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

Rune M. Jensen
IT University of Copenhagen,
2400 Copenhagen, Denmark
rmj@itu.dk

Manuela M. Veloso and **Randal E. Bryant**
Computer Science Department, Carnegie Mellon University
Pittsburgh,PA 15213-3891, USA
{mmv,bryant}@cs.cmu.edu

Abstract

Symbolic non-deterministic planning represents action effects as sets of possible next states. In this paper, 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.

Introduction

MDP solving (e.g., Puterman 1994) and Symbolic Non-Deterministic Planning (SNDP) (e.g., Cimatti *et al.* 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. 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. In particular, symbolic methods based on Algebraic Decision Diagrams (ADDs, Bahar *et al.* 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 paper, 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 (Balemi *et al.* 1993; Perraju, Rana, & Sarkar 1997). 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 combinations of 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

*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 in this document are those of the authors and should not be interpreted as necessarily representing the official policies of DARPA, the Air Force, or the US Government.

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 by 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 paper 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 (ICAPS-03 2003). 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 & Bryant (2003). The n-GFTP$_S$ algorithm is efficient when secondary effects are local. A secondary effect is local when there exists a short path leading from any resulting state of the secondary effect (an error state) to the resulting state of the primary effect of the action (the state reached when the action succeeds). When secondary effects are local, the error states 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 experiments illustrate the natural connection between the existence of fault tolerant plans and the redundancy characteristics of the modeled system. Moreover, they show that even 1-fault tolerant plans impose much stronger requirements on the system than weak plans. Finally, they indicate that faults in real-world domains often cause non-local transitions that require specialized planning algorithms to be handled efficiently.

Previous work explicitly representing and reasoning about action failure is very limited. Some reactive planning approaches take action failure into account (e.g. Georgeff & Lansky 1986; Williams *et al.* 2003), but do not involve producing 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 paper are the first automated planning algorithms for generating fault tolerant plans given a description of the domain that explicitly represents failure effects of actions.

In the following section, we present necessary SNDP terminology and results. We then define n-fault tolerant plans and describe the developed fault tolerant planning algorithms. Finally, we present our experimental evaluation and draw conclusions.

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

Following Cimatti *et al.* (2003), we use CTL to 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* A ("for all execution paths") and E ("for some execution path") and one *temporal operator* 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.

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

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

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 $\mathcal{M}(\pi), s_0 \models \mathtt{AF}\,G$.

Weak, strong cyclic, and strong plans can be synthesized by the NDP algorithm shown below. The algorithm performs a backward breadth-first search from the goal states to the initial state. 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), NDP computes the state-action pairs (SAs) of the backward search frontier of the states C currently covered by the plan (l.3). This set of SAs is called a precomponent of C since it contains states that can reach C in one step. In a guided version of the algorithm (Jensen, Veloso, & Bryant 2003), the SAs of the precomponent is partitioned according to a heuristic measure (e.g. an estimate of the distance to the initial state).

function NDP(s_0, G)
1 $P \leftarrow \emptyset$; $C \leftarrow G$
2 **while** $s_0 \notin C$
3 $\quad P_c \leftarrow \text{PRECOMP}(C)$
4 $\quad$ **if** $P_c = \emptyset$ **then return** "no plan exists"
5 $\quad$ **else**
6 $\quad\quad P \leftarrow P \cup P_c$
7 $\quad\quad C \leftarrow C \cup \text{STATES}(P_c)$
8 **return** P

The strong, strong cyclic, and weak planning algorithms only differ by the definition of the precomponent. Let $\text{PREIMG}(C)$ denote the set of SAs where the action applied in the state may lead into the set of states C. That is $\text{PREIMG}(C) \equiv \{\langle s, a \rangle : \text{NEXT}(s, a) \cap C \neq \emptyset\}$. The weak and strong precomponent are then defined by

$$PC_w(C) \equiv \text{PREIMG}(C) \setminus C \times Act,$$
$$PC_s(C) \equiv (\text{PREIMG}(C) \setminus \text{PREIMG}(\overline{C})) \setminus C \times Act.$$

The strong cyclic precomponent depends a fixed point computation. We refer the reader to Jensen (2003a) for details.

Due to the breadth-first search carried out by NDP, weak solutions have minimum length best-case execution paths and strong solutions 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 $\text{EXEC}(s, \pi) \equiv \{q : q \text{ is a path of } \mathcal{M}(\pi) \text{ and } q_0 = s\}$ denote the set of execution paths of π starting at s. Let the length of a path $q = q_0 q_1 \cdots$ with respect to a set of states C be defined by

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

Let $\text{MIN}(s, C, \pi)$ and $\text{MAX}(s, C, \pi)$ denote the minimum and maximum length of an execution path from s to C of a plan π

$$\text{MIN}(s, C, \pi) \equiv \min_{q \in \text{EXEC}(s,\pi)} |q|_C,$$
$$\text{MAX}(s, C, \pi) \equiv \max_{q \in \text{EXEC}(s,\pi)} |q|_C.$$

Similarly, let Π denote the set of all plans of $\mathcal{D}$ and let $\text{WDIST}(s, C)$ (weak distance) and $\text{SDIST}(s, C)$ (strong distance) denote the minimum of $\text{MIN}(s, C, \pi)$ and $\text{MAX}(s, C, \pi)$ for any plan $\pi \in \Pi$ of $\mathcal{D}$

$$\text{WDIST}(s, C) \equiv \min_{\pi \in \Pi} \text{MIN}(s, C, \pi),$$
$$\text{SDIST}(s, C) \equiv \min_{\pi \in \Pi} \text{MAX}(s, C, \pi).$$

Let WEAK and STRONG denote the NDP algorithm where $\text{PRECOMP}(C)$ is substituted with $PC_w(C)$ and $PC_s(C)$. For a weak plan $\pi_w = \text{WEAK}(s_0, G)$ and strong plan $\pi_s = \text{STRONG}(s_0, G)$, we then have

$$\text{MIN}(s_0, G, \pi_w) = \text{WDIST}(s_0, G),$$
$$\text{MAX}(s_0, G, \pi_s) = \text{SDIST}(s_0, G).$$

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.

Definition 1 (Fault Tolerant Planning Domain) *A fault tolerant planning domain is a tuple $\langle S, Act, \rightarrow, \rightsquigarrow \rangle$ 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.*

An n-fault tolerant planning problem is a non-deterministic planning problem extended with the fault limit n.

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

An n-fault tolerant plan is defined via a transformation of an n-fault tolerant planning problem to a non-deterministic planning problem. The transformation adds a fault counter f to the state description and models secondary effects only when $f \leq n$.

Definition 3 (Induced Non-Det. Planning Problem) *Let $\mathcal{P} = \langle \mathcal{D}, s_0, G, n \rangle$ where $\mathcal{D} = \langle S, Act, \rightarrow, \rightsquigarrow \rangle$ be an n-fault tolerant planning problem. The non-deterministic planning problem induced from $\mathcal{P}$ is $\mathcal{P}^{nd} = \langle \mathcal{D}^{nd}, \langle s_0, 0 \rangle, G \times \{0, \cdots, n\} \rangle$ where $\mathcal{D}^{nd} = \langle S^{nd}, Act^{nd}, \rightarrow^{nd} \rangle$ given by*

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

Definition 4 (Valid N-Fault Tolerant Plan) *A valid n-fault tolerant plan is a non-deterministic plan π for the non-deterministic planning problem induced from $\mathcal{P}$ where $\mathcal{M}(\pi), s_0^{nd} \models \mathtt{AF}\, G^{nd}$.*

Thus, an n-fault tolerant plan is valid if any execution path, where at most n failures happen, eventually reaches a goal state. An n-fault tolerant plan is optimal if it has minimum worst case execution length.

Definition 5 (Optimal N-Fault Tolerant Plan) *An optimal n-fault tolerant plan is a valid n-fault tolerant plan π where $\textsc{Max}(s_0^{nd}, G^{nd}, \pi) = \textsc{SDist}(s_0^{nd}, G^{nd})$.*

N-Fault Tolerant Planning Algorithms

It follows directly from the definition of strong plans that the STRONG algorithm returns a valid n-fault tolerant plan, if it exists, when given the induced non-deterministic planning problem as input. Moreover, it follows from the optimality of STRONG that the returned n-fault tolerant plan also is optimal. Let n-FTP$_S$ denote the STRONG algorithm applied to an n-fault tolerant planning problem. To improve performance further, we also consider an algorithm n-GFTP$_S$ based on the guided version of STRONG described in Jensen (2003a). Due to the pure heuristic search approach, n-GFTP$_S$ may return suboptimal solutions.

We may expect n-GFTP$_S$ to be 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. Indeed, in theory, the location of secondary effects may be completely uncorrelated with the location of primary effects. To solve this problem, we develop specialized algorithms where the planning for primary and secondary effects is decoupled. We constrain our investigation to 1-fault tolerant planning and introduce two algorithms: 1-FTP using blind search and 1-GFTP using guided search. 1-FTP is shown below. The function PREIMG$_f$ computes

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 $\quad f_c^0 \leftarrow \textsc{PreImg}(C^0) \setminus C^0 \times Act$
5 $\quad f^0 \leftarrow f_c^0 \setminus \textsc{PreImg}_f(\overline{C^1})$
6 $\quad$ **while** $f^0 = \emptyset$
7 $\quad\quad f^1 \leftarrow \textsc{PreImg}(C^1) \setminus C^1 \times Act$
8 $\quad\quad$ **if** $f^1 = \emptyset$ **then return** "no plan exists"
9 $\quad\quad F^1 \leftarrow F^1 \cup f^1$
10 $\quad\quad C^1 \leftarrow C^1 \cup \textsc{States}(f^1)$
11 $\quad\quad f^0 \leftarrow f_c^0 \setminus \textsc{PreImg}_f(\overline{C^1})$
12 $\quad F^0 \leftarrow F^0 \cup f^0$
13 $\quad C^0 \leftarrow C^0 \cup \textsc{States}(f^0)$
14 **return** $\langle F^0, F^1 \rangle$

the preimage of secondary effects. 1-FTP returns two non-deterministic plans F^0 and F^1 for the fault tolerant domain where F^0 is applied when no error has occurred, and the recovery plan F^1 is applied when one error has happened. An example of the non-deterministic plans F^0 and F^1 returned by 1-FTP is shown in Figure 1. 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 zero length 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).

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,

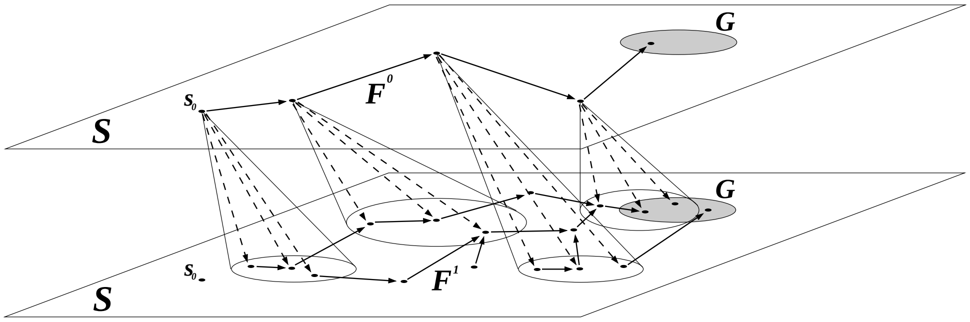


Figure 1: An example of the non-deterministic plans F^0 and F^1 returned by 1-FTP. Primary and secondary effects of actions are drawn with solid and dashed lines, respectively. In this example, we assume that F^0 forms a sequence of actions from the initial state to a goal state, while F^1 recovers all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

however, makes it suboptimal as the example in Figure 2 shows. In the first two iterations of the outer loop, $\langle p_2, b\rangle$ and

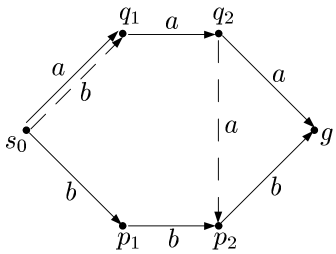


Figure 2: A problem with a single goal state g showing that 1-FTP may return suboptimal solutions. Dashed lines indicate secondary effects. Notice that action a and b only have secondary effects in q_2 and s_0, respectively. In all other states, the actions are assumed always to succeed.

$\langle p_1, b\rangle$ 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.

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. The over all design goal of 1-GFTP is to guide the expansion of F^0 toward the initial state using a heuristic h estimating the distance to the initial state, and then 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$_S$.

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 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 does not seem to be the case 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 location* of error states is non-local. However, the strategy assumes that the *relative location* 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 lowest 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 below. The keys in maps are sorted ascendingly. 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. [1] 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

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 $\quad t_s \leftarrow t_{CPU}$
6 $\quad \mathbf{PC} \leftarrow$ PRECOMPFTP$(\mathbf{C}^0)$
7 $\quad f^0 \leftarrow \emptyset$; $f_c^0 \leftarrow \emptyset$
8 $\quad \mathbf{f}_c^1 \leftarrow emptyMap$
9 $\quad i \leftarrow 0$
10 $\quad$ **while** $f^0 = \emptyset \wedge i < |\mathbf{PC}|$
11 $\quad\quad i \leftarrow i + 1$; $t \leftarrow t/2$
12 $\quad\quad f_c^0 \leftarrow f_c^0 \cup \mathbf{PC}[i]$
13 $\quad\quad \langle \mathbf{f}_c^1, f^0 \rangle \leftarrow$ EXPANDTIMED$(f_c^0, \mathbf{f}_c^1, C^1, t)$
14 $\quad$ **if** $f^0 = \emptyset$ **then**
15 $\quad\quad \langle \mathbf{f}_c^1, f^0 \rangle \leftarrow$ EXPANDTIMED$(f_c^0, \mathbf{f}_c^1, C^1, \infty)$
16 $\quad t \leftarrow t_{CPU} - t_s$
17 $\quad$ **if** $f^0 = \emptyset$ **then return** "no plan exists"
18 $\quad f^1 \leftarrow$ PRUNEUNUSED$(\mathbf{f}_c^1, f^0)$
19 $\quad F^1 \leftarrow F^1 \cup f^1$; $C^1 \leftarrow C^1 \cup$ STATES(f^1)
20 $\quad F^0 \leftarrow F^0 \cup f^0$
21 $\quad$ **for** $j = 1$ **to** i
22 $\quad\quad \mathbf{C}^0[h_j] \leftarrow \mathbf{C}^0[h_j] \cup$ STATES$(f^0 \cap \mathbf{PC}[h_j])$
23 **return** $\langle F^0, F^1 \rangle$

(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 (l. 6). For each entry in $\mathbf{C}^0$, PRECOMPFTP inserts the preimage in $\mathbf{PC}$ of each partition of the transition relation of primary effects. We assume that this partitioning has m subrelations $R_1, \cdots R_m$ where the transitions represented by R_i are associated with a change δh_i of the h-value (in forward direction). The inner loop (l. 10-13) of 1-GFTP expands the two candidates f_c^0 and f_c^1

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

for f^0 and f^1. In each iteration, a partition of the partitioned precomponent $\mathbf{PC}$ is added to f_c^0 (l. 12).[2] The function EXPANDTIMED (shown below) expands f_c^1. In iteration i, the time-out bound of the expansion is $t/2^i$. EXPANDTIMED returns early if:

1. A precomponent f^0 in the candidate f_c^0 is found where all error states are recovered (l. 5 and l. 11), or
2. f_c^1 has reached a fixed point.

The preimage added to f_c^1 in iteration i of EXPANDTIMED is stored in the map entry $\mathbf{f}_c^1[i]$ in order to prune SAs not used for recovery. Eventually f_c^0 may contain all the SAs in $\mathbf{PC}$ without any of these being recoverable. In this case, 1-GFTP expands f_c^1 (l. 15) untimed. 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 "no plan exists" (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 (shown below) that traverses backward through the preimages of $\mathbf{f}_c^1$ and marks states that either are error states of SAs in f^0, or states needed to recover previously marked states. The functions IMG(π) and IMG$_f(\pi)$ compute the reachable states of a set of SAs π for primary and secondary effects, respectively.

$$\begin{aligned} \text{IMG}(\pi) &\equiv \{s' : \exists \langle s, a \rangle \in \pi \,.\, s \xrightarrow{a} s'\}, \\ \text{IMG}_f(\pi) &\equiv \{s' : \exists \langle s, a \rangle \in \pi \,.\, s \overset{a}{\leadsto} s'\}. \end{aligned}$$

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

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

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 BIFROST 0.7 search engine (Jensen 2003b). 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. We refer the reader to Jensen (2003a) for a detailed description of the experiments.

[1] To simplify the presentation, we assume that all goal states have identical h-value. A generalization of the algorithm is trivial.

[2] Recall that $\mathbf{PC}$ is traversed ascendingly such that the partition with lowest h-value is added first.

function $\text{EXPANDTIMED}(f_c^0, \mathbf{f}_c^1, C^1, t)$

$t_s \leftarrow t_{CPU}$
$Oldf_c^1 \leftarrow \perp$
$i \leftarrow |\mathbf{f}_c^1|$
$recovS \leftarrow \text{STATES}(f_c^1) \cup C^1$
$f^0 \leftarrow f_c^0 \setminus \text{PREIMG}_f(\overline{recovS})$
while $f^0 = \emptyset \wedge Oldf_c^1 \neq f_c^1 \wedge t_{CPU} - t_s < t$
$\quad Oldf_c^1 \leftarrow f_c^1$
$\quad i \leftarrow i + 1$
$\quad \mathbf{f}_c^1[i] \leftarrow \text{PREIMG}(recovS) \setminus recovS \times Act$
$\quad recovS \leftarrow \text{STATES}(f_c^1) \cup C^1$
$\quad f^0 \leftarrow f_c^0 \setminus \text{PREIMG}_f(\overline{recovS})$
return $\langle \mathbf{f}_c^1, f^0 \rangle$

function $\text{PRUNEUNUSED}(\mathbf{f}_c^1, f^0)$

$err \leftarrow \text{IMG}_f(f^0)$
$img \leftarrow \emptyset;\ marked \leftarrow \emptyset$
for $i = |\mathbf{f}_c^1|$ **to** 1
$\quad \mathbf{f}_c^1[i] \leftarrow \mathbf{f}_c^1[i] \cap \big((err \cup img) \times Act\big)$
$\quad marked \leftarrow marked \cup \text{STATES}(\mathbf{f}_c^1[i])$
$\quad img \leftarrow \text{IMG}(\mathbf{f}_c^1[i])$
return $f_c^1 \cap (marked \times Act)$

Unguided Search

The main purpose of these experiments is to investigate fault tolerant plans for significant real-world domains and compare the performance of 1-FTP$_S$ and 1-FTP. With respect to the former, we have studied NASA's Deep Space One domain (DS1) and the "simple" Power Supply Restoration domain (PSR). In addition, we have examined an artificial power plant domain. For all of these domains, even 1-fault tolerant plans turn out to inflict high restrictions on the domain compared to weak plans. In particular, only 2 of the original 4 errors of the DS1 experiment could be considered in the domain model due to lack of redundancy in DS1's electrical system. This result is encouraging since it proves fault tolerant plans to be a substantially stronger solution class than weak plans and in addition illustrates the natural connection between the existence of fault tolerant plans and the redundancy characteristics of the modeled system.

The 1-FTP algorithm roughly outperforms 1-FTP$_S$ by a factor of 2 in all of these experiments. To investigate this performance difference further, we made additional experiments with a linear version of the PSR domain and the Beam domain (Cimatti *et al.* 2003). Figure 3 shows the results. As depicted 1-FTP has significantly better performance than 1-FTP$_S$ also in these domains.

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 one descends from a real-world study.

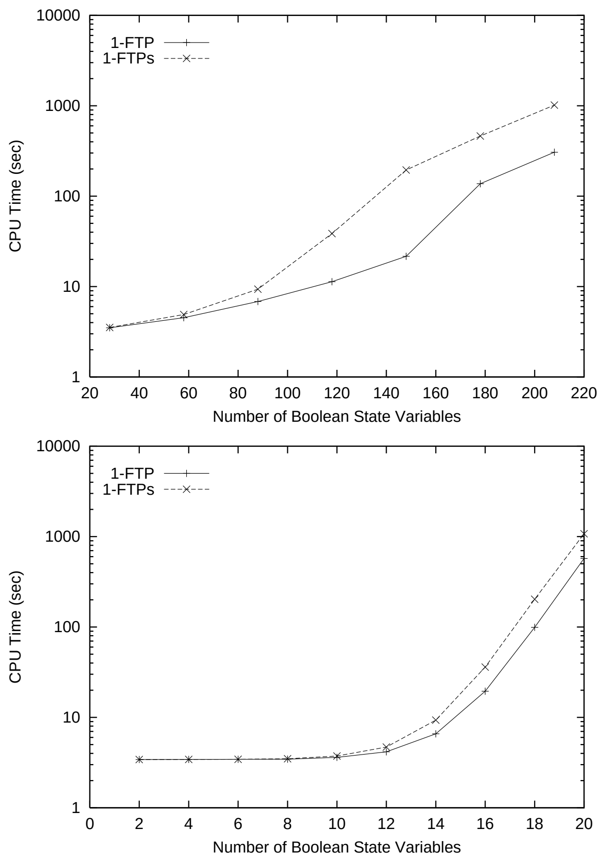
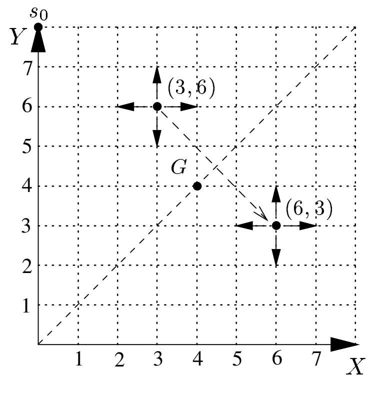


Figure 3: Results of linear PSR (top) and Beam experiments (bottom).

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 below. The 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 results are shown in Figure 4.

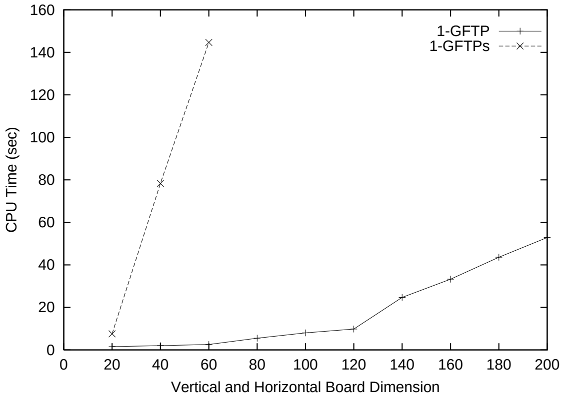


Figure 4: The results of the LV experiments.

As 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.

8-Puzzle The 8-Puzzle further demonstrates this difference between 1-GFTP and 1-GFTP$_S$. We consider a non-deterministic version of the 8-Puzzle where the secondary effects are self loops. Thus, error states are the most local possible. We use the usual sum of Manhattan distances of tiles as a heuristic estimate of the distance to the initial state. The results are shown in Figure 6. 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.

SIDMAR The final experiments are on the SIDMAR steel plant 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 5. The goal is to cast steel of different qualities. Pig iron is

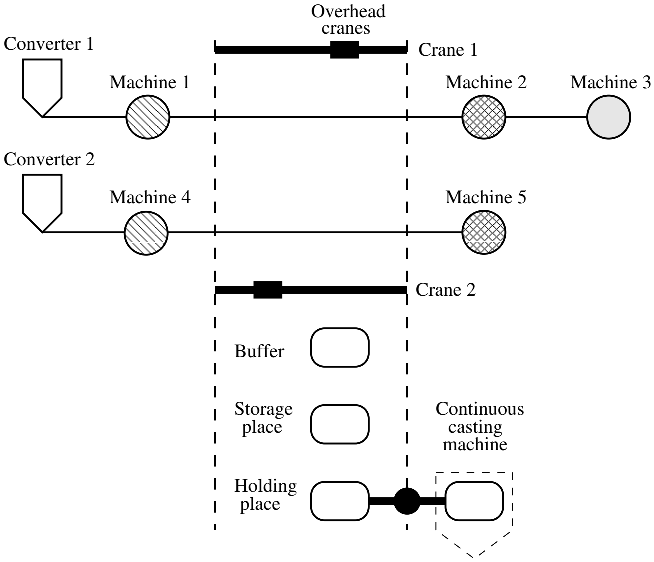


Figure 5: Layout of the SIDMAR steel plant.

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 heuristic is the sum of machine treatments carried out on the ladles. The results are shown in Figure 6. 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 limit 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 minimal. Thus, the strategy of 1-FTP to keep the recovery plan as small as possible does not seem to be

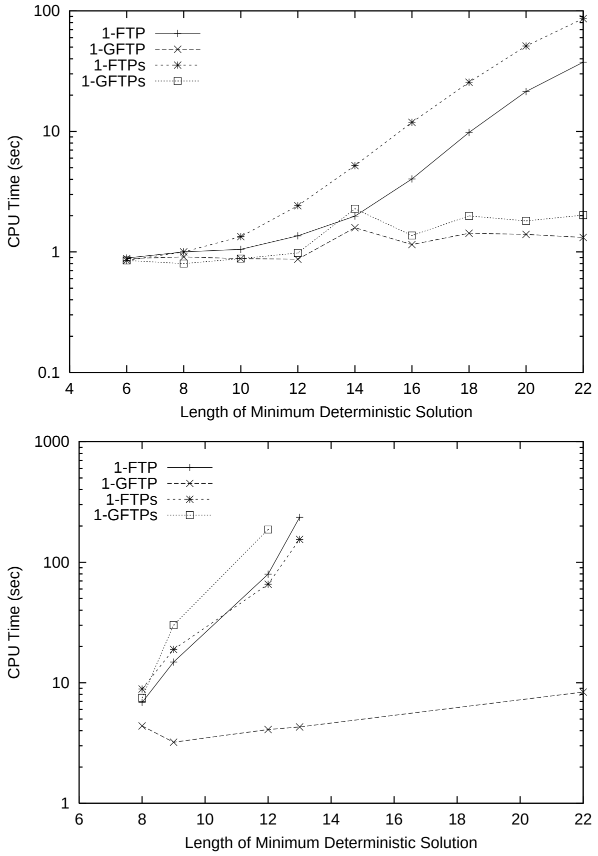


Figure 6: Results of the 8-Puzzle (top) and the SIDMAR experiments (bottom).

an advantage in general.

Conclusion

In this paper, 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 restrictive 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 more refined models of uncertainty in SNDP. A fruitful 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 faults.

References

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

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

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

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

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

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

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

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

ICAPS-03. 2003. Personal communication from anonymous ICAPS-03 referees.

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

Jensen, R. M. 2003a. *Efficient BDD-Based Planning for Non-Deterministic, Fault-Tolerant, and Adversarial Domains*. Ph.D. Dissertation, 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.itu.edu/people/rmj`.

Pecheur, C., and Simmons, R. 2000. From livingstone to SMV. In *FAABS*, 103–113.

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

Puterman, M. L. 1994. *Markov Decision Problems*. Wiley.

Thiébaux, S., and 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)*, 85–96.

Williams, B. C.; Ingham, M.; Chung, S. H.; and 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*, volume 9, 212–237.