both algorithms must iterate over all states before a solution is found. The results are shown in Figure 15. As expected, both algorithms have a limited performance in this domain. Again, however, we observe a slightly better performance of 1-FTP.

5.2 Guided Search

The main purpose of the experiments in this section is to study the difference between 1-GFTP and 1-GFTP$_S$. In particular, we are interested in investigating how sensitive these algorithms are to non-local error states and to what extent we may expect this to be a problem in practice. We study 3 domains, of which SIDMAR descends from a real-world study.

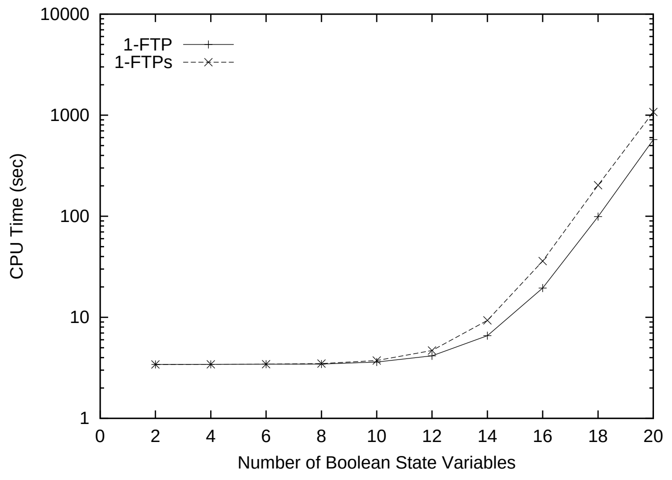

Figure 15: Results of the BeamWalk experiments.

LV

The LV domain is an artificial domain and has been designed to demonstrate the different properties of 1-GFTP and 1-GFTP$_S$. It is an $m \times m$ grid world with initial state $(0, m-1)$ and goal state $(\lfloor m/2 \rfloor, \lfloor m/2 \rfloor)$. The actions are Up, Down, Left, and Right. Above the $y = x$ line, actions may fail causing the x and y position to be swapped. Thus, error states are mirrored in the $y = x$ line. A 9×9 instance of the problem is shown in Figure 16. The

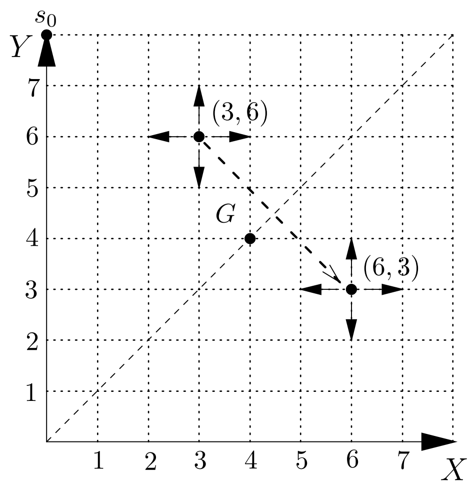

Figure 16: The 9×9 instance of the LV domain.

essential property is that error states are non-local, but that two states close to each other also have error states close to each other. This is the assumption made by 1-GFTP, but not 1-GFTP$_S$ that requires error states to be local. The heuristic value of a state is the Manhattan distance to the initial state. The BDD package parameters are $n = 5M$ and

$c = 500K$. Memory allocation takes 1.4 seconds. The results are shown in Figure 17. As

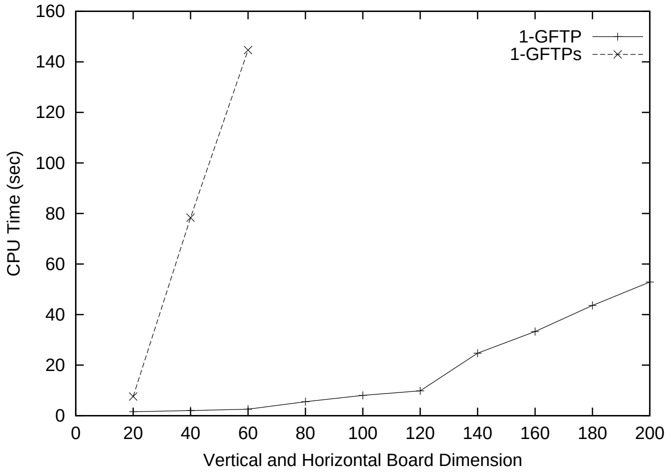

Figure 17: Results of the LV experiments.

depicted, the performance of 1-GFTP$_S$ degrades very fast with m due to the misguidance of the heuristic for the recovery part of the plan. Its total CPU time is more than 500 seconds after the first three experiments. 1-GFTP$_S$ is fairly unaffected by the error states. To explain this, consider how the backward search proceeds from the goal state. The guided precomponents of F^0 will cause this plan to beam out toward the initial state. Due to the relative locality of error states, the pruning of F^1 will cause F^1 to beam out in the opposite direction. Thus, both F^0 and F^1 remain small during the search.

Non-Deterministic 8-Puzzle

The 8-Puzzle consists of a 3×3 board with 8 numbered tiles and a blank space. A tile adjacent to the blank space can slide into the space. The goal is to reach a configuration, where the tiles are ordered ascendingly left to right, top to bottom. We consider a non-deterministic version of the 8-Puzzle, where the secondary effects are self loops as shown in Figure 18. Thus, error states are the most local possible. We use the usual sum of Manhattan distances of tiles as a heuristic for the distance to the initial state.

The experiment compares the performance of 1-FTP, 1-GFTP, 1-FTP$_S$, and 1-GFTP$_S$. The BDD package parameters are $n = 1M$ and $c = 100K$. Memory allocation takes 0.29 seconds. The number of Boolean state variables is 35 in all experiments. The results are shown in Figure 19. The results of the 8-Puzzle experiment further demonstrate the difference between 1-GFTP and 1-GFTP$_S$. Again, 1-FTP performs substantially better than 1-FTP$_S$. The guided algorithms 1-GFTP and 1-GFTP$_S$ have much better performance than the unguided algorithms. Due to local error states, however, there is no substantial performance difference between these two algorithms. As depicted, 1-FTP is slightly faster than 1-GFTP$_S$ in the experiment with a minimum deterministic solution length of 14. For such small problems, we may expect to see this since 1-FTP only expands the recovery plan when needed while 1-GFTP$_S$ expands the recovery part of its plan in each iteration.

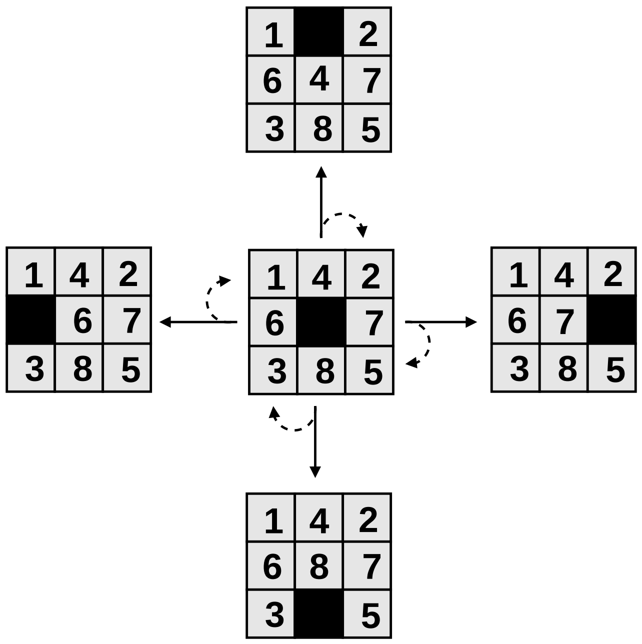

Figure 18: Primary (solid) and secondary (dashed) effects of the non-deterministic 8-Puzzle domain.

SIDMAR

The final experiments are on the SIDMAR domain. The purpose of these experiments is to study the robustness of 1-GFTP and 1-GFTP$_S$ to the kind of errors found in real-world domains. The SIDMAR domain is an abstract model of a real-world steel producing plant in Ghent, Belgium used as an ESPRIT case study (Fehnker, 1999). The layout of the steel plant is shown in Figure 20. The goal is to cast steel of different qualities. Pig iron is poured portion-wise in ladles by the two converter vessels. The ladles can move autonomously on the two east-west tracks. However, two ladles can not pass each other and there can at most be one ladle between machines. Ladles are moved in the north-south direction by the two overhead cranes. The pig iron must be treated differently to obtain steel of different qualities. There are three different treatments: 1) machine 1 and 4, 2) machine 2 and 5, and 3) machine 3. Before empty ladles are moved to the storage place, the steel is cast by the continuous casting machine. A ladle can only leave the casting machine, if there already is a filled ladle at the holding place. We assume that actions of machine 1,2,4, and 5 and move actions on the track may fail. The secondary effect of move actions is that nothing happens for the particular move. Later moves, however, may still succeed. The secondary effect of machine actions is that no treatment is carried out, and the machine is broken down permanently.

We consider casting two ladles of steel. The heuristic is the sum of machine treatments carried out on the ladles. The experiment compares the performance of 1-FTP, 1-GFTP, 1-FTP$_S$, and 1-GFTP$_S$. The BDD package parameters are $n = 5M$ and $c = 500K$.

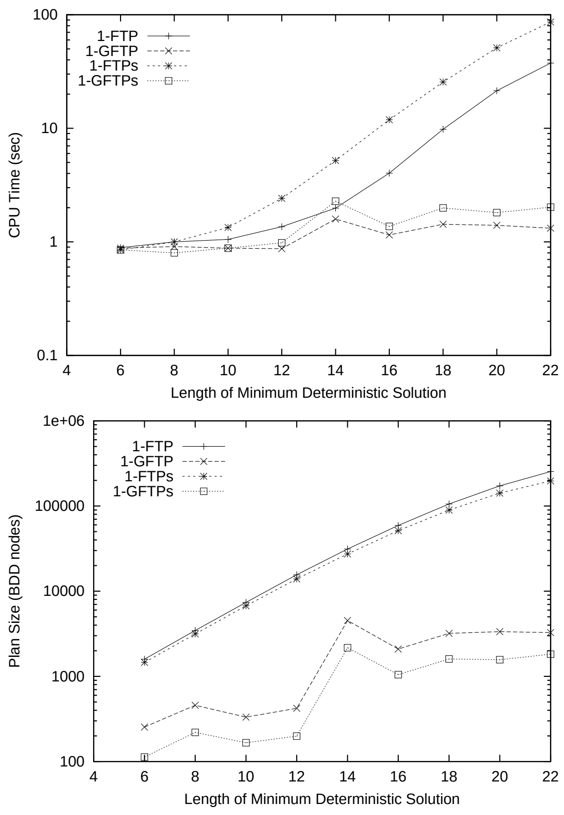

Figure 19: Results of the 8-Puzzle experiments.

Memory allocation takes 1.41 seconds. The number of Boolean state variables is 47 in all experiments. The results are shown in Figure 21. Missing data points indicates that the associated algorithm spent more than 500 seconds trying to solve the problem. The only algorithm with good performance is 1-GFTP. The experiment indicates that real-world domains may have non-local error states that limits the performance of 1-GFTP$_s$. Also notice that this is the only domain where 1-FTP does not outperform 1-FTP$_S$. In this domain, 1-FTP seems to be finding complex plans that fulfills that the recovery plan is minimum. Thus, the strategy of 1-FTP to keep the recovery plan as small as possible does not seem to be an advantage in general.

6. Related Work

Disjunctive action effects in conditional and symbolic non-deterministic planning are often caused by action failures (e.g., Peot & Smith, 1992; Weld, Anderson, & Smith, 1998; Cimatti et al., 2003). This is also the case for the large body of work in AI on fault diagnosis (e.g., Kleer & Williams, 1987; Hammond, 1990; Senjen & De Beler, 1993; Doyle, 1995). However, previous work explicitly representing and reasoning about success and failure effects of actions is very limited. The Elmer system (McCalla & Ward, 1982) uses

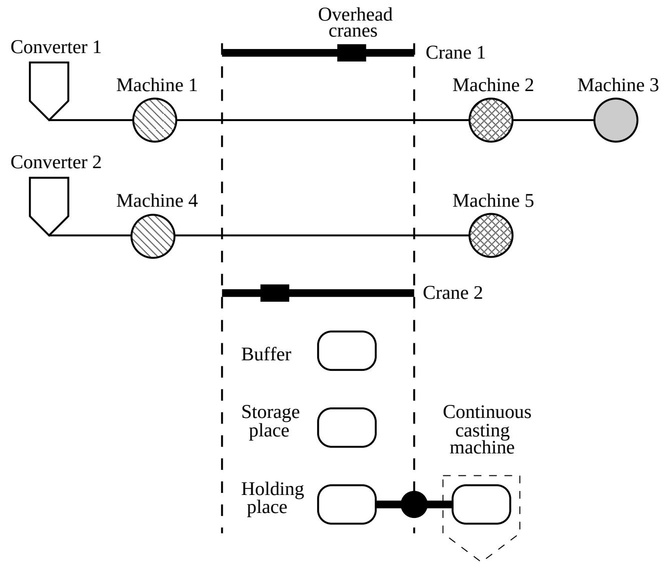

Figure 20: Layout of the SIDMAR steel plant.

error transitions from abstract actions to detect and recover from failures. In the Procedural Reasoning System (PSR, Georgeff & Lansky, 1986), the procedure descriptions defines the effect of successful and unsuccessful execution of a procedure. Similarly, the Reactive Model Based Programming Language (RMPL, Williams, Ingham, Chung, & Elliott, 2003) and its underlying executor Titan can handle faults at runtime. The approach, however, does not involve computing a fault tolerant plan. The `MRG` planning language (Giunchiglia, Spalazzi, & Traverso, 1994) explicitly models failure effects. However, this work does not include planning algorithms for generating fault tolerant plans. To our knowledge, the n-fault tolerant planning algorithms introduced in this article are the first automated planning algorithms for generating fault tolerant plans given a description of the domain that explicitly represents failure effects of actions.

There has also been a large amount of work on fault diagnosis in Discrete Event System (DES) control theory. This work has mainly focused on analyzing event sequences in order to determine if a fault has happened, and if so, which kind of fault (Sampath, Sengupta, Lafortune, Sinnamohideen, & Teneketzis, 1995, 1996; Sampath, Sengupta, Lafortune, & Teneketzis, 1998; Su, 2001). However, there has also been a considerable amount of work on fault models. These models can be characterized as either *transition based* or *state based.* Most work (e.g., Chen & Patton, 1999; Cho & Lim, 1998; Cin, 1997) use the transition based model and regard *faults* as unexpected changes in a system that tends to degrade the overall system performance rather than causing a total breakdown. The term *failure* suggests a complete breakdown of a system component or function. The transition based model is also used in supervisory control (Ramadge & Wonham, 1987) where faults usually are considered uncontrollable events (Balemi, Hoffmann, Gyugyi, Wong-Toi, & Franklin, 1993; Cho & Lim, 1998). Within this frame, an approach to fault tolerant control has been considered that is closely related to n-fault tolerant planning. Perraju, Rana, and Sarkar (1997) specify

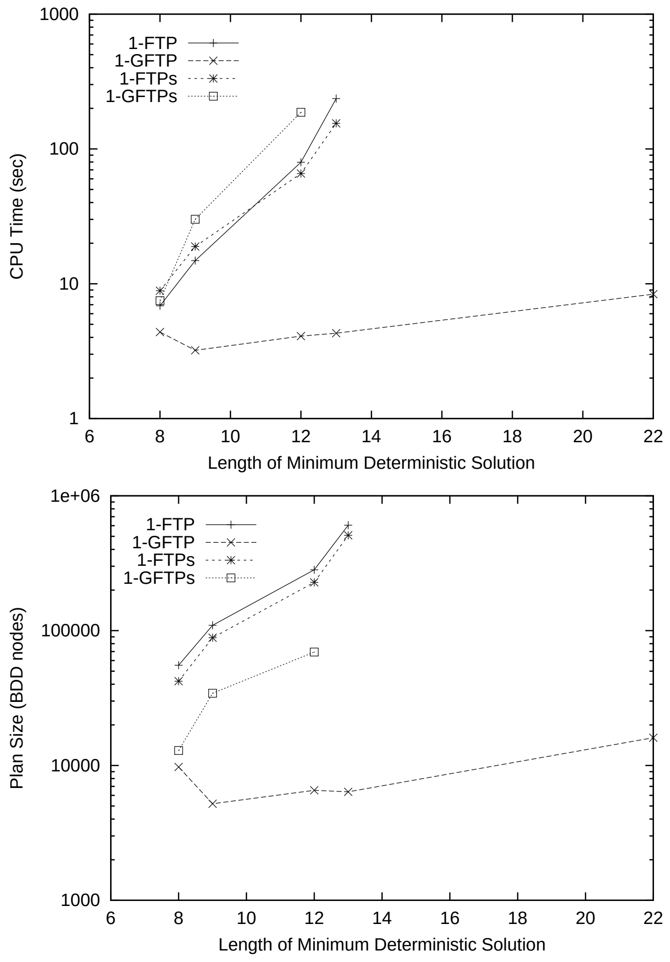

Figure 21: Results of the SIDMAR experiments.

fault tolerance for mission critical systems. A *masking fault tolerant system* can recover from any fault. A *t-fault tolerant system* can recover from up to t faults occurring during its life time. The system is modeled by an automaton with start states, but no goal states. In addition, no algorithms or theory for automated controller synthesis are provided. The state based models usually divides the state space into ranges of operation of some system e.g., "normal operation range", "admissible error range", and "non-admissible error range" (Klein & Wehlan, 1996), or "good" and "bad" states (Özveren & Willsky, 1991). As for transition based models, however, this work does not focus on developing efficient synthesis algorithms.

7. Conclusion

In this article, we have introduced n-fault tolerant plans as a new solution class of SNDP. Fault tolerant plans reside in the gap between weak plans and strong cyclic and strong plans. They are more strict than weak plans, but more relaxed than strong cyclic and strong plans. Optimal n-fault tolerant plans can be generated by the strong planning algorithm via a reduction to a strong planning problem. Our experimental evaluation shows, however, that due to non-local error states, it is often beneficial to decouple the planning for primary and secondary effects of actions.

Fault tolerant planning is a first step toward probabilistic uncertainty models in SNDP. A promising direction for future work is to move further in this direction and consider fault tolerant plans that are adjusted to the likelihood of faults or to consider probabilistic solution classes with other transition semantics than failures.

Acknowledgments

The authors wish to thank Paolo Traverso, Reid Simmons, and Bruce Krogh for fruitful discussions and feed back on this work. This research was carried out while the first author was at Carnegie Mellon University. The research is sponsored in part by the Danish Research Agency and the United States Air Force under Grants Nos F30602-00-2-0549 and F30602-98-2-0135. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency, the Air Force, or the US Government.

Appendix A. Proofs

This appendix contains correctness proofs for 1-FTP and 1-GFTP. For a code segment containing a loop that assigns to a variable C, we may use subscripts to refer to the different values assigned to C. Hence, C_0 denotes the value of C before the first iteration of the loop, and C_i for $i > 0$ denotes the value assigned to C in iteration i of the loop.

We define correctness of 1-FTP and 1-GFTP in terms of the induced non-deterministic plan. The algorithms work by adding precomponents f^1 and f^0 to the plans F^1 and F^0, respectively. The set of states covered by these plans are C^1 and C^0. To define the changes in the induced plan, we use the following notation

$$\begin{aligned}
(f^k)^{\mathcal{I}} &= \{\langle\langle s,k\rangle,a\rangle \,:\, \langle s,a\rangle \in f^k\},\\
\langle f^0,f^1\rangle^{\mathcal{I}} &= (f^0)^{\mathcal{I}} \cup (f^1)^{\mathcal{I}},\\
\pi^{\mathcal{I}} &= \langle F^0,F^1\rangle^{\mathcal{I}},\\
(C^k)^{\mathcal{I}} &= \{\langle s,k\rangle \,:\, s \in C^k\},\\
C^{\mathcal{I}} &= (C^0)^{\mathcal{I}} \cup (C^1)^{\mathcal{I}}.
\end{aligned}$$

A.1 1-FTP

Let subscript i denote the iteration number of the outer while loop (l.3-13) and let subscript j denote the iteration number of the inner while loop (l.6-11) for some arbitrary iteration of the outer loop.

Lemma 1 $C^0_{i+1} \supset C^0_i$.

Proof. It follows from l.4, 5, and 11 that $\textsc{States}(f^0_{i+1}) \cap C^0_i = \emptyset$. Further, l.13 gives $C^0_{i+1} = C^0_i \cup \textsc{States}(f^0_{i+1})$. If C^0_{i+1} is computed then l.6 has been executed which gives $f^0_{i+1} \neq \emptyset$. This proves the claim. □

Lemma 2 $C^1_{j+1} \supset C^1_j$.

Proof. It follows from l.7 and l.10 that $\textsc{States}(f^1_{j+1}) \cap C^1_j = \emptyset$ and $C^1_{j+1} = C^1_j \cup \textsc{States}(f^1_{j+1})$. Further, if C^1_{j+1} is computed then 1-FTP does not terminate at l.8. Thus $f^1_{j+1} \neq \emptyset$, which proves the claim. □

Theorem 3 (Termination) *1-*FTP *terminates.*

Proof. Assume by contradiction that 1-FTP diverges. Hence, at some point either the outer or inner while loop diverge. However, by Lemma 1 and Lemma 2 this would mean that the cardinality of C^0 or C^1 would be unbounded. But this is impossible since the state space of the planning problem is finite. □

Lemma 3 *If $\mathcal{M}(\pi^{\mathcal{I}}), C^{\mathcal{I}} \models \texttt{AF}\, G^{\mathcal{I}}$ before executing the inner loop in some iteration of the outer loop then $\mathcal{M}(\pi^{\mathcal{I}}_j), C^{\mathcal{I}}_j \models \texttt{AF}\, G^{\mathcal{I}}$, where $\pi^{\mathcal{I}}_j$ and $C^{\mathcal{I}}_j$ denote the value of $\pi^{\mathcal{I}}$ and $C^{\mathcal{I}}$ after iteration j of the inner loop.*

Proof. By induction on j.
Case $j = 0$. If $\mathcal{M}(\pi^{\mathcal{I}}), C^{\mathcal{I}} \models \texttt{AF}\, G^{\mathcal{I}}$ before executing the inner loop then trivially $\mathcal{M}(\pi^{\mathcal{I}}_0), C^{\mathcal{I}}_0 \models \texttt{AF}\, G^{\mathcal{I}}$ since $\pi^{\mathcal{I}}_0 = \pi^{\mathcal{I}}$ and $C^{\mathcal{I}}_0 = C^{\mathcal{I}}$.
Case $j > 0$. The induction hypothesis is $\mathcal{M}(\pi^{\mathcal{I}}_{j-1}), C^{\mathcal{I}}_{j-1} \models \texttt{AF}\, G^{\mathcal{I}}$ if $\mathcal{M}(\pi^{\mathcal{I}}), C^{\mathcal{I}} \models \texttt{AF}\, G^{\mathcal{I}}$ before executing the inner loop. L.7 gives, $f^1_j = \textsc{PreImg}(C^1_{j-1}) \setminus C^1_{j-1} \times \mathit{Act}$. Thus, for all $\langle s, a\rangle \in f^1_j$, we have $\textsc{Next}(s,a) \subseteq C^1_{j-1}$ and $\textsc{Next}(s,a) \cap C^1_{j-1} \neq \emptyset$. By definition of $\texttt{AF}$, this means $\mathcal{M}((f^1_j)^{\mathcal{I}}), \textsc{States}((f^1_j)^{\mathcal{I}}) \models \texttt{AF}\, (C^1_{j-1})^{\mathcal{I}}$. But then by definition of $\texttt{AF}$ and the induction hypothesis, we have $\mathcal{M}(\pi^{\mathcal{I}}_{j-1} \cup (f^1_j)^{\mathcal{I}}), \textsc{States}((f^1_j)^{\mathcal{I}}) \cup C^{\mathcal{I}}_{j-1} \models \texttt{AF}\, G^{\mathcal{I}}$. That is $\mathcal{M}(\pi^{\mathcal{I}}_j), C^{\mathcal{I}}_j \models \texttt{AF}\, G^{\mathcal{I}}$. □

Lemma 4 $\mathcal{M}(\pi^{\mathcal{I}}_i), C^{\mathcal{I}}_i \models \texttt{AF}\, G^{\mathcal{I}}$*, where $\pi^{\mathcal{I}}_i$ and $C^{\mathcal{I}}_i$ denote the value of $\pi^{\mathcal{I}}$ and $C^{\mathcal{I}}$ after iteration i of the outer loop.*

Proof. By induction on i.
Case $i = 0$. It follows from l.1 and l.2 that $C^{\mathcal{I}}_0 = G^{\mathcal{I}}$ and $\pi^{\mathcal{I}}_0 = \emptyset$. Thus, by definition of $\texttt{AF}$, $\mathcal{M}(\pi^{\mathcal{I}}_0), C^{\mathcal{I}}_0 \models \texttt{AF}\, G^{\mathcal{I}}$.
Case $i > 0$. The induction hypothesis is $\mathcal{M}(\pi^{\mathcal{I}}_{i-1}), C^{\mathcal{I}}_{i-1} \models \texttt{AF}\, G^{\mathcal{I}}$. Let $\pi^{\mathcal{I}}_{i'}$ and $C^{\mathcal{I}}_{i'}$ denote the value of $\pi^{\mathcal{I}}$ and $C^{\mathcal{I}}$ after executing the inner loop in iteration i of the outer loop. Since $\pi^{\mathcal{I}} = \pi^{\mathcal{I}}_{i-1}$ and $C^{\mathcal{I}} = C^{\mathcal{I}}_{i-1}$ before executing the inner loop, it follows from the induction hypothesis and Lemma 3 that $\mathcal{M}(\pi^{\mathcal{I}}_{i'}), C^{\mathcal{I}}_{i'} \models \texttt{AF}\, G^{\mathcal{I}}$. Further, it follows from l.4, 5, and 11 that $f^0_i = R \setminus U$, where $R = \textsc{PreImg}(C^0_{i'}) \setminus C^0_{i'} \times \mathit{Act}$ and $U = \textsc{PreImg}_f(\overline{C^1_{i'}})$. From $f^0_i \subseteq R$ it follows that for all $\langle s, a\rangle \in f^0_i$ we have $\textsc{Next}(s,a) \subseteq C^0_{i'}$ and $\textsc{Next}(s,a) \cap C^0_{i'} \neq \emptyset$. Moreover, since $f^0_i \cap U = \emptyset$, we have $\cup_{\langle s,a\rangle \in f^0_i} \textsc{Next}_f(s,a) \subseteq C^1_{i'}$. Thus by definition of $\texttt{AF}$, $\mathcal{M}((f^0_i)^{\mathcal{I}}), \textsc{States}((f^0_i)^{\mathcal{I}}) \models \texttt{AF}\, C^{\mathcal{I}}_{i'}$. Combining this with $\mathcal{M}(\pi^{\mathcal{I}}_{i'}), C^{\mathcal{I}}_{i'} \models \texttt{AF}\, G^{\mathcal{I}}$, we get $\mathcal{M}(\pi^{\mathcal{I}}_{i'} \cup (f^0_i)^{\mathcal{I}}), C^{\mathcal{I}}_{i'} \cup \textsc{States}((f^0_i)^{\mathcal{I}}) \models \texttt{AF}\, G^{\mathcal{I}}$. That is $\mathcal{M}(\pi^{\mathcal{I}}_i), C^{\mathcal{I}}_i \models \texttt{AF}\, G^{\mathcal{I}}$. □

Theorem 4 (Soundness) *1-*FTP *is sound.*

Proof. Assume that 1-FTP(s_0, G) terminates successfully in iteration i of the outer while loop and returns the 1-fault tolerant plan $\pi^{\mathcal{F}} = \langle F^0, F^1 \rangle$. We want to show that $\pi^{\mathcal{F}}$ is valid i.e., $\mathcal{M}(\pi^{\mathcal{I}}), s_0^{\mathcal{I}} \models \texttt{AF}\, G^{\mathcal{I}}$. It follows from l.3 that $s_0 \in C_i^0$. Thus, $s_0^{\mathcal{I}} \in C_i^{\mathcal{I}}$. But then since $\pi^{\mathcal{I}} = \pi_i^{\mathcal{I}}$ it follows from Lemma 4 that $\mathcal{M}(\pi^{\mathcal{I}}), s_0^{\mathcal{I}} \models \texttt{AF}\, G^{\mathcal{I}}$. □

Theorem 5 (Completeness) *1-*FTP *is complete.*

Proof. Assume by contradiction that there exists a 1-fault tolerant plan $\pi^{\mathcal{F}}$ such that $\mathcal{M}(\pi^{\mathcal{I}}), s_0^{\mathcal{I}} \models \texttt{AF}\, G^{\mathcal{I}}$ but that 1-FTP(s_0, G) terminates with failure in iteration j of the inner loop executed in iteration i of the outer loop. It follows from l.4 that $f_{c_i}^0 = \text{PreImg}(C_{i-1}^0) \setminus C_{i-1}^0 \times Act$. Further l.5, 6, and 11 give $f_i^0 = f_{c_i}^0 \setminus \text{PreImg}_f(\overline{C_j^1}) = \emptyset$. Thus, for any $\langle s, a \rangle \in f_{c_i}^0$, the current recovery plan F_j^1 does not cover its error states $\text{Next}_f(s, a) \not\subseteq \text{States}(F_j^1) = C_j^1$. By l.7 and l.8 $f_j^1 = \text{PreImg}(C_j^1) \setminus C_j^1 \times Act = \emptyset$. Thus, F^1 can not be extended to cover error states of SAs in $f_{c_i}^0$. However, this is impossible, since by l.3 $s_0 \notin C_{i-1}^0$ and there exists an execution path from s_0 to G of a 1-fault tolerant plan intersecting $\text{States}(f_{c_i}^0)$. □

A.2 1-GFTP

The proofs in this section use subscript i to denote the iteration number of the outer while loop (l.4-22) of 1-GFTP and subscript j to denote the iteration number of the while loop (l.6-11) of ExpandTimed. In addition, the proofs will often refer to the unbolded name of a map which denotes the set of elements stored in the map.

Lemma 5 $C_{i+1}^0 \supset C_i^0$.

Proof. From l.21 and l.22 of 1-GFTP, we have $C_{i+1}^0 = C_i^0 \cup (\text{States}(f_{i+1}^0) \cap PC_{i+1})$. Further, it follows from l.7, 12, 13, and 15 of 1-GFTP, l.5, 11, and 12 of ExpandTimed, and l.4 and l.5 of PreCompFTP that $f_{i+1}^0 \subseteq f_{c_{i+1}}^0 \subseteq \text{PC}_{i+1}$ and $\text{States}(\text{PC}_i) \cap C_i^0 = \emptyset$. Thus, $C_{i+1}^0 = C_i^0 \cup \text{States}(f_{i+1}^0)$ and $f_{i+1}^0 \cap C_i^0 = \emptyset$. But since l.17 of 1-GFTP has been executed in iteration i of the outer loop, we also have $f_{i+1}^0 \neq \emptyset$ which proves the claim. □

Lemma 6 *If the while loop of* ExpandTimed *does not terminate in iteration $j+2$ then $f_{c_{j+1}}^1 \supset f_{c_j}^1$.*

Proof. It follows from l.8 and l.9 that $f_{c_{j+1}}^1 \supseteq f_{c_j}^1$. But since the while loop continues in iteration $j+2$, we have $Oldf_{c_{j+1}}^1 \neq f_{c_{j+1}}^1$. Hence, it follows from l.7 that $f_{c_{j+1}}^1 \neq f_{c_j}^1$. □

Theorem 6 (Termination) *1-*GFTP *terminates.*

Proof. Assume by contradiction that 1-GFTP diverges. Hence, at some point either one of the sub-functions or loops of 1-GFTP diverges. Since the inner while loop (l.10-13) is bounded and the loops in PreCompFTP and PruneUnused are bounded, it must be either the outer while loop of 1-GFTP (l.4-22) or ExpandTimed that diverge. Assume ExpandTimed for some call diverges. By Lemma 6, however, this means that the cardinality of f_c^1 grows unbounded which is impossible since the state space is finite. Assume instead that the outer loop of 1-GFTP diverges. However, by Lemma 6 this means that the cardinality of C^0 grows unbounded which is impossible since the state space is finite. □

Lemma 7 *After executing the while loop in* ExpandTimed*, the following holds*

1. *for all* $\langle s,a\rangle \in \mathbf{f}_c^1[1]$*, we have* $\textsc{Next}(s,a) \subseteq C^1$ *and* $\textsc{Next}(s,a) \cap C^1 \neq \emptyset$ *,*
2. *for all* $\langle s,a\rangle \in \mathbf{f}_c^1[k]$*, where* $k > 1$*, we have* $\textsc{Next}(s,a) \subseteq \textsc{States}(\mathbf{f}_c^1[k-1])$ *and* $\textsc{Next}(s,a) \cap \textsc{States}(\mathbf{f}_c^1[k-1]) \neq \emptyset$ *.*

Proof. 1) Due to l.4 and l.9 of ExpandTimed and the fact that $f_c^1 = \emptyset$ (l.8 of 1-GFTP) when ExpandTimed assigns a value to $\mathbf{f}_c^1[1]$, we have $\mathbf{f}_c^1[1] = \textsc{PreImg}(C^1) \setminus C^1$ which proves the claim. 2) Let $R = \cup_{j=1}^{k-1}\textsc{States}(\mathbf{f}_c^1[j]) \cup C^1$. It follows from l.4, 9, and 10 of ExpandTimed that $\mathbf{f}_c^1[k] = \textsc{PreImg}(R) \setminus R \times Act$ for $k > 1$. Hence, for all $\langle s,a\rangle \in \mathbf{f}_c^1[k]$, we have $\textsc{Next}(s,a) \subseteq \textsc{States}(R)$ and $\textsc{Next}(s,a) \cap \textsc{States}(R) \neq \emptyset$. Assume by contradiction that there exists an $\langle s,a\rangle \in \mathbf{f}_c^1[k]$ such that $\textsc{Next}(s,a) \not\subseteq \textsc{States}(\mathbf{f}_c^1[k-1])$ or $\textsc{Next}(s,a) \cap \textsc{States}(\mathbf{f}_c^1[k-1]) = \emptyset$. But then $\langle s,a\rangle \in \cup_{j=1}^{k-1}\mathbf{f}_c^1[j]$ which is impossible since $\cup_{j=1}^{k-1}\mathbf{f}_c^1[j] \subseteq R \times Act$ and $\langle s,a\rangle \notin R \times Act$. □

Lemma 8 *After executing the for loop in* PruneUnused*, the following holds*

1. *for all* $\langle s,a\rangle \in \mathbf{f}_c^1[1]$*, we have* $\textsc{Next}(s,a) \subseteq C^1$ *and* $\textsc{Next}(s,a) \cap C^1 \neq \emptyset$ *,*
2. *for all* $\langle s,a\rangle \in \mathbf{f}_c^1[k]$*, where* $k > 1$*, we have* $\textsc{Next}(s,a) \subseteq \textsc{States}(\mathbf{f}_c^1[k-1])$ *and* $\textsc{Next}(s,a) \cap \textsc{States}(\mathbf{f}_c^1[k-1]) \neq \emptyset$ *.*

Proof. Let $\mathbf{f}_c^1[k]'$ denote the value of $\mathbf{f}_c^1[k]$ after executing the while loop of ExpandTimed and let $\mathbf{f}_c^1[k]$ denote the value of $\mathbf{f}_c^1[k]$ after executing the for loop of PruneUnused. 1) It follows from l.4 of PruneUnused that $\mathbf{f}_c^1[1] \subseteq \mathbf{f}_c^1[1]'$. Thus, the claim follows from Lemma 7 1). 2) It follows from l.2, 4, and 5 of PruneUnused that $\mathbf{f}_c^1[k-1] = \mathbf{f}_c^1[k-1]' \cap (err \cup (\cup_{\langle s,a\rangle \in \mathbf{f}_c^1[k]}\textsc{Next}(s,a))) \times Act$ and $\mathbf{f}_c^1[k] \subseteq \mathbf{f}_c^1[k]'$. Thus, it follows from Lemma 7 that for all $\langle s,a\rangle \in \mathbf{f}_c^1[k]$, we have $\textsc{Next}(s,a) \subseteq \textsc{States}(\mathbf{f}_c^1[k-1]')$ and $\textsc{Next}(s,a) \cap \textsc{States}(\mathbf{f}_c^1[k-1]') \neq \emptyset$. But since $\mathbf{f}_c^1[k-1]' \cap \textsc{Next}(s,a) \times Act \subseteq \mathbf{f}_c^1[k-1]$, we must also have $\textsc{Next}(s,a) \subseteq \textsc{States}(\mathbf{f}_c^1[k-1])$ and $\textsc{Next}(s,a) \cap \textsc{States}(\mathbf{f}_c^1[k-1]) \neq \emptyset$. □

Lemma 9 *After executing the for loop in* PruneUnused*, we have*
$\mathcal{M}((f_{c_k}^1)^{\mathcal{I}}), \textsc{States}((f_{c_k}^1)^{\mathcal{I}}) \models \texttt{AF}\,(C^1)^{\mathcal{I}}$*, where* $f_{c_k}^1 = \cup_{j=1}^{k}\mathbf{f}_c^1[j]$

Proof. By induction on k.
Case $k = 0$. Follows trivially from the definition of AF .
Case $k = 1$. Follows from Lemma 8 and the definition of AF .
Case $k > 1$. The induction hypothesis is $\mathcal{M}((f_{c_{k-1}}^1)^{\mathcal{I}}), \textsc{States}((f_{c_{k-1}}^1)^{\mathcal{I}}) \models \texttt{AF}\,(C^1)^{\mathcal{I}}$. It follows from Lemma 8 that for all $\langle s,a\rangle \in \mathbf{f}_c^1[k]$, we have $\textsc{Next}(s,a) \subseteq \textsc{States}(\mathbf{f}_c^1[k-1])$ and $\textsc{Next}(s,a) \cap \textsc{States}(\mathbf{f}_c^1[k-1]) \neq \emptyset$ after executing the for loop in PruneUnused. Thus, $\mathcal{M}((\mathbf{f}_c^1[k])^{\mathcal{I}}), \textsc{States}(\mathbf{f}_c^1[k])^{\mathcal{I}}) \models \texttt{AF}\,\textsc{States}((f_{c_{k-1}}^1)^{\mathcal{I}})$ which combined with the induction hypothesis and definition of AF gives $\mathcal{M}((f_{c_{k-1}}^1)^{\mathcal{I}} \cup (\mathbf{f}_c^1[k])^{\mathcal{I}}), \textsc{States}((f_{c_{k-1}}^1)^{\mathcal{I}} \cup (\mathbf{f}_c^1[k])^{\mathcal{I}}) \models \texttt{AF}\,(C^1)^{\mathcal{I}}$. That is $\mathcal{M}((f_{c_k}^1)^{\mathcal{I}}), \textsc{States}((f_{c_k}^1)^{\mathcal{I}}) \models \texttt{AF}\,(C^1)^{\mathcal{I}}$. □

Lemma 10 *At l.18 of 1-*GFTP*, we have* $\mathcal{M}((f^0)^{\mathcal{I}}), \textsc{States}((f^0)^{\mathcal{I}}) \models \texttt{AF}\,C^{\mathcal{I}} \cup \textsc{States}((f^1)^{\mathcal{I}})$.

Proof. It follows from l.4 and l.5 of PreCompFTP and l.12 of 1-GFTP that $f_c^0 \subseteq \mathrm{PC} \subseteq \overline{\textsc{PreImg}(C^0)}$. Further, l.4, 5, and 11 of ExpandTimed give $f^0 = f_c^0 \setminus \textsc{PreImg}(\overline{\textsc{States}(f_c^1) \cup C^1})$. Since $f^0 \subseteq f_c^0$, for all $\langle s,a\rangle \in f^0$, we have $\textsc{Next}(s,a) \subseteq C^0$ and $\textsc{Next}(s,a) \cap C^0 \neq \emptyset$. Furthermore, since $f^0 \cap \textsc{PreImg}(\overline{\textsc{States}(f_c^1) \cup C^1}) = \emptyset$, we have $\cup_{\langle s,a\rangle \in f^0} \textsc{Next}_f(s,a) \subseteq \textsc{States}(f_c^1) \cup C^1$. In PruneUnused, l.1 gives $err = \cup_{\langle s,a\rangle \in f^0} \textsc{Next}_f(s,a)$. Thus, after the for loop of PruneUnused, we still have $\cup_{\langle s,a\rangle \in f^0} \textsc{Next}_f(s,a) \subseteq \textsc{States}(f_c^1) \cup C^1$. Thus, by definition of $\mathtt{AF}$, we have $\mathcal{M}((f^0)^\mathcal{I}), \textsc{States}((f^0)^\mathcal{I}) \models \mathtt{AF}\,(C^0)^\mathcal{I} \cup (C^1)^\mathcal{I} \cup \textsc{States}((f^1)^\mathcal{I})$ at l.18 of 1-GFTP. That is $\mathcal{M}((f^0)^\mathcal{I}), \textsc{States}((f^0)^\mathcal{I}) \models \mathtt{AF}\,C^\mathcal{I} \cup \textsc{States}((f^1)^\mathcal{I})$. □

Lemma 11 *$\mathcal{M}(\pi_i^\mathcal{I}), C_i^\mathcal{I} \models \mathtt{AF}\,G^\mathcal{I}$, where $\pi_i^\mathcal{I}$ and $C_i^\mathcal{I}$ denote the value of $\pi^\mathcal{I}$ and $C^\mathcal{I}$ after iteration i of the outer loop of 1-*GFTP*.*

Proof. By induction on i.
Case $i = 0$. It follows from l.1 and l.2 of 1-GFTP that $C_0^\mathcal{I} = G^\mathcal{I}$ and $\pi_0^\mathcal{I} = \emptyset$. Thus, by definition of $\mathtt{AF}$, we have $\mathcal{M}(\pi_0^\mathcal{I}), C_0^\mathcal{I} \models \mathtt{AF}\,G^\mathcal{I}$.
Case $i > 0$. The induction hypothesis is $\mathcal{M}(\pi_{i-1}^\mathcal{I}), C_{i-1}^\mathcal{I} \models \mathtt{AF}\,G^\mathcal{I}$. We have $f_i^1 = f^1_{c_{|\mathbf{f}_c^1|}}$, where $f^1_{c_k}$ is defined in Lemma 9. Thus, by Lemma 9, $\mathcal{M}((f_i^1)^\mathcal{I}), \textsc{States}((f_i^1)^\mathcal{I}) \models \mathtt{AF}\,(C_{i-1}^1)^\mathcal{I}$. From Lemma 10, we get $\mathcal{M}((f_i^0)^\mathcal{I}), \textsc{States}((f_i^0)^\mathcal{I}) \models \mathtt{AF}\,C_{i-1}^\mathcal{I} \cup \textsc{States}((f_i^1)^\mathcal{I})$. Thus by definition of $\mathtt{AF}$ $\mathcal{M}((f_i^0)^\mathcal{I} \cup (f_i^1)^\mathcal{I}), \textsc{States}((f_i^0)^\mathcal{I} \cup (f_i^1)^\mathcal{I}) \models \mathtt{AF}\,C_{i-1}^\mathcal{I}$. Combined with the induction hypothesis we get $\mathcal{M}(\pi_{i-1}^\mathcal{I} \cup (f_i^0)^\mathcal{I} \cup (f_i^1)^\mathcal{I}), \textsc{States}(\pi_{i-1}^\mathcal{I} \cup (f_i^0)^\mathcal{I} \cup (f_i^1)^\mathcal{I}) \models \mathtt{AF}\,C_{i-1}^\mathcal{I}$. That is $\mathcal{M}(\pi_i^\mathcal{I}), \textsc{States}(\pi_i^\mathcal{I}) \models \mathtt{AF}\,C_{i-1}^\mathcal{I}$. □

Theorem 7 (Soundness) *1-*GFTP *is sound.*

Proof. Assume that 1-GFTP(s_0, G) terminates successfully in iteration i of the outer loop and returns the 1-fault tolerant plan $\pi^\mathcal{F} = \langle F^0, F^1\rangle$. We want to show that $\pi^\mathcal{F}$ is valid i.e., $\mathcal{M}(\pi^\mathcal{I}), s_0^\mathcal{I} \models \mathtt{AF}\,G^\mathcal{I}$. It follows from l.4 that $s_0 \in C_i^0$. Thus, $s_0^\mathcal{I} \in C_i^\mathcal{I}$. But then it follows from Lemma 11 that $\mathcal{M}(\pi^\mathcal{I}), s_0^\mathcal{I} \models \mathtt{AF}\,G^\mathcal{I}$, since $\pi^\mathcal{I} = \pi_i^\mathcal{I}$. □

Theorem 8 (Completeness) *1-*GFTP *is complete.*

Proof. Assume by contradiction that there exists a 1-fault tolerant plan $\pi^\mathcal{F}$ such that $\mathcal{M}(\pi^\mathcal{I}), s_0^\mathcal{I} \models \mathtt{AF}\,G^\mathcal{I}$ but that 1-GFTP(s_0, G) terminates with failure in iteration i of the outer loop. Since l.17 of 1-GFTP is executed, we know that the second call to ExpandTimed terminated and returned $f^0 = \emptyset$. However, in the second call to ExpandTimed, we have $t = \infty$ which means that the while loop of ExpandTimed terminated because $Oldf_c^1 = f_c^1$. At this point, l.12 of 1-GFTP and l.4 of PreCompFTP give $f_c^0 = \textsc{PreImg}(C^0) \setminus \overline{C^0 \times Act}$, and from l.5 and l.11 of ExpandTimed it follows $f^0 = f_c^0 \setminus \textsc{PreImg}_f(\overline{\textsc{States}(f_c^1) \cup C^1})$. Since $Oldf_c^1 = f_c^1$ there does not exist a recovery plan F^1 that cover more than the states $C^1 \cup \textsc{States}(f_c^1)$, and it follows from $f^0 = \emptyset$ and the expression above that this recovery plan does not cover any of the error states of f_c^0 (i.e., there does not exist $\langle s,a\rangle \in f_c^0$ such that $\textsc{Next}_f(s,a) \subseteq C^1 \cup \textsc{States}(f_c^1)$). However, this is impossible, since by l.4 of 1-GFTP, we have $s_0 \notin C^0$ and there exists an execution path from s_0 to G of a 1-fault tolerant plan intersecting $\textsc{States}(f^0)$. □

References

Bahar, R., Frohm, E., Gaona, C., Hachtel, E., Macii, A., Pardo, A., & Somenzi, F. (1993). Algebraic decision diagrams and their applications. In *IEEE/ACM International Conference on CAD*, pp. 188–191.

Balemi, S., Hoffmann, G. J., Gyugyi, P., Wong-Toi, H., & Franklin, G. F. (1993). Supervisory control of a rapid thermal multiprocessor. *IEEE Trans. on Automatic Control*, *38*(7).

Bertoli, P., Cimatti, A., Slanley, J., & Thiébaux, S. (2002). Solving power supply restoration problems with planning via symbolic model checking. In *Proceedings of the 15th European Conference on Artificial Intelligence ECAI'02*.

Bryant, R. E. (1986). Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, *8*, 677–691.

Chen, J., & Patton, R. J. (1999). *Robust Model-Based Fault Diagnosis for Dynamic Systems*. Kluwer Academic Publishers.

Cho, K.-H., & Lim, J.-T. (1998). Synthesis of fault tolerant supervisor for automated manufacturing systems: A case study on photolithographic process. *IEEE Trans. on Robotics and Automation*, 348–351.

Cimatti, A., Pistore, M., Roveri, M., & Traverso, P. (2003). Weak, Strong, and Strong Cyclic Planning via Symbolic Model Checking. *Artificial Intelligence*, *147*(1-2). Elsevier Science publishers.

Cimatti, A., Roveri, M., & Traverso, P. (1998). Automatic OBDD-based generation of universal plans in non-deterministic domains. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI'98)*, pp. 875–881. AAAI Press.

Cin, M. D. (1997). Verifying fault-tolerant behavior of state machines. In *Proceedings of the Second IEEE High-Assurance Systems Engineering Workshop HASE 97*, pp. 97–99.

Doyle, R. J. (1995). Determining the loci of anomalies using minimal causal models. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, pp. 1821–1827.

Fehnker, A. (1999). Scheduling a steel plant with timed automata. In *Sixth International Conference on Real-Time Computing Systems and Applications (RTCSA'99)*. IEEE Computer Society Press.

Feng, Z., & Hansen, E. (2002). Symbolic LAO* search for factored markov decision processes. In *Proceedings of the AIPS-02 Workshop on Planning via Model Checking*, pp. 49–53.

Georgeff, M., & Lansky, A. L. (1986). Procedural knowledge. *Proceedings of IEEE*, *74*(10), 1383–1398.

Giunchiglia, F., Spalazzi, L., & Traverso, P. (1994). Planning with failure. In *Proceedings of the 2nd International Conference on Artificial Intelligence Planning Systems*.

Hammond, K. (1990). Explaining and repairing plans that fail. *Artificial Intelligence*, *40*, 173–228.

Hoey, J., St-Aubin, R., & Hu, A. (1999). SPUDD: Stochastic planning using decision diagrams. In *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence*, pp. 279–288.

Jensen, R. M. (2002). A comparison study between the CUDD and BuDDy OBDD package applied to AI-planning problems. Tech. rep., Computer Science Department, Carnegie Mellon University. CMU-CS-02-173.

Jensen, R. M. (2003a). *Efficient BDD-Based Planning for Non-Deterministic, Fault-Tolerant, and Adversarial Domains*. Ph.D. thesis, Carnegie Mellon University. CMU-CS-03-139.

Jensen, R. M. (2003b). The BDD-based InFoRmed planning and cOntroller Synthesis Tool BIFROST version 0.7. `http://www.cs.cmu.edu/~runej/systems/bifrost07.html`.

Jensen, R. M., & Veloso, M. M. (2000). OBDD-based universal planning for synchronized agents in non-deterministic domains. *Journal of Artificial Intelligence Research*, *13*, 189–226.

Jensen, R. M., Veloso, M. M., & Bryant, R. E. (2003). Guided symbolic universal planning. In *Proceedings of the 13th International Conference on Automated Planning and Scheduling ICAPS-03*, pp. 123–132.

Kleer, J., & Williams, B. C. (1987). Diagnosing multiple faults. *Artificial Intelligence*, *32*(1), 97–130.

Klein, E., & Wehlan, H. (1996). Systematic design of a protective controller in process industries by means of the boolean differential calculus. In *Proceedings of WODES-96*.

Lind-Nielsen, J. (1999). BuDDy - A Binary Decision Diagram Package. Tech. rep. IT-TR: 1999-028, Institute of Information Technology, Technical University of Denmark. `http://cs.it.dtu.dk/buddy`.

McCalla, G., & Ward, B. (1982). Error detection and recovery in a dynamic planning environment. In *Proceedings of the 2nd National Conference on Artificial Intelligence (AAAI'82)*, pp. 172–175.

Muscettola, N., Nayak, P. P., Pell, B., & Williams, B. C. (1998). Remote agent: To boldly go where no AI system has gone before. *Artificial Intelligence*, *103*(1-2), 5–47.

Özveren, C. M., & Willsky, A. S. (1991). Stability and stabilizability of discrete event dynamic systems. *Journal of ACM*, 730–752.

Pecheur, C., & Simmons, R. (2000). From livingstone to SMV. In *FAABS*, pp. 103–113.

Peot, M., & Smith, D. (1992). Conditional nonlinear planning. In *Proceedings of the 1'st International Conference on Artificial Intelligence Planning Systems (AIPS'92)*, pp. 189–197. Morgan Kaufmann.

Perraju, T. S., Rana, S. P., & Sarkar, S. P. (1997). Specifying fault tolerance in mission critical systems. In *Proceedings of High-Assurance Systems Engineering Workshop, 1996*, pp. 24–31. IEEE.

Ramadge, P. J., & Wonham, W. M. (1987). Supervisory control of a class of discrete event processes. *SIAM J. Control Optim.*, *25*(1), 206–230.

Sampath, M., Sengupta, R., Lafortune, S., Sinnamohideen, K., & Teneketzis, D. (1995). Diagnosability of discrete-event systems. *IEEE Trans. on Automatic Control*, *40*(9), 1555–1575.

Sampath, M., Sengupta, R., Lafortune, S., Sinnamohideen, K., & Teneketzis, D. (1996). Failure diagnosis using discrete-event models. *IEEE Trans. on Control Systems Technology*, *4*(2), 105–123.

Sampath, M., Sengupta, R., Lafortune, S., & Teneketzis, D. (1998). Active diagnosis of discrete-event systems. *IEEE Trans. on Automatic Control*, *43*(7), 908–929.

Senjen, R., & De Beler, M. (1993). Hybrid expert systems for monitoring and fault diagnosis. In *Proceedings of the 9th IEEE Conference on Artificial Intelligence Applications*, pp. 235–241.

Somenzi, F. (1996). CUDD: Colorado university decision diagram package.. `ftp://vlsi.colorado.edu/pub/`.

Su, R. (2001). Decentralized fault diagnosis for discrete-event systems. Master's thesis, Dept. Electl. Engrg., Univ. of Toronto.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement Learning: an Introduction.* MIT Press.

Tesauro, G. (1995). Temporal difference learning and TD-Gammon. *Communications of the ACM*, *38*(3), 58–68.

Thiébaux, S., & Cordier, M. O. (2001). Supply restoration in power distribution systems – a benchmark for planning under uncertainty. In *Pre-Proceedings of the 6th European Conference on Planning (ECP-01)*, pp. 85–96.

Weld, D. S., Anderson, C. R., & Smith, D. E. (1998). Extending graphplan to handle uncertainty and sensing actions. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI'98).*

Williams, B., & Nayak, P. (1996). A model-based approach to reactive self-configuring systems. In *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI'96).*

Williams, B. C., Ingham, M., Chung, S. H., & Elliott, P. H. (2003). Model-based programming of intelligent embedded systems and robotic space explorers. In *Proceedings of the IEEE: Special Issue on Modeling and Design of Embedded Software*, Vol. 9, pp. 212–237.

Yang, B., Simmons, R., Bryant, R. E., & O'Hallaron, R. O. (1999). Optimizing symbolic model checking for constraint-rich models. In *Proceedings of Computer-Aided Verification (CAV'99)*, pp. 328–340.