

**Università
di Brescia**

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

**DOTTORATO DI RICERCA IN INGEGNERIA
DELL'INFORMAZIONE**

SSD: ING-INF/05

CICLO XXXVIII

**MODELS AND ALGORITHMS FOR RESILIENT
PLANNING UNDER ACTION FAILURES**

Dottorando:

Rovetta Alberto

Matricola 712511

Supervisore:

Prof. Alfonso E. Gerevini

Co Supervisor:

Dr. Diego Aineto

Prof. Enrico Scala

Prof. Ivan Serina

Abstract

Classical planning typically assumes that an agent’s actions are executed successfully as planned. However, in real-world scenarios, this assumption rarely holds: actions may fail in unpredictable ways that are not explicitly captured in the planning model. To address this challenge, this thesis introduces the *Resilient Planning problem*, whose goal is to ensure that an agent can achieve its objectives even in the presence of a bounded number of action failures.

Within this new problem, the RESPLAN algorithm is presented for the generation of *k-resilient plans*, that is, plans guaranteeing goal achievement as long as no more than k action failures occur during execution. Building upon this foundation, the thesis also proposes an enhanced version of RESPLAN that incorporates two main improvements. The first is a pruning strategy based on *landmarks*, which discards paths that are guaranteed to be inconsistent with the recovery constraints a priori. This approach reduces the search space, particularly in unsatisfiable instances where no resilient plan exists, by preventing the exploration of branches that cannot lead to a valid solution. The second introduces a *planning adaptation strategy* where regressed state formulas are used to identify resilient plan suffixes. This allows the search to reconnect to previously verified states, avoiding redundant planner calls and reducing the number of iterations to identify a resilient plan when one exists.

The proposed methods are experimentally evaluated against the original RESPLAN algorithm and approaches based on Fully Observable Non-Deterministic (FOND) planning, demonstrating substantial improvements in terms of efficiency and robustness.

Overall, the thesis advances the state of the art in robust automated planning, contributing to the development of agents capable of acting reliably in unpredictable environments subject to action failures.

Abstract

La pianificazione classica si basa tipicamente sull'ipotesi intrinseca che le azioni di un agente vengano eseguite con successo come pianificato. Tuttavia, negli scenari del mondo reale, questa ipotesi si rivela troppo ottimistica: le azioni possono fallire in modi imprevedibili che non sono esplicitamente definiti nel modello di pianificazione. Per affrontare questa sfida, questa tesi introduce il *problema della Pianificazione Resiliente (Resilient Planning problem)*, il cui obiettivo è garantire che un agente possa raggiungere i propri obiettivi anche in presenza di un numero limitato di fallimenti delle azioni.

All'interno di questo nuovo problema, viene presentato l'algoritmo RESPLAN per la generazione di *piani k -resilienti*, ovvero piani che garantiscono il raggiungimento dell'obiettivo a condizione che non si verifichino più di k fallimenti delle azioni durante l'esecuzione. Basandosi su queste fondamenta, la tesi propone anche una versione potenziata di RESPLAN che incorpora due miglioramenti principali. Il primo è una strategia di potatura basata sui *landmarks*, che scarta a priori i percorsi che non soddisfano le condizioni di ripristino. Questo approccio riduce lo spazio di ricerca, in particolare nelle istanze insoddisfacibili dove non esiste alcun piano resiliente, prevenendo l'esplorazione di rami che non possono portare a una soluzione valida. Il secondo introduce una *strategia di adattamento della pianificazione* in cui le formule di stato regredite vengono utilizzate per identifi-

care suffissi di piano resilienti. Ciò consente, durante la ricerca, di riconnettersi a stati precedentemente verificati, evitando chiamate ridondanti al pianificatore e riducendo il numero di iterazioni necessarie per identificare un piano resiliente, qualora ne esista uno.

I metodi proposti sono valutati sperimentalmente rispetto all'algoritmo RESPLAN originale e agli approcci basati sulla pianificazione Non Deterministica Completamente Osservabile (FOND), dimostrando miglioramenti sostanziali in termini di efficienza e robustezza.

Complessivamente, la tesi offre un contributo allo stato dell'arte della pianificazione automatica robusta, permettendo lo sviluppo di agenti capaci di agire con affidabilità in ambienti incerti e soggetti a errori di esecuzione.

Contents

1	Preface	1
1.1	Topic of the Thesis	1
1.2	Motivation and Research Questions	3
1.3	Structure of the Thesis	4
2	Automated Planning	6
2.1	Classical Planning	7
2.2	State of the Art	12
2.2.1	Fully Observable Non-Deterministic (FOND) Planning .	12
2.2.2	Fault-Tolerant Planning	13
2.2.3	Plan Repair and System Resilience	15
3	Resilient Planning Problem	17
3.1	Formal Definition	18
3.1.1	k -Resilient States	18
3.1.2	Illustrative Example of Resilient States	20
3.1.3	K -Resilient Plan	22
3.1.4	Illustrative Example of a k -Resilient Plan	23
3.2	Relation among Resilient Planning, FT Planning and FOND . . .	24
3.3	Theoretical Properties	26

4	ResPlan Algorithm	30
4.1	RESPLAN: Algorithm Overview	31
4.1.1	Augmented Search Space	31
4.1.2	Operational Flow	34
4.1.3	Plan Extraction	36
4.1.4	Theoretical Properties	37
4.2	ResPlan Workflow Example	38
4.2.1	Initialization	38
4.2.2	First Iteration	39
4.2.3	Second and Third Iterations	40
4.2.4	Fourth Iteration	41
4.2.5	Proving that states $\mathcal{D}$, $\mathcal{B}$, $\mathcal{A}$ are 2-resilient	42
4.2.6	Generating the 2-resilient plan	44
4.3	RESPLAN Limitations	45
5	Enhanced RESPLAN: RESPLAN2	47
5.1	Landmarks Pruning Technique	48
5.1.1	Definition of Landmarks	48
5.1.2	Connecting Landmarks to Non-Resilient States	49
5.1.3	Example Landmark Pruning Technique	51
5.2	Adaptive Replanning Technique	51
5.2.1	Resilience Certificates	52
5.2.2	Example Adaptive Replanning Technique	54
5.3	RESPLAN2: Algorithm Overview	56
5.3.1	Operational Flow	56
5.3.2	Plan Extraction	64
5.4	Resilient Planning Policy	65

5.4.1	Policy Definition	65
5.4.2	Extended Certificates for Policy Synthesis	67
5.4.3	Policy Generation	71
5.4.4	Policy Validation	73
5.5	Theoretical Properties	75
6	Experimental Evaluation	77
6.1	Introduction to the Results	77
6.2	Comparison among RESPLAN2, RESPLAN and FOND	79
6.2.1	Coverage results	79
6.2.2	Runtime results	81
6.2.3	Cumulative Coverage over Time	83
6.2.4	Ablation Study	85
6.2.5	Summary	86
6.3	Landmark Structural Analysis	87
6.4	Computational Overhead and Runtime Distribution	89
6.5	Solution Quality Analysis	91
6.6	Policy Analysis	94
6.7	Evaluating Resilience of IPC Benchmarks	96
7	Conclusions	99

Chapter 1

Preface

This thesis illustrates the study and work carried out at the Information Engineering Department of Università degli Studi di Brescia during my Ph.D. in Information Engineering, curriculum Computer Science / Engineering and Control Systems, cohort XXXVIII. My work was done under the supervision of Prof. Alfonso Emilio Gerevini, Dr. Diego Aineto, Prof. Enrico Scala and Prof. Ivan Serina at Università degli Studi di Brescia. This thesis builds upon and extends the results presented in our previous works published in [1] and [32].

1.1 Topic of the Thesis

In classical planning, the objective is to find a sequence of actions (a plan) whose execution, starting from a given initial state, leads to a state satisfying the problem goal [12, 15]. Such plans are proven correct with respect to an abstract model of actions and states. However, when these plans are executed in the real world, actions may fail in unpredictable ways that are not explicitly represented in the model [21]. These failures may arise from exogenous events, hardware malfunc-

tions, missing resources, or incomplete or incorrect action specifications. As a consequence, even plans that are valid at the abstract level may not guarantee goal achievement at execution time.

In automated planning, a typical way to handle action failures is interleaving plan execution and replanning from the state where a failure occurs [15], possibly by repairing the current plan instead of replanning from scratch (e.g., [3, 11, 14, 34]). This strategy, however, does not always ensure that a new plan achieving the original goal can be found, since after a failure the system may reach a state from which the goal is no longer attainable. To address this limitation, the *Resilient Planning problem* has been introduced. Its objective is to generate plans that are robust to execution failures, ensuring that the agent can still reach the goal as long as the number of failures remains below a given bound k . Plans satisfying this property are called *k-resilient plans*.

The first algorithmic solution to this problem, named RESPLAN, computes resilient plans by identifying *k-resilient states*, states that a resilient plan can safely visit and from which replanning is always possible within the failure bound. While effective, the algorithm faces two major limitations: the high computational cost of proving that a state is resilient, and the need to exhaustively explore all possible alternatives to confirm that a state is not resilient.

To overcome these limitations, this thesis proposes an enhanced version of RESPLAN, introducing the RESPLAN2 algorithm that includes two major improvements. The first incorporates *landmarks* into a pruning strategy that significantly reduces unnecessary search and improves efficiency, especially in unsolvable instances. The second introduces a *planning adaptation strategy* based on regressed state formulas, which guides the (re)planning process and reduces the number of iterations required when a resilient plan exists.

Resilient Planning is closely related to *Fully Observable Non-Deterministic*

(FOND) [7, 6, 12] planning and to *Fault Tolerant* (FT) planning [8, 22]. In particular, Resilient Planning can be reformulated as a special case of FOND planning, where each action has two possible outcomes: either its nominal effects or an empty set representing a failure. However, this compilation approach is often computationally expensive, as it increases the branching factor and search complexity.

Experimental evaluations compare RESPLAN and its enhanced version against FOND-based compilations across a variety of classical planning benchmarks. The results show substantial improvements in terms of scalability, efficiency, and problem coverage, confirming the effectiveness of the proposed techniques. Overall, this work advances the state of the art in resilient automated planning, providing new methods for developing intelligent agents capable of acting reliably in dynamic and partially unpredictable environments.

1.2 Motivation and Research Questions

The motivation behind this research stems from the gap between theoretical planning models and their applicability in real-world scenarios. Classical planning assumes that all actions succeed as expected, but in practical applications this assumption rarely holds. Even a single unexpected failure can invalidate a plan and prevent the achievement of the goal. This limitation motivates the need for planning approaches that explicitly account for potential action failures and provide guarantees of goal reachability under bounded uncertainty.

To address this challenge, we focus on the Resilient Planning problem, which guarantees that an agent can achieve its goals even in the presence of a bounded number of action failures during execution. The problem arises when action failures are unpredictable and not explicitly modeled, such as when actions like

moving a vehicle between two locations or refueling at a gas station fail due to unforeseen circumstances like road closures or station outages. We aim to ensure that plans can still succeed despite such disruptions.

The main research questions addressed in this thesis can be summarized as follows:

- How can we formally define and reason about resilience in classical planning problems?
- How can we design algorithms capable of generating resilient plans?

Addressing these questions is essential for bridging the gap between abstract planning theory and the development of autonomous agents that can act reliably in dynamic, partially unpredictable environments.

1.3 Structure of the Thesis

In this section, we present a brief overview of the structure of the content set forth in this thesis.

In Chapter 2, we provide a brief introduction to Automated Planning. We first introduce the basics of Classical Planning, defining the formal model and the assumptions of deterministic environments. Then, we discuss the challenges of acting under uncertainty and the limitations of classical approaches in real-world scenarios. Finally, we describe the current state-of-the-art for robust and fault-tolerant planning, focusing on Fully Observable Non-Deterministic (FOND) planning.

Chapter 3 formally introduces the *Resilient Planning* problem, which is the core theoretical contribution of this work. We provide a strict definition of k -resilience and explain the recursive properties of resilient states. We also illus-

trate the differences and common points between our framework and existing non-deterministic planning models.

The algorithmic contributions of this thesis are reported in Chapters 4 and 5, which describe respectively the RESPLAN algorithm and the enhanced RESPLAN2 system.

First, in Chapter 4, we introduce the first complete solution for generating k -resilient plans. We describe the search space augmentation and the operational flow of the RESPLAN algorithm. In Section 4.2, we report a detailed workflow example based on a navigation domain to show how the algorithm identifies k -resilient states.

In Chapter 5, we address the computational bottlenecks of the initial approach by introducing RESPLAN2. We describe two novel techniques: *Landmark-based Pruning* (Section 5.1), which identifies non-resilient states early in the search, and *Adaptive Planning* (Section 5.2), which exploits resilience certificates to reuse search knowledge. Subsequently, we extend the framework to support the generation of *Resilient Policies* (Section 5.4). We describe the mechanism to extract the policy strategy from resilience certificates and present a formal *Policy Validator* to ensure the correctness of the generated strategies.

The experimental evaluation in Chapter 6 shows the performance gains in terms of scalability and runtime efficiency between RESPLAN, RESPLAN2, and the current state of the art. We study the impact of the enhancements present in RESPLAN2 with an ablation study. Finally, we provide an analysis of the solutions obtained.

Lastly, in Chapter 7, we summarize the main results of our research and propose several directions for future work aimed at extending resilient planning.

Chapter 2

Automated Planning

One of the core problems of Artificial Intelligence (AI) is the problem of intelligent behaviour, that is, the capability of using one's knowledge about the world to make decisions and act effectively in novel situations. In particular, we are interested in selecting the sequence of actions that an autonomous agent has to perform in order to achieve a desired goal in a given environment. In the literature, three main approaches have been proposed to address this problem:

- **programming-based approach:** the agent's behaviour is explicitly encoded by the programmer through a predefined set of rules or procedures that determine which actions to execute in each situation;
- **learning-based approach:** the agent learns how to act from experience, typically through Machine Learning techniques such as Deep Learning or Reinforcement Learning;
- **model-based approach:** the agent relies on an abstract model of actions, states, and goals to automatically compute the best course of action through reasoning.

Automated Planning is the branch of AI that formalizes this model-based approach to action selection. The goal of a planning system is to automatically generate a sequence of actions (a *plan*) that transforms an initial state of the world into one that satisfies a given goal condition. Such plans are typically validated against a formal and abstract model of the domain.

Model-based approaches to planning are generally composed of three key components:

- **the model**, which defines the dynamics of the environment, the agent's actions, and its objectives;
- **the representation languages**, which allow the model to be specified in a compact and expressive form;
- **the algorithms**, which use these representations to automatically compute valid and efficient plans.

However, in real-world applications, the execution of a plan cannot always be assumed to succeed as expected: actions may fail due to unpredictable conditions such as exogenous events, hardware malfunctions, or incomplete domain models. The next section presents the current state of the art in this field and outlines the approaches that inspired the work developed in this thesis.

2.1 Classical Planning

We focus on the branch of Automated Planning known as *Classical Planning*, where the environment is deterministic and both the initial state and goals are fully known. The model underlying classical planning can be described as a model $\mathcal{S} = \langle S, s_0, S_G, A, f, c \rangle$, where:

- S is a finite and discrete set of possible states. These states are a representation of all the possible configurations of the environment;
- $s_0 \in S$ is the initial state (i.e. the initial configuration of the environment);
- $S_G \subseteq S$ is the non-empty set of goal states;
- $A(s) \subseteq A$ is the set of actions in A that are applicable in each state $s \in S$;
- $f(a, s)$ is a deterministic transition function where $s' = f(a, s)$ is the state that follows s after applying action $a \in A(s)$;
- $c(a, s)$ is a positive cost for executing action a in the state s .

A solution or *plan* π is a sequence of applicable actions $\pi = a_0, a_1, \dots, a_n$ that generates a state sequence $s_0, s_1, \dots, s_{n+1}$ where $a_i \in A(s_i)$, $s_i \in S$, $s_{i+1} = f(a_i, s_i)$ and s_{n+1} is a goal state (i.e. $s_{n+1} \in S_G$). The cost of the plan is defined as:

$$\text{cost}(\pi) = \sum_{i=0}^n c(a_i, s_i) \quad (2.1)$$

That is the sum of the action costs $c(a_i, s_i)$ with $i \in [0, n]$. A solution plan is optimal if it has minimum cost among all possible solution plans. A common cost structure consists in setting all action costs $c(a, s)$ to 1. In this case the cost of the plan is given by its length, and the optimal plans are the shortest ones.

There are two types of language for expressing classical planning models in a compact form: in the first, the state variables are all boolean while in the other they are multivalued and can be represented as a value from a finite domain [15]. The simplest classical planning language in use is STRIPS [10], a language based on boolean variables. In STRIPS, the boolean variables that compose a state can be called *facts*, *fluents* or *atoms*. A planning problem in STRIPS is represented by a tuple $\Pi = \langle F, A, s_0, G \rangle$ where:

```

(define (domain blocksworld)
  (:requirements :strips :equality)
  (:predicates (clear ?x) (on-table ?x) (arm-empty) (holding ?x) (on ?x ?y))
  (:action pick-up
    :parameters (?ob)
    :precondition (and (clear ?ob) (on-table ?ob) (arm-empty))
    :effect (and (holding ?ob) (not (clear ?ob)) (not (on-table ?ob))
                 (not (arm-empty))))
  (:action put-down
    :parameters (?ob)
    :precondition (and (holding ?ob))
    :effect (and (clear ?ob) (arm-empty) (on-table ?ob)
                 (not (holding ?ob))))
  (:action stack
    :parameters (?ob ?underob)
    :precondition (and (clear ?underob) (holding ?ob))
    :effect (and (arm-empty) (clear ?ob) (on ?ob ?underob)
                 (not (clear ?underob)) (not (holding ?ob))))
  (:action unstack
    :parameters (?ob ?underob)
    :precondition (and (on ?ob ?underob) (clear ?ob) (arm-empty))
    :effect (and (holding ?ob) (clear ?underob)
                 (not (on ?ob ?underob)) (not (clear ?ob)) (not (arm-empty))))
  )
  (define (problem pb2)
    (:domain blocksworld)
    (:objects block_a block_b)
    (:init (on-table block_a) (on-table block_b) (clear block_a)
           (clear block_b) (arm-empty))
    (:goal (and (on block_a block_b)))
  )
)

```

Figure 2.1: An instance of the BLOCKSWORLD domain expressed in PDDL

- F represents the set of atoms or propositions of interest;
- A represents the set of actions;
- $s_0 \subseteq F$ represents the initial state;
- $G \subseteq F$ represents the goal.

In STRIPS, the actions $a \in A$ are represented by three sets of atoms over F called the Add ($Add(a)$), Delete ($Del(a)$) and Precondition ($Pre(a)$) lists. The Add list describes the atoms that the action a makes true, the Delete list describes the

atoms that a makes false and the Precondition list describes the atoms that must be true in order for the action a to be executed.

Planning Domain Definition Language (PDDL) [26] is a language that extends the STRIPS language by adding a number of syntactic constructs, like, for instance, allowing preconditions and goals to contain negative literals. In Figure 2.1, we report the description of the well-known BLOCKSWORLD domain, in which the agent has the goal of building one or more stacks of blocks, and only one block may be moved at a time, expressed in PDDL. As we can see in Figure 2.1, PDDL problems are expressed in two parts enclosed in a “define” clause:

- the **domain** part: it represents the general domain. In this part the actions are expressed using generic atoms defined using predicate names, variables and constants;
- the **problem** part: it represents a particular domain instance. In this part, the object names that will replace the variables are declared, together with the atoms used to describe the initial state, and a formula describing the goal state.

As we can see in Figure 2.1, in the BLOCKSWORLD domain the predicates are `clear`, `on-table`, `holding`, `arm-empty` and `on`. The predicates `clear`, `on-table` and `holding` represent the property that the objects (i.e. the blocks) can have; the objects are expressed through the use of variables (e.g. `?x`). The predicate `arm-empty` is a property of the environment as it does not involve any object while the predicate `on` expresses the relation between two objects. Following the predicates definition we have four actions: `Pick-Up`, `Put-Down`, `Stack` and `Unstack`. `Pick-Up` is the action of lifting a block, represented by the variable `?ob`, from a table; similarly, `Put-Down` is the action of dropping a block on the table. The action `Unstack` concerns lifting a block `?ob` from the

top of another block ?underob; the opposite action of dropping ?ob on the top of ?underob is represented by action Stack.

Finally, in the last define clause of Figure 2.1, we have the problem definition. In this very simple problem instance there are two blocks: Block_A and Block_B. In the initial state both blocks are on the table and the goal of the problem is to reach a state where Block_A is on top of Block_B.

While PDDL is the standard for modeling, most modern planners do not solve the problem directly in its propositional form. Instead, they translate the PDDL representation into a more computationally efficient formalism known as SAS+ (Simplified Action Structures) [4] [18].

The SAS+ formalism uses multi-valued state variables instead of boolean atoms. A planning task in SAS+ is defined by a tuple $\Pi = \langle V, A, s_0, G \rangle$, where:

- $V = \{v_1, \dots, v_n\}$ is a set of state variables, each associated with a finite domain D_i . A state s is a complete assignment that maps each variable v_i to a single value in D_i ;
- A is a set of actions, where each $a = \langle pre(a), eff(a) \rangle$ consists of partial assignments to V ;
- s_0 is the initial complete assignment, and G is the goal partial assignment.

The primary advantage of SAS+ is its ability to encode structural mutual exclusions implicitly. For example, in the BLOCKSWORLD domain, the status of a robot arm, which in STRIPS requires multiple atoms like arm-empty and holding(b1), can be represented by a single SAS+ variable v_{arm} with a domain $D_{arm} = \{empty, b_1, b_2, \dots\}$. This representation significantly reduces the state space and allows planners to exploit value dependencies through structures like the Causal Graph [17], which is essential for modern heuristic calculation.

2.2 State of the Art

The study of planning under uncertainty has produced several extensions of the classical deterministic framework to explicitly handle cases in which actions may not behave as expected. Among these, two research directions are particularly relevant to this work: *Fully Observable Non-Deterministic* (FOND) planning and *Fault-Tolerant* (FT) planning. Both aim to guarantee goal achievement despite execution failures, although they differ in the way uncertainty is represented and managed.

2.2.1 Fully Observable Non-Deterministic (FOND) Planning

FOND planning extends classical planning by allowing actions to have multiple possible outcomes. In this setting, the agent always has complete knowledge of the current state of the world, but cannot predict with certainty the effect of its actions. The objective is therefore to synthesize a policy, a mapping from states to actions, that guarantees goal achievement regardless of how the non-determinism unfolds.

Formally, a FOND planning problem can be defined as a tuple $\langle F, A, s_0, G \rangle$, where F is the set of fluents, A the set of actions, $s_0 \subseteq F$ the initial state, and $G \subseteq F$ the goal. Each action $a \in A$ is described by a set of preconditions $Pre(a)$ and a set of possible effect sets $Eff(a) = \{E_1, E_2, \dots, E_n\}$, each representing one of the possible outcomes. When a is executed in a state that satisfies $Pre(a)$, one of these outcomes is applied, leading to a successor state.

Early symbolic planners such as MBP [6] and NDP [7] used model checking and Binary Decision Diagrams (BDDs) to compute strong or strong-cyclic policies. More recent systems, including MyND [25], PRP [29], and FOND-SAT [13], exploit heuristic or SAT-based search to improve scalability.

In recent years, the field has seen a shift toward native heuristic search and more efficient SAT encodings. Notable advancements include the PALADINVS planner [30], which introduces a native approach for FOND search, and the work of Messa and Grahl Pereira [27], who provided a formal foundation for native heuristic search in non-deterministic environments. Additionally, modern SAT-based techniques such as FOND-FOR-ALL [9] have further extended the state of the art by providing complete search strategies. While these planners are highly expressive, they remain computationally demanding as they must reason over the entire set of possible non-deterministic outcomes.

2.2.2 Fault-Tolerant Planning

Fault-Tolerant (FT) planning represents a formalism that interpolates between classical and contingent planning. While contingent planning adopts "extreme pessimism" by requiring a plan to guarantee the goal under any realization of action outcomes and classical planning adopts "extreme optimism" by assuming only deterministic intended effects, the FT model requires plans to be provably robust for up to a bounded number κ of arbitrary operator faults during plan execution.

Following the formalization by Domshlak [8], given a contingent planning task $\Pi = \langle F, A, s_0, G \rangle$, the FT model introduces an *exception model* $\mathcal{F}$. An exception model is a function $\mathcal{F} : \bigcup_{a \in A} \text{eff}(a) \rightarrow \mathbb{N}$ that maps each possible effect of each operator to a number of "exceptions", or unintended artifacts, associated with that effect. In particular:

- The operator effects e for which $\mathcal{F}(e) = 0$ correspond to the *primary effects* of the respective operators.
- An exception model is called *normative* if every operator is associated with

at least one primary effect.

- A model is *1-primary* if each operator has at most one primary effect.

Let $\rho = (s_0, e_0, s_1, e_1, \dots)$ be an execution of a policy π . In this formalization, the explicit inclusion of the effect e_i is necessary because the transition to the next state s_{i+1} alone does not uniquely identify the number of exceptions encountered. In the FT model, different effects of the same operator might lead to the same successor state s_{i+1} while carrying different exception values $\mathcal{F}(e_i)$. The total number of exceptions encountered is defined as $\mathcal{F}(\rho) = \sum_{i=0}^{\infty} \mathcal{F}(e_i)$.

As noted by Jensen et al. [22], a non-stationary κ -plan can be captured as a stationary policy for an $(\mathcal{F}, \kappa)$ -reformulation of the original task. In this reformulation, the state space is augmented to include the accumulated count of failures:

- **States:** $S^{(\mathcal{F}, \kappa)} = S \times \{0, \dots, \kappa\}$, representing world states and the number of exceptions so far.
- **Initial State:** $s_0^{(\mathcal{F}, \kappa)} = (s_0, 0)$.
- **Goal States:** $S_G^{(\mathcal{F}, \kappa)} = \{(s, i) \mid s \in S_G, 0 \leq i \leq \kappa\}$.

For each state (s, i) and action a applicable in s , applying a leads to a new state (s', i') defined by the effect $e \in \text{eff}(a)$ as:

$$\text{Res}[(s, i); e] = (\text{Res}[s; e], i + \mathcal{F}(e)) \quad (2.2)$$

provided that $i + \mathcal{F}(e) \leq \kappa$. If an effect would push the counter beyond κ , the transition is ignored as the execution would no longer be κ -admissible. It has been proven that if the model $\mathcal{F}$ is normative, a stationary κ -plan always exists for any solvable FT task.

The $(\mathcal{F}, \kappa)$ -reformulation can be directly viewed as a *Fully Observable Non-Deterministic* (FOND) planning task. In this setting, the non-determinism is restricted by the fault budget κ : an operator o in an augmented state (s, i) remains non-deterministic, but its possible outcomes are limited to those effects e such that $i + \mathcal{F}(e) \leq \kappa$.

As shown by Domshlak [8], if the exception model is *1-primary*, the FT task exhibits appealing solution structures. The compilation process leverages the fact that FT-planning with 1-primary models is in PSPACE, making it no harder than classical planning in the worst case. Specifically, this compilation involves:

- Mapping each augmented state (s, i) to a state in the reformulation.
- Encoding the counter i within the state variables to keep track of the remaining “fault budget”.
- Defining the transition function such that it branches between primary and exceptional effects while pruning any branch that exceeds κ .

This compilation is particularly effective because it allows for exploiting various search techniques to find policies that are robust for up to κ operator faults.

2.2.3 Plan Repair and System Resilience

Beyond the boundaries of non-deterministic and fault-tolerant planning, the problem of handling execution failures has been extensively studied in the context of Plan Repair. Unlike the proactive approach of FOND or FT planning, which accounts for contingencies during the synthesis phase, plan repair focuses on reactive strategies to modify a failed plan locally instead of replanning from scratch.

A foundational contribution in this area is provided by Heger and Singh [16], who demonstrate how combining contingency planning with local plan repair

and re-planning can significantly enhance the robustness of robotic assembly tasks. Their work emphasizes the importance of "re-stitching" a plan after an unexpected event, a concept that shares similarities with the recovery strategies often emerging in resilient planning. These reactive frameworks are often integrated into modern robotics architectures, such as PlanSys2 [24], to manage the gap between high-level logical reasoning and physical execution.

Furthermore, the concept of resilience is a cornerstone in other engineering and scientific disciplines, such as Fault-Tolerant Control (FTC). In control theory, FTC aims to maintain acceptable system stability and performance despite the presence of component failures, such as sensor malfunctions or actuator faults [2]. In these contexts, resilience is often achieved through structural redundancy, allowing the system to reconfigure its behavior to compensate for the loss of specific capabilities.

From a more theoretical standpoint, the transitions from operational to failure states can be analyzed through the lens of Catastrophe Theory [23]. This mathematical framework provides a way of modeling sudden, discontinuous shifts in system behavior resulting from small, continuous changes in external parameters. While Resilient Planning operates within discrete state-spaces, this multidisciplinary perspective, which even finds applications in modeling infrastructure resilience during natural disasters [33], underscores the importance of structural "safety margin". These are core elements that our framework formalizes within the automated planning paradigm to prevent a system from crossing the threshold into an unrecoverable failure state.

Chapter 3

Resilient Planning Problem

The approaches discussed in the previous chapter, namely Fully Observable Non-Deterministic (FOND) and Fault-Tolerant (FT) planning, allow modeling environments with multiple action outcomes. However, these formalisms require explicitly modeling every possible failure outcome at the formulation level. This means the designer must anticipate and represent all potential ways an action can fail, which is often impractical in complex real-world scenarios.

To address this, this thesis introduces the *Resilient Planning Problem*. The core advantage of this formulation is that it does not require explicit modeling of failure effects. Instead, it accounts for the possibility of failure without needing to anticipate its specific consequences. Specifically, by assuming that the state remains unchanged after a failure, the resilient planning framework can support any failure caused by exogenous variables that are outside our world representation.

Under this formulation, resilience is achieved within a deterministic framework, based on two key assumptions:

- When an action fails, the state of the environment remains unchanged;
- Once an action fails, it cannot be executed again in the same plan execution.

3.1 Formal Definition

Formally, a *Resilient Planning Problem* is defined as an extension of the classical planning framework presented in Section 2.1, augmented with a non-negative integer K representing the maximum number of action failures that a resilient plan is guaranteed to recover from.

A Resilient Planning Problem is a pair $\langle \Pi, K \rangle$, where $\Pi = \langle F, A, s_0, G \rangle$ is a classical planning problem, with F , A , s_0 , and G representing respectively the set of facts, actions, initial state, and goal condition.

The objective is to compute a plan that guarantees goal achievement even if up to K actions fail during execution.

The remainder of this section introduces the formal definitions of resilience at the level of states and plans, following the recursive formulation proposed by Aineto et al. [1].

3.1.1 k -Resilient States

The formalization of Resilient Planning and its solutions is based on the notion of *resilient states*. Given a Resilient Planning Problem $\langle \Pi, K \rangle$, a state s is defined as *k-resilient* (for $k \leq K$) if the goal state G can still be achieved from s even when up to k actions fail during execution.

Intuitively, resilience expresses the ability of a state to guarantee goal reachability under a bounded number of failures, where actions that fail become permanently unavailable. Formally, resilience is defined recursively by considering both the successful and failed outcomes of applicable actions. We denote by $A(s)$ the set of actions applicable in state s , and by $s[a]$ the successor state obtained by applying $a \in A(s)$ in s .

Definition 1. Let $\Pi = \langle F, A, s, G \rangle$ be a planning problem, and let $S = 2^F$ denote the state space induced by F . A state $s \in S$ is said to be k -resilient if the following conditions hold:

- **Case 1:** If $s \supseteq G$, then s is k -resilient for all $k \geq 0$.
- **Case 2:** If $k = 0$, then s is 0-resilient if and only if there exists an action $a \in A(s)$ such that the successor state $s[a]$ is 0-resilient. This coincides with the classical notion of solvability: there exists a classical plan π from s that achieves the goal G .
- **Case 3:** For $k \geq 1$, s is k -resilient if and only if there exists an action $a \in A(s)$ such that:
 - $s[a]$ is k -resilient, and
 - s is $(k-1)$ -resilient in $\langle F, A \setminus \{a\}, s, G \rangle$; this means that state s remains $(k-1)$ -resilient even if the failed action a is removed from the set of available actions A .

This recursive definition captures the two possible outcomes of executing an action: if the action succeeds, the successor state $s[a]$ must remain k -resilient; if the action fails, the current state remains unchanged and the failed action is permanently removed from the available set, reducing the remaining failure budget

by one. A state is therefore k -resilient if it can still guarantee goal achievement after any sequence of at most k such failures.

Resilience is not an intrinsic or static property of a state $s \in S$. Instead, it is strictly functional to the specific goal G defined by the planning problem. Formally, a state s may be k -resilient with respect to a goal G , but only k' -resilient (where $k' < k$) or even non-resilient when a different goal G' is considered. This dependency arises because the existence of the alternative trajectories required by Definition 1 is determined by the reachability of the target configuration G under action failures.

3.1.2 Illustrative Example of Resilient States

To clarify the notion of state resilience, we illustrate it through a simple navigation problem. The domain shown in Figure 3.1 consists of seven locations (A–G) connected by different means of transportation: roads (solid lines), railways (dashed lines), and flights (dotted lines). The agent starts at location A and must reach the goal location G. Each connection corresponds to an action that moves the agent between two states. We can now analyse the resilience of each location by reasoning backward from the goal.

0-resilient states: A state is *0-resilient* if there exists at least one plan leading from it to the goal. Equivalently, the corresponding classical planning problem is solvable from that state. In Figure 3.2, for example, location F can reach G directly through $\text{train}(F, G)$, but if that action fails, the goal becomes unreachable from F. Therefore, state F is only 0-resilient, as it cannot tolerate any failures, while G is trivially k -resilient for all k .

1-resilient states. For a state to be k -resilient with $k \geq 1$, it must satisfy two conditions: (i) there exists an applicable action that leads to a k -resilient suc-

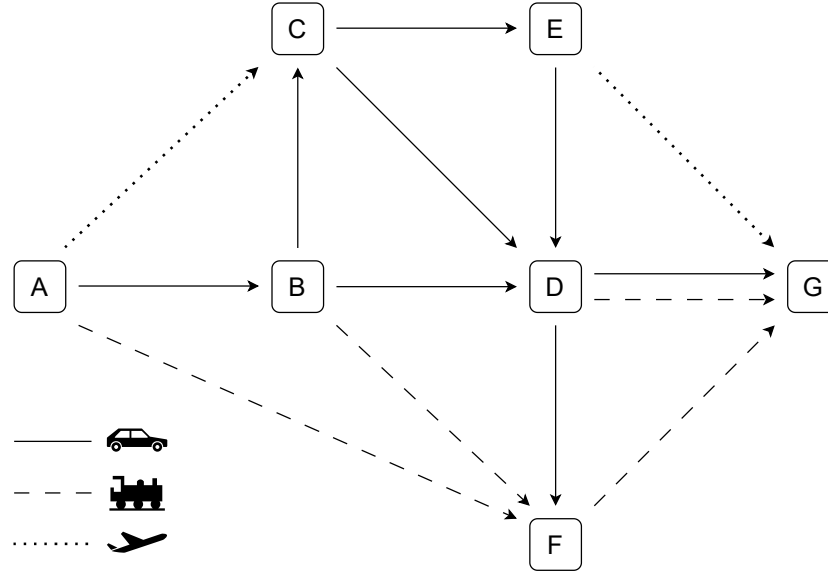


Figure 3.1: Navigation domain with locations and available actions. The goal is to move from A to G using different transport modalities (road, train, flight).

cessor, and (ii) if that action fails, the current state remains $(k-1)$ -resilient once that action is removed from the available actions. Following this logic, in the Figure 3.3, state E is 1-resilient, since it provides a backup route to recover from a single failure: if $\text{plane}(E, G)$ fails, the agent can still reach the goal via $\text{car}(E, D)$ and $\text{car}(D, G)$.

2-resilient states. In Figure 3.4, state D is 2-resilient, as it provides enough redundant ways to reach the goal even after two action failures. If both $\text{car}(D, G)$ and $\text{train}(D, G)$ fail, the agent can still reach G through the alternative route $\text{car}(D, F)$ followed by $\text{train}(F, G)$. Hence, state D ensures goal reachability under up to two execution failures.

This example illustrates how resilience emerges from structural redundancy in the action space: states with multiple independent paths to the goal can tolerate a higher number of execution failures. The recursive definition of k -resilience

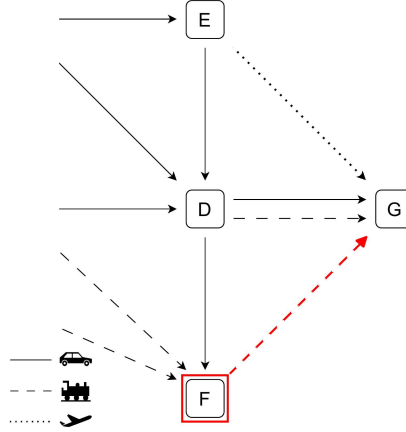


Figure 3.2: Identification of 0-resilient states. States in red (here, F) have a single available action leading to the goal and cannot tolerate any failures.

naturally captures the interplay between plan redundancy and bounded failure tolerance.

3.1.3 K -Resilient Plan

The concept of a K -resilient plan builds directly upon the definition of K -resilient states.

Definition 2 (K -Resilient Plan). *Given a planning problem $\Pi = \langle F, A, s_0, G \rangle$, a solution plan $\pi = \langle a_1, \dots, a_n \rangle$ is K -resilient if every state s_i in its induced trajectory $\tau = \langle s_0, s_1, \dots, s_n \rangle$ is a K -resilient state.*

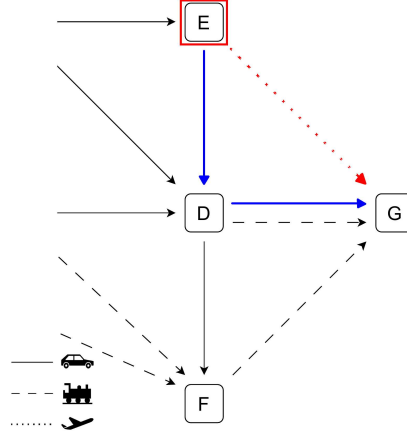


Figure 3.3: Identification of 1-resilient states. States in red (here, E) have two alternative plans that can reach the goal and can tolerate one action failure.

By definition, a K -resilient plan guarantees that even if up to K action failures occur during execution, there exists a recovery strategy from any intermediate state to reach the goal.

3.1.4 Illustrative Example of a k -Resilient Plan

To further clarify the concept of resilience, we now illustrate how the notion of k -resilient states can be used to identify k -resilient plans. Continuing from the navigation domain introduced in Section 3.1.2, consider again the problem of reaching location G from A. The objective is to determine a resilient plan that ensures goal achievement even in the presence of execution failures.

A plan is k -resilient if it remains able to achieve the goal even when up to k actions fail during execution. In this example, the plan

$$\pi = (\text{car}(A, B), \text{car}(B, D), \text{car}(D, G))$$

is a 2-resilient plan because all the states along its trajectory are at least 2-

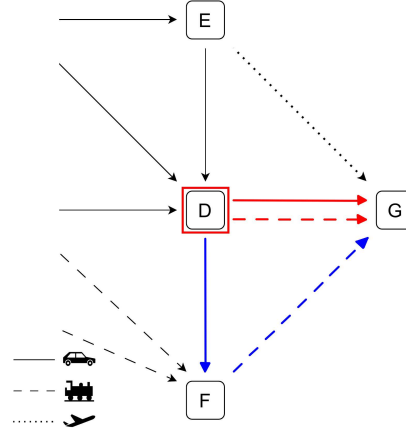


Figure 3.4: Identification of 2-resilient states. States in red (here, D) have three alternative plans that can reach the goal and can tolerate up to two action failures.

resilient. Even if two actions fail, alternative routes exist that allow the agent to still reach the goal.

This example highlights how resilience in planning emerges from the presence of redundant, independent subplans that preserve goal reachability even when part of the plan fails. The concept of a k -resilient plan thus extends classical planning by embedding robustness directly into the plan structure.

3.2 Relation among Resilient Planning, FT Planning and FOND

The Resilient Planning Problem can be naturally positioned within the broader frameworks of Fault-Tolerant (FT) and Fully Observable Non-Deterministic (FOND) planning. In this context, resilient planning can be viewed as a specific instance of FT planning where action failures are modeled as null outcomes: if a failure occurs, the state remains unchanged, but the action becomes unavailable for future use.

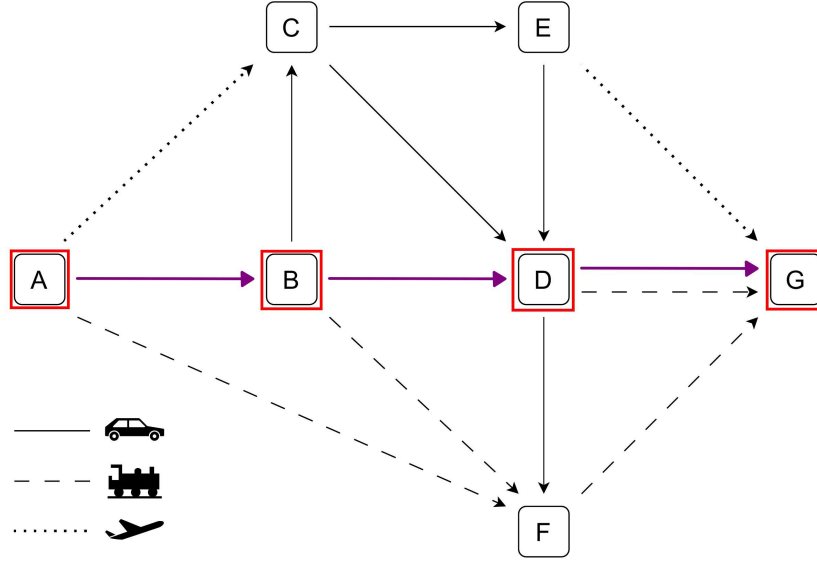


Figure 3.5: Example of a 2-resilient plan, represented by the purple edges.

Formally, this relationship is established by considering each deterministic action as an action a within a contingent planning task $\Pi = \langle F, A, s_0, G \rangle$. In the resilient planning, the primary effect e (where $\mathcal{F}(e) = 0$) represents the successful execution of the action, while the exceptional effects (where $\mathcal{F}(e) > 0$) represent the failure. This aligns with the 1-primary normative exception model defined by Domshlak [8], where each action is associated with exactly one intended primary effect and at least one faulty outcome.

The resilience parameter K , which bounds the number of allowed failures, directly corresponds to the fault tolerance bound κ in the FT model. Consequently, a resilient plan is equivalent to a κ -plan, as it must guarantee goal reachability along all κ -admissible executions, those where the total count of exceptions $\mathcal{F}(\rho)$ does not exceed κ .

Furthermore, any FT planning task under a 1-primary model can be systematically compiled into a FOND problem through an $(\mathcal{F}, \kappa)$ -reformulation.

Consequently, the following relationship holds:

$$\text{Resilient Planning} \subseteq \text{Fault-Tolerant Planning} \subseteq \text{FOND}.$$

This hierarchy makes explicit that resilience represents a bounded form of fault tolerance, and that both can be operationalized within the general framework of non-deterministic planning.

3.3 Theoretical Properties

Two key properties naturally arise from the previous definitions and provide useful insight into the structure of resilient planning problems. These properties can be formalized as follows:

Theorem 1 (Existence of a Resilient Plan). *The resilient planning problem $\langle \Pi, K \rangle$, where $\Pi = \langle F, A, s_0, G \rangle$, has a solution if and only if the initial state s_0 is K -resilient.*

Proof. The proof is structured by showing both the necessity and sufficiency of the condition.

Necessity. Suppose the resilient planning problem $\langle \Pi, K \rangle$ has a solution. According to the Definition 2, there exists a K -resilient plan π for Π . This plan induces a state trajectory $(s_0, s_1, \dots, s_n)$ such that $s_n \models G$ and, by the properties of resilient plans, every state s_i in the trajectory (for $0 \leq i \leq n$) must be K -resilient. In particular, this implies that the initial state s_0 is K -resilient.

Sufficiency. Suppose that the initial state s_0 is K -resilient. According to Definition 1, the K -resilience of s_0 implies the existence of an action $a \in A$ such that the resulting state s_1 is also K -resilient, and the state remains $(K - 1)$ -resilient

even if a fails. By induction on the minimum distance to a goal state within the K -resilient states, well-defined since the state space is finite, there exists a trajectory $\tau = \langle s_0, s_1, \dots, s_n \rangle$ from the initial state to a goal state $s_n \models G$ where every intermediate state s_i is K -resilient. By Definition 2, such a trajectory corresponds to a K -resilient plan π for Π , which constitutes a solution to the resilient planning problem $\langle \Pi, K \rangle$.

Therefore, a k -resilient plan exists if and only if s_0 is k -resilient, completing the proof. $\square$

Theorem 2 (Monotonicity of Resilience). *For any state $s \in S$ and any $k \geq 0$, if s is k -resilient, then s is also k' -resilient for every $0 \leq k' < k$. Conversely, if s is not k -resilient, it is not k' -resilient for any $k' > k$.*

Proof. The property follows by induction on the resilience level k . If a state s is k -resilient, by Definition 1 there exists an action a such that the successor state s' is k -resilient and s is $(k - 1)$ -resilient even if a is removed. By the inductive hypothesis, s' is also $(k - 1)$ -resilient. These conditions satisfy the requirements for $(k - 1)$ -resilience of state s . By repeating this argument, it follows that s is also k' -resilient for all $0 \leq k' < k$.

The second part of the theorem follows by contraposition. Suppose, for the sake of contradiction, that a state s is not k -resilient but is k' -resilient for some $k' > k$. According to the first part of this theorem, if s is k' -resilient, then it must also be resilient for every level lower than k' , including level k . This directly contradicts the initial assumption that s is not k -resilient. Therefore, if a state is not k -resilient, it cannot be k' -resilient for any $k' > k$. $\square$

Theorem 3 (Non Resilient Propagation). *Let s be a state that is not k -resilient in any subproblem where the actions in the set V are forbidden. Then, for any action*

set $A' \supset (A \setminus V)$ and failure budget $k' = k + |A' \setminus (A \setminus V)|$, the state s is not k' -resilient in any subproblem where the actions in A' are available.

Proof. Assume for contradiction that s is not k -resilient when the available actions are $A \setminus V$, but it is k' -resilient when the available actions are A' , with $k' = k + |A' \setminus (A \setminus V)|$.

Consider a failure scenario where all actions in $A' \setminus (A \setminus V)$ fail. By the Definition 1 of a resilient state, if s is k' -resilient under A' , then after the failure of these $|A' \setminus (A \setminus V)|$ actions, s must remain k -resilient under the remaining actions. Since the remaining actions are exactly $(A' \setminus (A' \setminus (A \setminus V))) = A \setminus V$, this implies that s is k -resilient when the available actions are $A \setminus V$.

This directly contradicts our initial assumption. Therefore, s cannot be k' -resilient under A' , completing the proof. $\square$

We analyze the complexity of the Resilient Planning problem.

Theorem 4 (Problem Complexity). *The resilient planning problem $\langle \Pi, K \rangle$, where $\Pi = \langle F, A, s_0, G \rangle$, is PSPACE-complete.*

Proof. Membership: To prove that the problem belongs to PSPACE, we show that the property of k -resilience (Definition 1) can be verified within polynomial space. As outlined in Section 3.2, a resilient planning instance $\langle \Pi, K \rangle$ can be compiled into a bounded fault-tolerant task whose states are triples $\langle s, k, V \rangle$: s is the world state, $k \in \{0, \dots, K\}$ is the remaining failure budget, and $V \subseteq A$ is the set of actions that have already failed. Each augmented state can be represented in polynomial space: $O(|F| + \log K + |A|)$ bits suffice to encode a subset of facts, a bounded integer, and a subset of actions. Without loss of generality we may assume $K \leq |A|$, since after $|A|$ failures no action can fail for the first time.

The verification of k -resilience corresponds to finding a winning strategy in a reachability game over the augmented state space. This game can be solved by

a depth-first search that reuses space along each path: at any point the search stack contains at most $O(|A|)$ frames, each using $O(|F| + \log K + |A|)$ space, for a total of polynomial space. Each transition can be computed on the fly in polynomial space, and the finiteness of the augmented state space naturally guarantees termination or cycle detection within polynomial space. Hence the existence of such a strategy can be decided in PSPACE, consistently with the known complexity of AND/OR graph reachability and of strong-policy existence in fully observable non-deterministic environments.

Hardness. PSPACE-hardness is established via a reduction from classical STRIPS planning, which is PSPACE-complete [5]. A resilient planning instance $\langle \Pi, 0 \rangle$ with failure budget $K = 0$ is equivalent to the classical planning problem $\Pi = \langle F, A, s_0, G \rangle$: Case 2 of Definition 1 collapses to the standard reachability condition, so a state is 0-resilient if and only if there exists a sequence of actions leading to a goal state. Since classical STRIPS planning is a special case of Resilient Planning, the latter is also PSPACE-hard. $\square$

Chapter 4

ResPlan Algorithm

In this chapter, we address the problem of computing k -resilient plans, so plans that remain valid even in the presence of a bounded number of action failures. Building on the formal definitions introduced in Chapter 3, we present an algorithmic solution to the Resilient Planning Problem that operates directly within the framework of classical planning.

The main intuition behind RESPLAN is that a plan is resilient if all the states it traverses are resilient themselves, so that the goal remains reachable even after a limited number of failures. To construct such plans, the algorithm alternates between two phases:

- a **planning phase**, where a classical planner is used to generate candidate plans and their associated state trajectories;
- a **resilience verification phase**, where the algorithm checks whether all states along the plan are k -resilient according to the recursive Definition 1.

Through this iterative interaction between planning and verification, RESPLAN progressively identifies resilient regions of the state space and discards those

proven non-resilient. The process continues until the initial state is proven K -resilient, meaning that a plan achieving the goal under the given failure bound has been found, or until it can conclude that no resilient solution exists.

The RESPLAN algorithm represents the first complete specialized algorithm for the Resilient Planning Problem. Instead of relying on computationally expensive non-deterministic solvers, RESPLAN leverages efficient classical planners within an iterative verification loop to guarantee robustness against specific failure bounds.

This chapter describes and extends the work originally presented in [1].

4.1 RESPLAN: Algorithm Overview

4.1.1 Augmented Search Space

The algorithm highlighted in Algorithm 1 operates over an augmented search space whose nodes are triples $\langle s, k, V \rangle$, where:

- s is a classical planning state;
- $k \in \{0, \dots, K\}$ is the remaining failure budget;
- $V \subseteq A$ is the set of actions that have failed and are no longer available.

Each node $\langle s, k, V \rangle$ represents the subproblem: “Is s k -resilient when all actions in V are unavailable?”

During search, RESPLAN maintains two disjoint sets:

$$R_{\uparrow} \text{ (proven resilient nodes),} \quad R_{\downarrow} \text{ (proven non-resilient nodes).}$$

Algorithm 1 RESPLAN

Input: Resilient Planning problem $\langle \Pi = \langle F, A, s_0, G \rangle, K \rangle$

Output: K -resilient plan π^K if it exists; unsolvable otherwise

```

1:  $Open := \{\langle s_0, K, \emptyset \rangle\}; \mathcal{R}_\uparrow := \emptyset; \mathcal{R}_\downarrow := \emptyset;$ 
2: while  $Open \neq \emptyset$  do
3:    $\langle s, k, V \rangle := Open.pop()$ 
4:   if  $\langle s, k, V \rangle \notin \mathcal{R}_\uparrow \cup \mathcal{R}_\downarrow$  then
5:     if  $RCheck(s, k, V, A, G, \mathcal{R}_\uparrow)$  then
6:        $\mathcal{R}_\uparrow.add(s, k, V)$ 
7:     else
8:        $\Pi' := \langle F, A \setminus V, s, G \rangle$ 
9:        $S_\downarrow := \{s' \mid \langle s', k, V \rangle \in \mathcal{R}_\downarrow\}$ 
10:       $\pi, \tau := ComputePlan(\Pi', S_\downarrow)$ 
11:      if  $\pi = null$  then
12:         $UpdateNonResilient(s, k, V, \mathcal{R}_\downarrow)$ 
13:      else if  $k \geq 1$  then
14:        for  $i = 1$  to  $i = |\pi|$  do
15:           $Open.push(\tau_{i-1}, k, V)$ 
16:           $Open.push(\tau_{i-1}, k - 1, V \cup \{\pi_i\})$ 
17:           $\mathcal{R}_\uparrow.add(\tau_{|\pi|}, k, V)$ 
18:        else
19:          for  $i = 1$  to  $i = |\pi| + 1$  do
20:             $\mathcal{R}_\uparrow.add(\tau_{i-1}, 0, V)$ 
21: end while

```

```

22: if  $\langle s_0, K, \emptyset \rangle \in \mathcal{R}_\uparrow$  then
23:    $\pi^K := \text{ExtractSolution}(\Pi, K, \mathcal{R}_\uparrow)$ 
24:   return  $\pi^K$ 
25: else
26:   return unsolvable

27: function  $RCheck(s, k, V, A, G, \mathcal{R}_\uparrow)$ 
28:   for  $a \in A(s) \setminus V$  do
29:     if  $\langle s[a], k, V \rangle \in \mathcal{R}_\uparrow \wedge \langle s, k-1, V \cup \{a\} \rangle \in \mathcal{R}_\uparrow$  then
30:       return True
31:   return False
32: end function

33: function  $\text{ExtractSolution}(\langle F, A, s_0, G \rangle, K, \mathcal{R}_\uparrow)$ 
34:    $s := s_0$ 
35:    $\pi^K := ()$ 
36:   while  $s \not\models G$  do
37:     for  $a \in A(s)$  do
38:       if  $\langle s[a], K, \emptyset \rangle \in \mathcal{R}_\uparrow$  then
39:          $s := s[a]$ 
40:          $\pi^K.append(a)$ 
41:       break
42:   end while
43:   return  $\pi^K$ 
44: end function

```

4.1.2 Operational Flow

The algorithm begins with an initialization phase. Two sets, $R_{\uparrow}$ and $R_{\downarrow}$, are initialized as empty. The set $R_{\uparrow}$ stores all nodes $\langle s, k, V \rangle$ that have been proven k -resilient, while $R_{\downarrow}$ stores those that have been proven non-resilient. An *Open* list, implemented as a Last-In, First-Out (LIFO) stack, is initialized with the initial node $\langle s_0, K, \emptyset \rangle$. After initialization, while the open list is not empty, a node $\langle s, k, V \rangle$ is extracted from the *Open* list (thus performing a depth-first exploration), and RESPLAN proceeds as follows:

(1) Node Status Check (line 4). Before performing any computation, the algorithm checks whether the extracted node $\langle s, k, V \rangle$ has already been classified. If the node is contained in $R_{\uparrow}$ (proven resilient), it is discarded, since its resilience has been previously verified. Conversely, if it belongs to $R_{\downarrow}$ (proven non-resilient), the algorithm backtracks, as no further exploration from this node is needed. Only nodes not yet classified in either set are further processed.

(2) ResilientCheck (RCHECK) (line 5). After verifying that the current node does not belong to either the resilient or non-resilient sets, the algorithm attempts to satisfy the recursive definition of k -resilient states using already known information. For $k \geq 1$, if there exists $a \in A(s) \setminus V$ such that

$$\langle s[a], k, V \rangle \in R_{\uparrow} \quad \text{and} \quad \langle s, k-1, V \cup \{a\} \rangle \in R_{\uparrow},$$

then $\langle s, k, V \rangle$ is directly added to $R_{\uparrow}$ (line 6). This corresponds to verifying both the “success” and the “failure” branches of the recursive definition, and allows RESPLAN to prove the resilience of $\langle s, k, V \rangle$ without invoking the planner again. For the case $k = 0$, only the “success” branch needs to be considered, since 0-

resilience does not tolerate any failures: a node $\langle s, 0, V \rangle$ can be proven resilient whenever there exists an action $a \in A(s) \setminus V$ such that $\langle s[a], 0, V \rangle \in R_{\uparrow}$.

(3) Classical Planning Call (line 10). If RCHECK fails, RESPLAN constructs a classical planning problem

$$\Pi' = \langle F, A \setminus V, s, G \rangle$$

and invokes a classical planner (COMPUTEPLAN) while forbidding visits to states already known to be non-resilient. For the current pair (k, V) , we define

$$S_{\downarrow} := \{ s' \mid \langle s', k, V \rangle \in R_{\downarrow} \}.$$

This is the set of states that are already known to be non-resilient under the same (k, V) , and the planner is forbidden from visiting any state in $S_{\downarrow}$.

There are two possible outcomes:

- **Failure (lines 11-12):** no plan is found. This implies that no sequence of actions in $A \setminus V$ satisfies the resilience conditions; thus, the state s is not k -resilient under the forbidden set V , and the node $\langle s, k, V \rangle$ is added to $R_{\downarrow}$. To optimize the search, we propagate this non-resilience based on Theorem 3. Since s is not k -resilient for V , it is also non-resilient for any configuration $\langle s, k', V' \rangle$ such that $V' \subseteq V$ and $k' = k + |V \setminus V'|$. Given our invariant $k = K - |V|$, all such nodes correspond to the same state s under the same global failure budget K . By adding them to $R_{\downarrow}$, we prune the search space, ensuring s is treated as non-resilient in equivalent future configurations and preventing $S_{\downarrow}$ from re-inserting them into *Open*.

- **Success:** a plan π with trajectory $\tau = (s'_0 = s, \dots, s'_n)$ is returned.
 - If $k = 0$ (line 19-20), by definition each state s'_i reached along τ is 0-resilient in the restricted problem. Accordingly, all nodes $\langle s'_i, 0, V \rangle$ are added to $R_\uparrow$.
 - If $k \geq 1$ (lines 13-17), the algorithm must verify both the success and the failure branch for each action π_i in the plan. To this end, for each transition from s_{i-1} to s_i , RESPLAN pushes two new nodes onto *Open*:
 1. $\langle s_{i-1}, k, V \rangle$: to confirm the continued resilience of the current state;
 2. $\langle s_{i-1}, k-1, V \cup \{\pi_i\} \rangle$: to verify that the goal remains reachable even if action π_i fails.

Additionally, the goal state $\langle s_n, k, V \rangle$ is added to $R_\uparrow$ (line 18), as it trivially satisfies the resilience conditions.

4.1.3 Plan Extraction

When the algorithm terminates, two cases are possible. If the initial node $\langle s_0, K, \emptyset \rangle$ does not belong to $R_\uparrow$, then the problem is declared *unsolvable*, meaning that no K -resilient plan exists. Otherwise, if $\langle s_0, K, \emptyset \rangle \in R_\uparrow$, the algorithm reconstructs a valid plan by traversing the resilient states recorded in $R_\uparrow$.

Starting from the initial state s_0 , the algorithm repeatedly searches for an action $a \in A(s)$ such that the successor $\langle s[a], K, \emptyset \rangle$ also belongs to $R_\uparrow$. The first such action is appended to the plan, and the process continues from the new state $s[a]$. By iterating this selection until a goal state is reached, the algorithm builds a sequence of actions $\pi = (a_1, a_2, \dots, a_n)$ in which every intermediate state remains K -resilient by construction.

Since all states in $R^\uparrow$ are proved resilient through valid acyclic plans found by the classical planner, the set of K -resilient states reachable from s_0 is finite and acyclic; therefore the extraction procedure is guaranteed to terminate in a finite number of steps.

4.1.4 Theoretical Properties

We now provide basic guarantees for the correctness and termination of RESPLAN, assuming that the underlying classical planner used in COMPUTEPLAN is sound, complete, and always terminates.

Theorem 5. (*Termination*) *The algorithm RESPLAN terminates in a finite number of iterations.*

Proof. Given a finite set of states and a bounded number of failures K , the number of nodes $\langle s, k, V \rangle$ that can be generated is finite. During execution, each processed node is eventually classified as either resilient (added to $R_\uparrow$) or non-resilient (added to $R_\downarrow$), and classified nodes are never reconsidered. Since every iteration either classifies a new node or detects that there are no more nodes to process, no infinite sequence of expansions can occur. Therefore, RESPLAN is guaranteed to terminate after a finite number of iterations. $\square$

Theorem 6. (*Soundness*) *Every plan output by RESPLAN is guaranteed to be K -resilient.*

Proof. We proceed by induction on the order of insertion into $R^\uparrow$. The base case holds because goal states are added to $R^\uparrow$ directly, and any state $s \models G$ is k -resilient by Definition 1, Case 1. For the inductive step, every node added to $R^\uparrow$ satisfies the recursive definition of k -resilience, as both the success and failure conditions are explicitly verified against nodes already in $R^\uparrow$, which are correctly

classified by the inductive hypothesis. Conversely, nodes added to $R^\downarrow$ correspond to triples $\langle s, k, V \rangle$ for which no plan can fulfill the resilience condition. Consequently, if the algorithm outputs a plan, all its states belong to $R^\uparrow$ with resilience level K , and, by definition, the resulting plan is guaranteed to be K -resilient. $\square$

Theorem 7. (Completeness) *If a K -resilient plan exists, RESPLAN will eventually find it.*

Proof. The planner COMPUTEPLAN can always return a valid plan whenever one exists. Since the algorithm systematically instantiates both the “success” and “fallback” branches of the recursive definition of resilience, every state that can be shown to be K -resilient will eventually be added to $R^\uparrow$, and in particular $\langle s_0, K, \emptyset \rangle$ will be marked as resilient. By the plan extraction procedure, this guarantees that a K -resilient plan is returned whenever one exists. $\square$

4.2 ResPlan Workflow Example

In this section, we illustrate the workflow of the RESPLAN algorithm through an example based on the navigation domain shown in Figure 3.1. The agent starts at location $\mathcal{A}$ and the goal is to reach location $\mathcal{G}$. We assume $K = 2$, and we aim at computing a 2-resilient plan.

4.2.1 Initialization

The algorithm initializes the *Open* list with the initial node $\langle \mathcal{A}, 2, \emptyset \rangle$, while both the resilient set $R^\uparrow$ and the non-resilient set $R^\downarrow$ are empty. Intuitively, the first question the algorithm must answer is: “Is state $\mathcal{A}$ 2-resilient when no action has failed yet?”

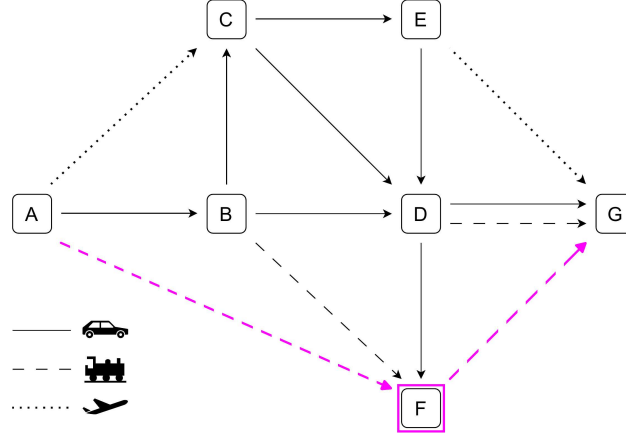


Figure 4.1: Example of first expansion.

4.2.2 First Iteration

The first node extracted from *Open* is $\langle \mathcal{A}, 2, \emptyset \rangle$.

Since the node does not appear in either $R_\uparrow$ or $R_\downarrow$, the algorithm performs the **RCHECK** procedure. At this early stage, the resilient set is empty, so **RCHECK** cannot prove the node resilient. As a result, **RESPLAN** invokes the classical planner on the restricted problem $\Pi' = \langle F, \mathcal{A}, \mathcal{G} \rangle$.

Assume that the planner returns the plan

$$\pi_1 = (\text{train}(\mathcal{A}, \mathcal{F}), \text{train}(\mathcal{F}, \mathcal{G})),$$

leading from $\mathcal{A}$ to state $\mathcal{F}$ and then to the goal $\mathcal{G}$. Let $\tau_1 = (\mathcal{A}, \mathcal{F}, \mathcal{G})$ be the corresponding trajectory.

For each pair (s_i, a_i) in τ_1 and resilience level $k = 2$, **RESPLAN** pushes back onto the *Open* list both the current subproblem and the one that models the failure of a_i .

In our example, from the transition $\mathcal{A} \xrightarrow{\text{train}(\mathcal{A}, \mathcal{F})} \mathcal{F}$ we obtain:

$$\langle \mathcal{A}, 2, \emptyset \rangle \quad \text{and} \quad \langle \mathcal{A}, 1, \{\text{train}(\mathcal{A}, \mathcal{F})\} \rangle,$$

and from the transition $\mathcal{F} \xrightarrow{\text{train}(\mathcal{F}, \mathcal{G})} \mathcal{G}$ we obtain:

$$\langle \mathcal{F}, 2, \emptyset \rangle \quad \text{and} \quad \langle \mathcal{F}, 1, \{\text{train}(\mathcal{F}, \mathcal{G})\} \rangle.$$

All such nodes are pushed onto the *Open* list in LIFO order. Since $\mathcal{G}$ is a goal state, the node $\langle \mathcal{G}, 2, \emptyset \rangle$ is immediately recognized as resilient and added to $R_{\uparrow}$.

4.2.3 Second and Third Iterations

At the second iteration, a new node is extracted from the *Open* list. The node $\langle \mathcal{F}, 1, \{\text{train}(\mathcal{F}, \mathcal{G})\} \rangle$ is selected. Since this node does not belong to $R_{\uparrow}$ nor to $R_{\downarrow}$, the algorithm runs RCHECK. At this point, however, there is still not enough information in $R_{\uparrow}$ to prove resilience, so RCHECK fails and a new planning call is required. RESPLAN then invokes the classical planner on the restricted problem

$$\Pi' = \langle F, A \setminus \{\text{train}(\mathcal{F}, \mathcal{G})\}, \mathcal{F}, \mathcal{G} \rangle.$$

Intuitively, this models the situation where the action $\text{train}(\mathcal{F}, \mathcal{G})$ has failed and is no longer available. In this problem, the planner cannot find any plan from $\mathcal{F}$ to $\mathcal{G}$, since $\text{train}(\mathcal{F}, \mathcal{G})$ is the only action that can reach the goal from $\mathcal{F}$. Therefore, Π' is unsolvable and the node $\langle \mathcal{F}, 1, \{\text{train}(\mathcal{F}, \mathcal{G})\} \rangle$ is added to the non-resilient set $R_{\downarrow}$.

This result has an immediate consequence for the resilience of $\mathcal{F}$ with budget 2. In fact, from state $\mathcal{F}$ with $k = 2$ the only applicable action is $\text{train}(\mathcal{F}, \mathcal{G})$,

and the “failure” branch corresponds exactly to $\langle \mathcal{F}, 1, \{\text{train}(\mathcal{F}, \mathcal{G})\} \rangle$, which we have just proven non-resilient. Since this failure branch is non-resilient, no action from $\mathcal{F}$ can satisfy the recursive definition of 2-resilience. As a consequence, the node $\langle \mathcal{F}, 2, \emptyset \rangle$ is also added to $R_\downarrow$. This is an instance of the propagation of non-resilience described in the main algorithm: once $\langle \mathcal{F}, 1, \{\text{train}(\mathcal{F}, \mathcal{G})\} \rangle$ is marked as non-resilient, the corresponding configuration $\langle \mathcal{F}, 2, \emptyset \rangle$, which represents the same state under the same global failure bound K , is immediately marked non-resilient as well.

In the next iteration, the node $\langle \mathcal{F}, 2, \emptyset \rangle$ is extracted from the *Open* list. Since it already belongs to $R_\downarrow$, RCHECK immediately detects that it is non-resilient and the node is discarded. The algorithm then proceeds to the following iteration.

4.2.4 Fourth Iteration

In a later iteration, the node $\langle \mathcal{A}, 2, \emptyset \rangle$ is extracted again from the *Open* list. At this point, however, the set $R_\downarrow$ is no longer empty: in particular, the node $\langle \mathcal{F}, 2, \emptyset \rangle$ has already been proven non-resilient. Since $\langle \mathcal{A}, 2, \emptyset \rangle$ does not belong to either $R_\uparrow$ or $R_\downarrow$, the algorithm runs RCHECK. As in the first iteration, RCHECK cannot prove resilience, so RESPLAN invokes the classical planner on the problem $\Pi' = \langle F, A, \mathcal{A}, \mathcal{G} \rangle$. This time, however, $R_\downarrow$ is not empty, and therefore the planning call must avoid all states already known to be non-resilient for the current pair $(k, V) = (2, \emptyset)$. In particular, the state $\mathcal{F}$ with budget 2 must be excluded, since it belongs to the corresponding set $S_\downarrow$. The planner is now searching for a new plan from $\mathcal{A}$ to $\mathcal{G}$ that does not traverse $\mathcal{F}$, because any plan passing through $\mathcal{F}$ cannot be 2-resilient.

Assume that the planner returns the alternative plan

$$\pi_2 = (\text{car}(\mathcal{A}, \mathcal{B}), \text{car}(\mathcal{B}, \mathcal{D}), \text{car}(\mathcal{D}, \mathcal{G})).$$

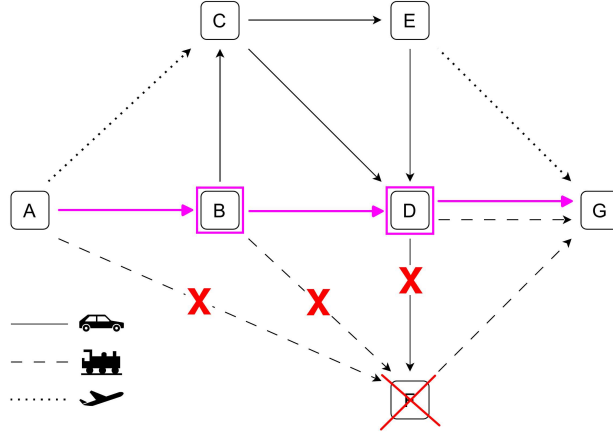


Figure 4.2: Example of second plan found.

For each transition in the induced trajectory, RESPLAN pushes back onto the *Open* list both the current subproblem and the one that models the failure of the corresponding action. In this case, the new nodes are:

$$\begin{aligned}
 \langle \mathcal{A}, 2, \emptyset \rangle \quad \text{and} \quad \langle \mathcal{A}, 1, \{\text{car}(\mathcal{A}, \mathcal{B})\} \rangle, \\
 \langle \mathcal{B}, 2, \emptyset \rangle \quad \text{and} \quad \langle \mathcal{B}, 1, \{\text{car}(\mathcal{B}, \mathcal{D})\} \rangle, \\
 \langle \mathcal{D}, 2, \emptyset \rangle \quad \text{and} \quad \langle \mathcal{D}, 1, \{\text{car}(\mathcal{D}, \mathcal{G})\} \rangle.
 \end{aligned}$$

All these nodes are inserted in LIFO order. At this point, the current iteration ends and the algorithm proceeds with a new iteration by extracting the next node from the *Open* list.

4.2.5 Proving that states $\mathcal{D}$, $\mathcal{B}$, $\mathcal{A}$ are 2-resilient

The next node extracted from the *Open* list is $\langle \mathcal{D}, 1, \{\text{car}(\mathcal{D}, \mathcal{G})\} \rangle$. Since this node does not belong to $R_{\uparrow}$ nor to $R_{\downarrow}$, the algorithm runs RCHECK. At this point, however, RCHECK cannot prove resilience, and the planner is invoked on the

subproblem $\Pi'' = \langle F, A \setminus \{\text{car}(\mathcal{D}, \mathcal{G})\}, \mathcal{D}, \mathcal{G} \rangle$. Intuitively, this models the situation where $\text{car}(\mathcal{D}, \mathcal{G})$ has failed and cannot be used anymore. Assume that the planner returns the plan $\pi_3 = (\text{train}(\mathcal{D}, \mathcal{G}))$.

For the transition $\mathcal{D} \xrightarrow{\text{train}(\mathcal{D}, \mathcal{G})} \mathcal{G}$ and resilience level $k = 1$, RESPLAN pushes onto the *Open* list:

$$\langle \mathcal{D}, 1, \{\text{car}(\mathcal{D}, \mathcal{G})\} \rangle \quad \text{and} \quad \langle \mathcal{D}, 0, \{\text{car}(\mathcal{D}, \mathcal{G}), \text{train}(\mathcal{D}, \mathcal{G})\} \rangle.$$

Moreover, since $\mathcal{G}$ is a goal state, the node $\langle \mathcal{G}, 1, \{\text{car}(\mathcal{D}, \mathcal{G})\} \rangle$ is immediately recognized as resilient and added to $R_\uparrow$.

In the following iteration, the node $\langle \mathcal{D}, 0, \{\text{car}(\mathcal{D}, \mathcal{G}), \text{train}(\mathcal{D}, \mathcal{G})\} \rangle$ is extracted from *Open*. Again, RCHECK fails to prove resilience, and the planner is called on the subproblem $\Pi''' = \langle F, A \setminus \{\text{car}(\mathcal{D}, \mathcal{G}), \text{train}(\mathcal{D}, \mathcal{G})\}, \mathcal{D}, \mathcal{G} \rangle$. Assume that the planner returns the plan $\pi_4 = (\text{car}(\mathcal{D}, \mathcal{F}), \text{train}(\mathcal{F}, \mathcal{G}))$, with trajectory $(\mathcal{D}, \mathcal{F}, \mathcal{G})$. Since $k = 0$, this proves that $\mathcal{D}$ is 0-resilient in the restricted problem. Consequently, the nodes

$$\langle \mathcal{D}, 0, \{\text{car}(\mathcal{D}, \mathcal{G}), \text{train}(\mathcal{D}, \mathcal{G})\} \rangle \quad \text{and} \quad \langle \mathcal{G}, 0, \{\text{car}(\mathcal{D}, \mathcal{G}), \text{train}(\mathcal{D}, \mathcal{G})\} \rangle$$

are added to $R_\uparrow$.

In the next iteration, the node $\langle \mathcal{D}, 1, \{\text{car}(\mathcal{D}, \mathcal{G})\} \rangle$ is extracted again from *Open*. Now RCHECK can prove resilience: the resilient set already contains both components required by the recursive test, namely the successor node $\langle \mathcal{G}, 1, \{\text{car}(\mathcal{D}, \mathcal{G})\} \rangle$ and the same state with a lower resilience level, $\langle \mathcal{D}, 0, \{\text{car}(\mathcal{D}, \mathcal{G}), \text{train}(\mathcal{D}, \mathcal{G})\} \rangle$. Therefore, $\langle \mathcal{D}, 1, \{\text{car}(\mathcal{D}, \mathcal{G})\} \rangle$ is added to $R_\uparrow$.

Following the same pattern, the algorithm extracts $\langle \mathcal{D}, 2, \emptyset \rangle$, and RCHECK can now prove resilience: the successor subproblem $\langle \mathcal{G}, 2, \emptyset \rangle$ is already in $R_\uparrow$, and the

failure outcome $\langle \mathcal{D}, 1, \{\text{car}(\mathcal{D}, \mathcal{G})\} \rangle$ has just been proven resilient. Therefore, $\langle \mathcal{D}, 2, \emptyset \rangle$ is added to $R_{\uparrow}$.

State $\mathcal{D}$ is thus the first state (other than the goal) proven to be 2-resilient. Following the same reasoning described above for state $\mathcal{D}$, the algorithm also revisits the nodes

$$\langle \mathcal{A}, 2, \emptyset \rangle \quad \text{and} \quad \langle \mathcal{B}, 2, \emptyset \rangle.$$

Both states admit multiple alternative plans that reach the goal, and for each candidate plan RESPLAN systematically evaluates the corresponding subproblems. By repeatedly applying the same recursive reasoning used for $\mathcal{D}$, the algorithm proves that all required branches are resilient. Consequently, both $\langle \mathcal{A}, 2, \emptyset \rangle$ and $\langle \mathcal{B}, 2, \emptyset \rangle$ are marked as 2-resilient and added to $R_{\uparrow}$.

4.2.6 Generating the 2-resilient plan

Once all nodes relevant to the initial configuration have been classified, RESPLAN has established that the initial state $\langle \mathcal{A}, 2, \emptyset \rangle$ belongs to the resilient set $R_{\uparrow}$. At this point, the algorithm can reconstruct a 2-resilient plan by following, for each resilient state, one of the actions whose associated subproblems (success and failure) have been verified as resilient during the search.

Starting from state $\mathcal{A}$, RESPLAN selects the action $\text{car}(\mathcal{A}, \mathcal{B})$, since its successor state $\langle \mathcal{B}, 2, \emptyset \rangle$ is known to be 2-resilient and both the corresponding success and failure subproblems satisfy the recursive definition of resilience. From $\mathcal{B}$, the algorithm proceeds analogously with $\text{car}(\mathcal{B}, \mathcal{D})$, reaching the 2-resilient state $\mathcal{D}$, and finally applies $\text{train}(\mathcal{D}, \mathcal{G})$ to reach the goal.

In this example, the resulting 2-resilient plan is

$$\pi^* = (\text{car}(\mathcal{A}, \mathcal{B}), \text{car}(\mathcal{B}, \mathcal{D}), \text{train}(\mathcal{D}, \mathcal{G})),$$

which induces the trajectory $(\mathcal{A}, \mathcal{B}, \mathcal{D}, \mathcal{G})$ and, by construction, guarantees that all states along this trajectory are 2-resilient.

4.3 RESPLAN Limitations

While the RESPLAN algorithm offers a robust solution for resilient planning, it faces several limitations that can affect its performance, especially when applied to larger and more complex planning problems. One of the most significant challenges with RESPLAN is proving the non-resilience of a problem, particularly in cases that involve high resilience levels. The algorithm requires the full exploration of both success and failure branches for each action to prove the non-resilience of each state, which leads to exponential growth in the number of subproblems as the failure budget K increases. This expansion of the state space can quickly overwhelm the system, leading to excessive time and memory requirements. As the number of possible failure scenarios increases, the algorithm must evaluate an increasing number of nodes, many of which are redundant or non-contributory to the final resilient plan. This makes the algorithm inefficient for domains with large state spaces or when high resilience levels are required.

Another limitation of RESPLAN is its reliance on classical planners for generating plans within subproblems. In RESPLAN, the classical planner is invoked repeatedly for each subproblem, with each invocation focused on a restricted version of the original problem. This approach can be inefficient because the algorithm does not make use of previously acquired information. Each new in-

vocation requires the planner to compute solutions from scratch, even if similar or overlapping subproblems were already solved during previous iterations.

As a result, this lack of information reuse leads to redundant planning efforts and can significantly increase computation time, particularly in more complex scenarios where no resilient plan exists for a given subproblem or when the planner is unable to find a solution. These repeated calls to the classical planner can severely impact the overall performance of RESPLAN, making it less efficient for large-scale or complex planning problems.

To address these limitations, the next chapter introduces RESPLAN2, an enhanced version of RESPLAN, which incorporates key improvements that enable it to handle larger and more complex problems.

Chapter 5

Enhanced RESPLAN: RESPLAN2

In the previous chapter, we presented the RESPLAN algorithm and, in the final section, we discussed its limitations, particularly with regard to the exponential growth of the state space and the inefficiencies caused by the repeated invocation of classical planners.

In this chapter, we introduce RESPLAN2 [32], an enhanced version of the original RESPLAN algorithm, designed to overcome these limitations. RESPLAN2 addresses the issue of inefficient knowledge reuse by incorporating two key advancements: landmark-based pruning and adaptive replanning strategies. These innovations allow RESPLAN2 to avoid unnecessary state explorations, reduce reliance on classical planners, and more effectively utilize the information gathered during previous planning iterations.

By implementing these improvements, RESPLAN2 enhances scalability and performance, making it better suited for real-world resilient planning applications. This chapter will explore the theoretical foundations of these advancements, explain how they are integrated into RESPLAN2, and demonstrate how they contribute to the algorithm's overall efficiency and effectiveness.

5.1 Landmarks Pruning Technique

A central aspect of RESPLAN2 is the use of *landmarks* as a form of control knowledge to guide and constrain the search in the state space, reducing unnecessary explorations and improving overall efficiency. In this section, we formally define landmarks, describe how they are extracted in the context of a planning problem, and explain how RESPLAN2 exploits their properties to perform effective *landmark-based pruning*.

5.1.1 Definition of Landmarks

In the literature ([20, 31, 19]), a landmark for a planning task is defined as a fact (or, more generally, a state property or an action) that must necessarily be satisfied at some point in any valid plan that leads from the initial state to the goal state. In practical terms, if a fact f is a landmark for the goal G , then in every sequence of actions that achieves G , regardless of the specific plan chosen, there must be a state in which f is satisfied. This makes landmarks weak but global constraints on the plan space. Some specific types of landmarks include:

- **Fact Landmarks:** propositional facts that must become true at some point;
- **Action Landmarks:** actions that must be executed at least once;
- **Disjunctive Landmarks:** sets of facts or actions where at least one of the elements must be satisfied (useful when there are alternatives).

In the context of RESPLAN2, the landmarks we use are specifically fact landmarks, which are formally defined as follows:

Definition 3. A fact $f \in F$ is a landmark if, for every trajectory τ induced by a solution plan for Π , $\exists i \leq |\tau| : f \in \tau_i$.

We define $\text{LANDMARKS}(F, A, s, G)$ as the set of fact landmarks generated from the s in the planning problem $\Pi = \langle F, A, s, G \rangle$.

5.1.2 Connecting Landmarks to Non-Resilient States

In `RESPLAN2`, landmarks are specifically employed to identify non-resilient states. This enables early pruning of the search space by excluding these states from further exploration as soon as they are detected.

A state is considered k -resilient if, after up to k action failures, there exists a sequence of actions that still leads to the goal state. However, if a state does not satisfy the necessary conditions for resilience, it can be deemed non-resilient. The key idea here is that, since up to k action failures can occur, potentially affecting the actions that achieve a landmark, at least $k+1$ actions must exist that achieve the same landmark for a resilient plan to be possible. If this redundancy is not found, the state can be considered non-resilient. In other words, we leverage the concept of landmarks to define a sufficient condition for proving the non-resilience of a state. Specifically, if the required redundancy of actions to achieve a landmark is absent, the state is marked as non-resilient.

Formally, let $\text{Ach}(f, A) = \{a \in A \mid f \in \text{add}(a)\}$ be the subset of actions in A that achieve the fact f . We show that the existence of a landmark f such that $|\text{Ach}(f, A)| \leq k$ is a sufficient condition to declare that a state s cannot be k -resilient. In other words, if there are fewer than $k+1$ actions that can achieve a given landmark f , then the state is non-resilient, as it is not possible to maintain the achievement of the landmark after k action failures.

Following this intuition, we define the following theorem that formalizes the connection between landmarks and state resilience.

Theorem 8. Let $\Pi = \langle F, A, s, G \rangle$ be a planning problem and k a non-negative integer. If there exists a fact-landmark $f \in \text{LANDMARKS}(F, A, s, G)$ such that $|\text{Ach}(f, A)| \leq k$, then the state s is not k -resilient in Π .

Proof. Assume that there exists a landmark $f \in \text{LANDMARKS}(F, A, s, G)$ such that $|\text{Ach}(f, A)| \leq k$ and s is k -resilient. We derive a contradiction with Definition 1.

- **Case (i):** If $s \supseteq G$, then $\text{LANDMARKS}(F, A, s, G) = \emptyset$, and therefore the landmark f cannot exist. Contradiction.
- **Case (ii):** If $k = 0$, then $|\text{Ach}(f, A)| \leq 0$, meaning that no action can achieve f . Therefore, no solution plan exists for $\langle F, A, s, G \rangle$, and if no plan exists, a state s cannot be 0-resilient. Contradiction.
- **Case (iii):** $k > 0$. Let $V^* = \text{Ach}(f, A)$ be the set of all achievers of f in A . By hypothesis $|V^*| \leq k$.

Consider the residual problem $\Pi^* = \langle F, A \setminus V^*, s, G \rangle$. In Π^* the set of achievers of f is

$$\text{Ach}(f, A \setminus V^*) = \text{Ach}(f, A) \setminus V^* = \emptyset.$$

Since f is a landmark for Π , every solution plan must pass through a state in which f holds (Definition 3). With no achiever of f available in Π^* , no solution plan exists, and therefore s is not 0-resilient in Π^* .

By the contrapositive of Theorem 3 since s is not 0-resilient in $\langle F, A \setminus V^*, s, G \rangle$ and $|V^*| \leq k$, it follows that s is not k -resilient in $\langle F, A, s, G \rangle$.

This directly contradicts our assumption. Contradiction.

Therefore, if $|\text{Ach}(f, A)| \leq k$, the state s is non-resilient. □

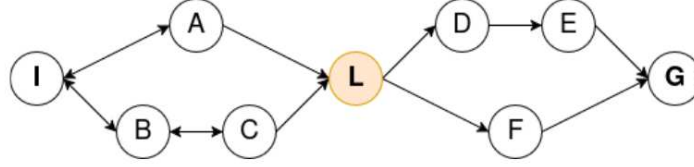


Figure 5.1: Example of landmark pruning technique.

5.1.3 Example Landmark Pruning Technique

Let us consider the graph in Figure 5.1, which represents a planning problem with nodes as states and directed edges as actions. The initial state is $\mathcal{I}$, and the goal is to reach $\mathcal{G}$. In this scenario, we are interested in identifying fact landmarks that must be visited in any valid plan. In this example, the fact landmark that we can find from the state $\mathcal{I}$ is: $\text{LANDMARKS}(F, A, \mathcal{I}, \mathcal{G}) = \mathcal{L}$. The key observation is that if the action leading to $\mathcal{L}$ fails, the plan must have alternative actions to ensure resilience.

Now, applying Theorem 8 based on landmarks, we can deduce the following: the landmark $\mathcal{L}$ has only two achievers: one from the state $\mathcal{A}$ and one from the state $\mathcal{C}$. If both actions fail, it becomes impossible to reach the goal from the initial state $\mathcal{I}$, making it non-resilient. Therefore, if we want to find at least one valid 2-resilient plan, we can immediately conclude that it does not exist: by Theorem 8, $|\text{Ach}(\mathcal{L}, A)| \leq 2$ implies that $\mathcal{I}$ is not 2-resilient.

5.2 Adaptive Replanning Technique

In RESPLAN, planning is performed from scratch at every iteration, without reusing the information acquired in previous steps. This approach is conceptually simple but inefficient, as the search is recomputed even when existing knowledge could be exploited.

The adaptive replanning strategy in RESPLAN2 addresses this limitation by allowing the planner to reconnect the current search to already verified resilient suffixes. This mechanism is based on the decomposition of k -resilient plans: if a state s' is already known to be k -resilient, it implies the existence of a k -resilient plan suffix from s' to the goal. Consequently, to prove that s is k -resilient, it is sufficient to find a plan prefix from s to s' and verify that every state along this prefix satisfies the k -resilience criteria.

This allows the algorithm to terminate the planning phase early, as the resilience of the trajectory from s' to the goal has already been established under the same constraints (k, V) . The task then reduces to verifying the resilience of the newly generated prefix from s to s' .

5.2.1 Resilience Certificates

In this chapter, we introduce a new representation of sets of resilient states, called *resilience certificates*, which provide an efficient way to recognise when the current search state can be safely reconnected to a previously verified resilient suffix. Instead of storing individual resilient states, RESPLAN2 stores abstract conditions under which a state can be guaranteed to be k -resilient.

A k -resilience certificate for a planning problem $\Pi = \langle F, A, s_0, G \rangle$ is a pair $\langle \phi, D \rangle$, where $\phi \subseteq F$ is a formula representing a partial description of the state (conjunctions of positive literals), and $D \subseteq A$ is a set of actions. A certificate $\langle \phi, D \rangle$ represents a set of resilient states: any state s such that $s \models \phi$ is k -resilient, provided that all actions in D remain available. In other words, ϕ encodes the relevant part of the state, and D is sufficient to build resilient plans from s .

Resilience certificates are constructed by propagating resilience information backwards along the actions that appear in the plans found by the planner. Given

a formula ϕ that characterises a set of (already) resilient states, we use regression to compute a new formula that describes the states from which executing a given action leads into ϕ . In this way, we can derive larger sets of resilient states without enumerating them explicitly. To this end, we use a regression operator. Let $\phi \subseteq F$ be a formula and a an action. The regression of ϕ over a , written $Reg(\phi, a)$, is defined as:

$$Reg(\phi, a) = (\phi \setminus add(a)) \cup pre(a).$$

Throughout this work, regression is applied only along valid plans, ensuring that $del(a) \cap \phi = \emptyset$ always holds. Thus, $Reg(\phi, a)$ characterises the states from which executing a leads to a state satisfying ϕ . To connect the notion of k -resilient state (Definition 1) with resilience certificates, we now formally define an inductive definition of k -resilience certificates.

Definition 4. Let $\Pi = \langle F, A, s_0, G \rangle$ be a planning problem, and k a non-negative integer.

- **Case 1:** $\langle G, \emptyset \rangle$ is a k -resilience certificate.
- **Case 2:** If $\langle \phi, D \rangle$ is a 0-resilience certificate, then $\langle Reg(\phi, a), D \cup \{a\} \rangle$ is a 0-resilience certificate, for any $a \in A$.
- **Case 3:** If $\langle \phi, D \rangle$ is a k -resilience certificate and $\langle \phi', D' \rangle$ is a $(k-1)$ -resilience certificate with $a \notin D'$, then $\langle Reg(\phi, a) \cup \phi', D \cup D' \cup \{a\} \rangle$ is a k -resilience certificate, for any $a \in A$.

Theorem 9. Let $\Pi = \langle F, A, s_0, G \rangle$ be a planning problem, and $\langle \phi, D \rangle$ be a k -resilience certificate for Π . Any state s such that $s \models \phi$ is k -resilient in Π .

Proof. We verify the three cases of Definition 4 by induction on its structure. For case (i), any state $s \supseteq G$ is a goal state and is therefore k -resilient for all

$k \geq 0$ by Definition 1, Case 1. For the 0-resilience certificates of case (ii), $\phi = \text{Reg}(\phi_0, a)$ implies that a is applicable in s and that $s[a] \models \phi_0$; by induction $s[a]$ is 0-resilient, so a plan to G exists from s . For the general case (iii), $s \models \text{Reg}(\phi_k, a)$ guarantees that a is applicable and $s[a]$ is k -resilient (success branch), while $s \models \phi_{k-1}$ guarantees that s is $(k-1)$ -resilient without a ; note that by construction $a \notin D_{k-1}$, ensuring that the fallback certificate remains applicable after the failure of a (failure branch); by Definition 1, Case 3, s is therefore k -resilient. $\square$

Resilience certificates generalize resilient states, as a single certificate captures a set of resilient states. This enables a more practical implementation of the adaptation strategy outlined above, since instead of associating a state solely with the resilience level and a planning problem, we can record the actual set of actions that were used to prove the state resilient and the formula that the state needs to satisfy to execute these actions.

5.2.2 Example Adaptive Replanning Technique

We illustrate the proposed adaptive replanning strategy through a simple example. Consider a resilient planning problem $\langle \Pi, 1 \rangle$ with $\Pi = \langle F, A, \{i\}, G \rangle$, where $K = 1$, the set of facts is $F = \{i, p, q, r, g\}$, the actions are $A = \{a, b, c_1, c_2, d_1, d_2, e_1, e_2\}$, the initial state is $s_0 = \{i\}$, and the goal condition is $G = \{g\}$. The precondition, add and delete lists of each action, together with the induced state space, are shown in Figure 5.2.

Suppose that, in the first replanning episode, the algorithm generates the candidate plan (a, b, c_1) . By analysing this plan and its recovery options for a single failure, the procedure proves that state $s = \{p, r\}$ is 1-resilient, exploiting the alternative action c_2 to recover from potential failures of c_1 . As a consequence,

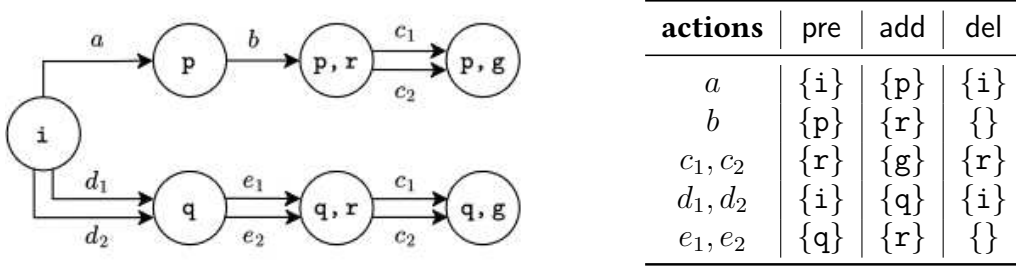


Figure 5.2: Example illustrating adaptive replanning strategy.

the algorithm records the resilience certificate $\langle \{r\}, \{c_1, c_2\} \rangle$, stating that any state containing r is 1-resilient as long as actions c_1 and c_2 remain available.

As the search proceeds, state p is eventually found to be non-1-resilient. This forces the algorithm to backtrack to the initial state i and start a new replanning episode. Assume that, in this second episode, the returned plan is (d_1, e_1) , which reaches state $\{q, r\}$. Since this state satisfies the condition of the previously stored resilience certificate (it contains the formula r and the actions in $D = \{c_1, c_2\}$ are still available), the planner can halt the search early and return this partial plan. The plan prefix (d_1, e_1) is then connected to the already verified resilient suffix (c_1) , eventually yielding the complete 1-resilient plan (d_1, e_1, c_1) .

Crucially, this kind of plan reuse would not be possible without resilience certificates. A non-adaptive resilient planner could also establish that state $\{p, r\}$ is 1-resilient, but it would have to verify the resilience of $\{q, r\}$ independently, incurring additional search and planner calls.

The structure highlighted in this example is common in crafting domains, where several ingredients must be collected before manufacturing the final product. A similar pattern appears in delivery domains or in logistics scenario, where certain parts of the transportation network exhibit higher resilience than others. In this setting, it is advantageous to reuse plans whenever possible, exactly as in this example

5.3 RESPLAN2: Algorithm Overview

In this section, we present the advanced version of the RESPLAN algorithm, RESPLAN2, designed to overcome the limitations discussed in Chapter 4. It introduces two key improvements: (1) an early detection mechanism for non-resilient states, and (2) an adaptive replanning strategy. While RESPLAN retains the basic structure of the original algorithm, several modifications have been made to support these new features, as shown in Algorithm 2.

The main advancement over the previous version is the introduction of resilience certificates, which replace the resilient states in the algorithm's memory $R \uparrow$. In the original version, $R \uparrow$ contained the nodes $\langle s, k, V \rangle$ whenever a state s was proven to be k -resilient. In RESPLAN2, $R \uparrow$ stores resilience certificates in the form of triples $\langle \phi, k, D \rangle$, already presented and discussed.

In the following, we present the key differences between RESPLAN2 and the original RESPLAN algorithm.

5.3.1 Operational Flow

As in the original implementation, RESPLAN2 takes as input a resilient planning problem $\langle \Pi = \langle F, A, s_0, G \rangle, K \rangle$, and the goal is to provide a valid resilient plan, or declare the problem unsolvable otherwise.

The new algorithm introduces the function $\text{LMNONRESILIENT}(s, k, V)$ (lines 48-53), based on Theorem 8. This function detects whether a state s can be considered non-resilient at an early stage, using landmarks. Specifically, it returns *True* if there exists a landmark f such that $|\text{Ach}(f, A \setminus V)| \leq k$, marking the state as non-resilient; otherwise, it returns *False*.

Algorithm 2 RESPLAN2

Input: Resilient Planning problem $\langle \Pi = \langle F, A, s_0, G \rangle, K \rangle$
Output: K -resilient plan π^K if it exists; unsolvable otherwise

```

1: if  $LMNonResilient(s_0, K, \emptyset)$  then
2:   return Unsolvable
3:  $Open := \{\langle s_0, K, \emptyset \rangle\}; \mathcal{R} \uparrow := \emptyset; \mathcal{R} \downarrow := \emptyset;$ 
4: while  $Open \neq \emptyset$  do
5:    $\langle s, k, V \rangle := Open.pop()$ 
6:   if  $\langle s, k, V \rangle \notin \mathcal{R} \downarrow \wedge \nexists \langle \phi', k', D' \rangle \in \mathcal{R} \uparrow$ 
7:     s.t.  $\langle s, k, V \rangle \sim \langle \phi', k', D' \rangle$  then
8:       if  $RCheck(s, k, V, A, G, \mathcal{R} \uparrow)$  then
9:         continue
10:      else
11:         $G_\phi = SelectCandidateDjGoals(k, V, \mathcal{R} \uparrow)$ 
12:         $\Pi' := \langle F, A \setminus V, s, G \cup G_\phi \rangle$ 
13:         $S_\downarrow := \{s' \mid \langle s', k, V \rangle \in \mathcal{R} \downarrow\}$ 
14:         $\pi, \tau, G^* := ComputePlan(\Pi', S_\downarrow)$ 
15:        if  $\pi = null$  then
16:           $UpdateNonResilient(s, k, V, \mathcal{R} \downarrow)$ 
17:        else if  $k \geq 1$  then
18:          for  $i = 1$  to  $i = |\pi|$  do
19:            if  $LMNonResilient(\tau_{i-1}, k, V)$  then
20:               $UpdateNonResilient(\tau_{i-1}, k, V, \mathcal{R} \downarrow)$ 
21:            break
22:          else
23:             $Open.push(\tau_{i-1}, k, V)$ 

```

```

24:            $Open.push(\tau_{i-1}, k - 1, V \cup \{\pi_i\})$ 
25:            $\mathcal{R} \uparrow.add(\tau_{|\pi|}, k, V)$ 
26:       else
27:            $\phi \leftarrow G^*, D \leftarrow \emptyset$ 
28:           for  $i = |\pi|$  to  $i = 1$  do
29:                $\phi \leftarrow Reg(\phi, \pi_i), D \leftarrow D \cup \{\pi_i\}$ 
30:               add  $\langle \phi, 0, D \rangle$  to  $\mathcal{R}_\uparrow$ 
31:       end while
32:   if  $\exists \langle \phi, K, D \rangle \in \mathcal{R} \uparrow$  s.t.  $s_0 \models \phi$  then
33:        $\pi^K := ExtractSolution(\Pi, K, \mathcal{R} \uparrow)$ 
34:       return  $\pi^K$ 
35:   else
36:       return Unsolvable

37: function  $RCheck(s, k, V)$ 
38:     for  $a \in A(s) \setminus V$  do
39:         for  $\langle \phi', k', D' \rangle, \langle \phi'', k'', D'' \rangle \in \mathcal{R}_\uparrow$  do
40:             if  $\langle s[a], k, V \rangle \sim \langle \phi', k', D' \rangle \wedge$ 
41:                  $\langle s, k - 1, V \cup \{a\} \rangle \sim \langle \phi'', k'', D'' \rangle$  then
42:                  $\phi = Reg(\phi', a) \cup \phi''$ 
43:                  $D = D' \cup D'' \cup \{a\}$ 
44:                 add  $\langle \phi, k, D \rangle$  to  $\mathcal{R}_\uparrow$ 
45:                 return True
46:     return False
47: end function

```

```

48: function LMNonResilient( $s, k, V$ )
49:   for  $f \in \text{Landmarks}(F, A, s, G)$  do
50:     if  $|\text{Ach}(f, A \setminus V)| \leq k$  then
51:       return True
52:   return False
53: end function

54: function SelectCandidateDjGoals( $k, V, \mathcal{R} \uparrow$ )
55:    $G_\phi = \emptyset$ 
56:   for  $\langle \phi, k', D \rangle \in \mathcal{R} \uparrow$  do
57:     if  $k' \geq k \wedge D \cap V = \emptyset$  then
58:        $G_\phi.add(\phi)$ 
59:   return  $G_\phi$ 
60: end function

```

This function is called at the beginning of the algorithm on the initial state node $\langle s_0, K, \emptyset \rangle$ (line 1). By using the landmark pruning technique, RESPLAN2 can determine the non-resilience of the initial state before proceeding with the initialization, thus avoiding unnecessary computations. If the initial state may still be resilient, the algorithm continues with the initialization phase, as in the original implementation.

(1) Node Status Check. (line 5-7) After that, the recursive part of the algorithm starts, where it proves the resilience of the node $\langle s, k, V \rangle$ extracted from the *Open* list, continuing until the list is empty. The algorithm first checks if the node has already been classified. The new algorithm uses resilience certificates to determine if the node has been proven resilient. We formalize this by writing $\langle s, k, V \rangle \sim \langle \phi, k', D \rangle$ to mean that $s \models \phi$, $k \leq k'$, and $D \subseteq A \setminus V$. In

other words, the certificate $\langle \phi, D \rangle$ guarantees that s is k -resilient in the problem $\langle F, A \setminus V, s, G \rangle$. To evaluate non-resilience, the algorithm uses the set $R_{\downarrow}$, which contains the non-resilient nodes.

(2) ResilientCheck (RCHECK). (line 8) After verifying that the current node does not belong to the non-resilient sets or that there exists a valid resilience certificate to prove its resilience, the algorithm attempts to satisfy the recursive definition of k -resilient states using the resilience certificates. For $k \geq 1$, if there exist two resilience certificates $\langle \phi', k', D' \rangle$ and $\langle \phi'', k'', D'' \rangle$, and if there exists $a \in A(s) \setminus V$ such that

$$\langle s[a], k, V \rangle \sim \langle \phi', k', D' \rangle \quad \text{and} \quad \langle s, k-1, V \cup \{a\} \rangle \sim \langle \phi'', k'', D'' \rangle,$$

this corresponds to verifying both the “success” and “failure” branches of the recursive definition of resilient states, but using the resilience certificates. This allows RESPLAN2 to prove the resilience of $\langle s, k, V \rangle$ without invoking the planner again.

For the case $k = 0$, only the “success” branch needs to be considered, since 0-resilience does not tolerate any failures. In this case, a node $\langle s, 0, V \rangle$ can be proven resilient whenever there exists an action $a \in A(s) \setminus V$ and a resilience certificate $\langle \phi', k', D' \rangle \in R_{\uparrow}$ such that $\langle s[a], 0, V \rangle \sim \langle \phi', k', D' \rangle$.

After that, in both cases, RESPLAN2 generates a new certificate that extends the current certificates and adds it to the set of resilience certificates $R_{\uparrow}$. Starting from the two resilience certificates $\langle \phi', k', D' \rangle$ and $\langle \phi'', k'', D'' \rangle$, the algorithm generates the following:

- $\phi = \text{Reg}(\phi', a) \cup \phi''$, where ϕ' is the formula that holds in state $s[a]$ (the state after the action a succeeds), obtained through regression over the

action a , and ϕ'' is the formula that holds when $s \models \phi''$ after the action a fails. By combining these two formulas, we obtain a new formula ϕ that guarantees the state satisfies the conditions for both the success and the failure of action a .

- $D = D' \cup D'' \cup \{a\}$, which represents the set of actions necessary for the current certificate. This set is updated by taking the union of the actions required by both certificates, adding the action a that connects the two certificates.

For the case $k = 0$, when generating the new resilience certificate only the successor certificate $\langle \phi', k', D' \rangle$ needs to be considered. In this case, the new state formula is obtained solely by regressing ϕ' over the action a , $\phi = \text{Reg}(\phi', a)$, and the set of necessary actions becomes $D = D' \cup \{a\}$.

This results in the creation of a new certificate $\langle \phi, k, D \rangle$, which is compatible with the current node, $\langle s, k, V \rangle \sim \langle \phi, k, D \rangle$. The certificate is then added to the set of resilience certificates $R \uparrow$, enabling RESPLAN2 to reuse it in future resilience checks.

(3) Classical Planning Call. (line 11-30) If RCHECK fails, RESPLAN2 calls a classical planner. It constructs a new planning task where the primary objective remains the original goal G , but the search is augmented to actively look for opportunities to reconnect to previously verified resilient planning suffixes. Formally, we define the problem as:

$$\Pi' = \langle F, A \setminus V, s, G \cup G_\phi \rangle,$$

where the set G_ϕ acts as a collection of conditions representing already verified resilient suffixes. While the planner searches for a plan towards G , reaching any

state that satisfies a formula in G_ϕ is considered a successful termination of the search. These formulas correspond to states that have already been verified as resilient in previous computations, allowing the agent to graft the new plan onto an existing resilient strategy.

The set G_ϕ is computed by a function `SELECTCANDIDATEDJGOALS(k, V, R↑)` (line 11). Intuitively, the function scans all resilience certificates $\langle \phi, k', D \rangle \in R \uparrow$ and collects those that are valid for the current subproblem. A certificate is compatible if its resilience level is at least as high as the required level, $k' \geq k$, and if all actions required by the certificate are still available ($D \subseteq A \setminus V$). For every such certificate, the corresponding formula ϕ is added to G_ϕ .

By leveraging these formulas, RESPLAN2 can terminate the classical search early once a state satisfying any $\phi \in G_\phi$ is reached. This mechanism avoids the re-exploration of solved subproblems.

The algorithm then invokes the classical planner with `COMPUTEPLAN` (line 14), while restricting the search space to avoid known dead-ends. For the current pair (k, V) , we define the set

$$S_\downarrow := \{ s' \mid \langle s', k, V \rangle \in R_\downarrow \},$$

which contains states already classified as non-resilient. The planner is prohibited from visiting any state in $S_\downarrow$, ensuring that the search focuses exclusively on states that are either resilient or potentially resilient. There are two possible outcomes:

- **Failure (line 15):** no plan is found. In this case, RESPLAN2 keeps the same behaviour as in the original algorithm.
- **Success (line 17):** the planner returns a plan π with trajectory $\tau = (\tau_1, \dots, \tau_n)$ that achieves either the original goal G or one of the resilient formulas in

G_ϕ . Let G^* denote the goal condition satisfied in τ_n ; G^* can be either G itself or a formula $\phi \in G_\phi$.

- If $k = 0$ (line 26-30), by definition the plan π and its trajectory τ certify that each state s'_i reached along τ is 0-resilient in the restricted problem. The algorithm then records this information by constructing new resilience certificates that model these states. Starting from the achieved goal condition G^* and an empty set of required actions, D , new certificates are generated by regression. Let $\pi = \langle \pi_1, \dots, \pi_n \rangle$ be the plan returned by the planner. We initialize $\phi := G^*$ and $D := \emptyset$, and then iterate over the actions of π from the last to the first. For each action π_i (with i decreasing from n to 1), we update

$$\phi \leftarrow \text{Reg}(\phi, \pi_i) \quad \text{and} \quad D \leftarrow D \cup \{\pi_i\}.$$

After each update, the resilience certificate $\langle \phi, 0, D \rangle$ is added to the set $R \uparrow$, thus associating a 0-resilience certificate to the corresponding state along the trajectory.

- If $k \geq 1$ (line 18-25), for each transition $(\tau_i, \pi_i, \tau_{i+1})$ in τ , RESPLAN2 behaves as in the original algorithm when populating the open list, but it first checks whether the state τ_i might be non-resilient by applying landmark pruning through the function LMNONRESILIENT. If this function returns *True*, meaning that the state has at least one landmark without a sufficient number of achievers to prove resilience, then τ_i is immediately marked as non-resilient and the population of the open list for the rest of the trajectory is aborted: none of the states in τ_i is inserted into the open list. As a consequence, the subplan from τ_i to the goal state returned by the classical planner is discarded and

no further expansion is performed along this trajectory.

5.3.2 Plan Extraction

When the algorithm terminates, two cases are possible. If there does not exist any certificate that guarantees that the initial state is K -resilient, then the problem is declared *Unsolvable*, meaning that no K -resilient plan exists. Otherwise, if there exists a certificate $\langle \phi, K, D \rangle$ such that $s_0 \models \phi$, the algorithm reconstructs a valid plan by traversing the resilience certificates.

Starting from the initial state s_0 , the algorithm repeatedly searches for an action $a \in A(s)$ such that the successor $\langle s[a], K, \emptyset \rangle$ also has a resilience certificate in $R_\uparrow$. The first such action is appended to the plan, and the process continues from the new state $s[a]$. By iterating this selection until a goal state is reached, the algorithm builds a sequence of actions $\pi = (a_1, a_2, \dots, a_n)$ in which every intermediate state remains K -resilient, since the existence of the corresponding certificate is guaranteed.

Since all transitions follow states that can be proven resilient with certificates, the reconstructed plan is guaranteed to reach the goal and is valid under the bound of K tolerated failures. Moreover, since all states covered by resilience certificates in $R^\uparrow$ with $V = \emptyset$ are proved resilient through valid acyclic plans found by the classical planner, the set of K -resilient states reachable from s_0 is finite and acyclic; therefore the extraction procedure is guaranteed to terminate in a finite number of steps.

While RESPLAN2 is now capable of providing K -resilient plans leveraging the expressiveness of classical planning to ensure fast and effective extraction, it still inherits a fundamental limitation of plan-based solutions: it does not inherently provide information about alternative recovery paths that could ensure a solution if failures occur during the execution of the main plan.

With the introduction of our new implementation, the algorithm's capabilities are significantly extended. We move beyond the simple extraction of sequences to provide a more expressive solution that aligns with the formalization of non-deterministic planning: a Resilient Policy. This approach allows us to offer not only the resilient plan but also all alternative recovery plans that guarantee reaching the goal in case of failures.

5.4 Resilient Planning Policy

In this section, we define the resilient policy, a new representation for a solution to the resilient problem. The concept of policy node will be defined, as well as how it is built and how it responds to unforeseen events during execution.

5.4.1 Policy Definition

A resilient policy is formally defined as a function that, given an environment state (called a *policy node*), maps the correct action to undertake in order to guarantee the successful achievement of the goal, despite the failures that may occur during execution.

A policy node is essentially a representation of a decision point in the execution of a resilient plan. It encapsulates all the necessary data to understand the current context. Each policy node corresponds to a specific moment in time during execution; formally:

Definition 5. A policy node $p \in P$ is a tuple $\langle s, k, V \rangle$, where:

- s represents the state of the world;
- k denotes the current available number of failures that can occur;

- V is the set of actions that have already been attempted and failed.

By using these nodes to represent the domain of our solution, we can define the resilient policy as the mapping that guides the agent through every possible contingency.

Definition 6. Given a resilient problem $\Pi = \langle \langle F, A, s_0, G \rangle, K \rangle$, a resilient policy is a function:

$$\pi^* : P \rightarrow A$$

defined on a set of policy nodes P . For every node $p = \langle s, k, V \rangle \in P$, the assigned action $a = \pi^*(p)$ must satisfy the following validity properties:

1. **Applicability:** The action a is applicable in state s .
2. **Goal Reachability (Success):** The successful execution of a leads to a successor node that either satisfies the goal G or is itself in P .
3. **Resilience to Failure:** If the failure budget allows ($k > 0$), the failure of action a leads to a successor node that is in P .

The resilient policy can be viewed as a binary exploration tree that guides the execution of the resilient plan. Each policy node represents a transition point: from each node, at most two branches emerge. One branch represents the successful execution of the action dictated by the policy, leading to a successor state; the other represents the failure of that action, which is explored only if the remaining failure budget k allows it. The policy ensures that every node in this tree is resilient, maintaining a guaranteed path to the goal regardless of which branch is taken.

Given a resilient policy π^* , for any pair $(p, a) \in \pi^*$ where $p = \langle s, k, V \rangle$, the transition logic can be visualized as follows:

$$(p, a) \implies \begin{cases} p_{succ} = \langle s[a], k, V \rangle & \text{if } a \text{ succeeds} \\ p_{fail} = \langle s, k - 1, V \cup \{a\} \rangle & \text{if } a \text{ fails and } k > 0 \end{cases}$$

Starting from the initial policy node $p_0 = \langle s_0, K, \emptyset \rangle$, the sequence of successful branches represents the K -resilient plan. However, the policy structure ensures that even if a failure branch is triggered, the agent is immediately directed to a new node that is also part of the policy, providing a new recovery strategy until the goal is reached.

5.4.2 Extended Certificates for Policy Synthesis

The extraction of the resilient policy from the search results poses a significant computational challenge. A naive approach, based on iteratively searching through the set of resilience certificates (similar to standard plan extraction), was discarded due to its poor scalability. Since the number of certificates can grow exponentially with the tolerable failure budget K , a search-based extraction would lead to prohibitive latencies during the synthesis phase.

To eliminate the need for redundant search, we leverage the information already generated during the RESPLAN2 execution. Instead of treating certificates as passive state-properties, we extend their structure to be more expressive, embedding the policy transitions directly within the certificates. The resulting *extended resilience certificate* is defined as:

$$c = \langle \langle \phi, k, D \rangle, a, c_{succ}, c_{fail} \rangle$$

where:

- $\langle \phi, k, D \rangle$ is the original resilient certificate representation, where ϕ is a

state formula, k is the number of tolerable failures, and D is the set of actions necessary to verify resilience.

- a represents the action that can be applied in any state s that satisfies ϕ .
- c_{succ} is the pointer to the extended certificate proving resilience for the successor state $s[a]$ (the *success pointer*).
- c_{fail} is the pointer to the extended certificate proving resilience for state s in case action a fails (the *failure pointer*).

Algorithmic Modifications

This new representation affects the algorithm in how a certificate is extracted from the set $\mathcal{R}_\uparrow$ and how the new extended certificate is created and saved. For instance, the certificate selection condition (line 6 of Algorithm 2) is updated to account for the new certificate structure:

$$\langle s, k, V \rangle \notin \mathcal{R}_\downarrow \wedge \nexists \langle \phi', k', D' \rangle, a', c'_{succ}, c'_{fail} \in \mathcal{R}_\uparrow$$

The second modification concerns the `RCHECK` function. The mechanism must shift from a simple resilience check to correctly managing the extended resilience certificate representation. The goal is to prove that state s is k -resilient even if the actions in V are unavailable, by explicitly constructing a new certificate c from two existing, previously verified certificates c_{succ} and c_{fail} found in $\mathcal{R}_\uparrow$.

The policy composition process involves three main steps, as shown in the updated `RCHECK` function (Algorithm 2):

```
function RCHECK( $s, k, V$ )
  for  $a \in A(s) \setminus V$  do
```

```

for  $c_{succ} = \langle \langle \phi', k', D' \rangle, a', c'_{succ}, c'_{fail} \rangle \wedge$ 
       $c_{fail} = \langle \langle \phi'', k'', D'' \rangle, a'', c''_{succ}, c''_{fail} \rangle \in \mathcal{R}_\uparrow$  do
  if  $\langle s[a], k, V \rangle \sim \langle \phi', k', D' \rangle \wedge$ 
     $\langle s, k-1, V \cup \{a\} \rangle \sim \langle \phi'', k'', D'' \rangle$  then
     $\phi = \text{Reg}(\phi', a) \cup \phi''$ 
     $D = D' \cup D'' \cup \{a\}$ 
     $c \leftarrow \langle \langle \phi, k, D \rangle, a, c_{succ}, c_{fail} \rangle$ 
    add  $c$  to  $\mathcal{R}_\uparrow$ 
    return True
  return False
end function

```

1. **Certificate Extraction and Compatibility Check:** For a chosen action a , the algorithm searches for a pair of existing extended resilience certificates, c_{succ} and c_{fail} , that together cover the required resilience conditions for state s :
 - **Success Branch:** c_{succ} must prove the k -resilience of the successor state $s[a]$.
 - **Failure Branch:** c_{fail} must prove the $(k-1)$ -resilience of the current state s given that action a is now forbidden ($V \cup \{a\}$).
2. **New Certificate Synthesis:** Once a compatible pair (c_{succ}, c_{fail}) is found, a new state formula ϕ is derived through regression:

$$\phi = \text{Reg}(\phi', a) \cup \phi''$$

This ensures ϕ includes all necessary preconditions for both the successful plan and the fallback strategy.

3. **Certification Chaining:** The final step is the construction of the new certificate c . The previously verified certificates c_{succ} and c_{fail} are directly embedded as pointers. This direct linking forms a complete decision policy for the state s without requiring any further search.

The final modification concerns the post-processing phase of the initial plan extraction (within the `PROCESSPLAN` function). The process must now incorporate an explicit linkage between consecutive certificates:

```

if  $k = 0$  then
   $\phi \leftarrow G^*, D \leftarrow \emptyset$ 
   $c_{succ} \leftarrow \langle \langle G^*, 0, D \rangle, \emptyset, \emptyset, \emptyset \rangle$   $\triangleright$  Extended Certificate for the Goal state.
  for  $i = |\pi|$  down to 1 do
     $\phi \leftarrow \text{Reg}(\phi, \pi_i), D \leftarrow D \cup \{\pi_i\}$ 
     $c \leftarrow \langle \langle \phi, 0, D \rangle, \pi_i, c_{succ}, \emptyset \rangle$ 
    add  $c$  to  $\mathcal{R}_\uparrow$ 
     $c_{succ} \leftarrow c$ 

```

This method successfully transforms the linear plan π into a set of linked policy fragments stored in $\mathcal{R}_\uparrow$, enabling immediate and efficient policy look-up during execution. This modification allows us to directly obtain the necessary information for building the policy without repeating the search, significantly reducing computational time and improving system efficiency. By following these pointers recursively, the entire policy can be reconstructed in linear time relative to the number of nodes, transforming a complex search problem into a simple graph traversal.

5.4.3 Policy Generation

The final phase of the framework is the synthesis of the Policy Graph π^* . By leveraging the extended resilience certificates stored in $\mathcal{R}_\uparrow$, the algorithm reconstructs the complete strategy through a recursive traversal, starting from the initial state.

The process is defined as follows:

1. **Initialization:** An *open list* is initialized with the root policy node $p_0 = \langle s_0, K, \emptyset \rangle$.
2. **Exploration Loop:** The process iterates as long as the open list is not empty. In each iteration, a policy node $p = \langle s, k, V \rangle$ is extracted.
3. **Goal and Terminal States:** If the state s satisfies the goal condition G , the current branch of the policy is considered successful and terminates. Otherwise, the algorithm proceeds to the lookup phase.
4. **Certificate Lookup and Action Assignment:** The algorithm identifies an extended certificate $c = \langle \langle \phi, k^*, D \rangle, a, c_{succ}, c_{fail} \rangle \in \mathcal{R}_\uparrow$ such that the current node is compatible with the certificate's conditions ($\langle s, k, V \rangle \sim \langle \phi, k^*, D \rangle$). Once the matching certificate c is found, the action a is assigned as the policy decision for node p . The pair $\langle p, a \rangle$ is then added to the Policy Graph π^* .
5. **Strategic Propagation:** Two new subproblems (policy nodes) are generated by following the explicit pointers embedded within the certificate c :
 - **Success Branch:** The successor node resulting from the successful execution of a : $\langle s[a], k, V \rangle$.
 - **Failure Branch:** If $k > 0$, the recovery node resulting from the failure of a : $\langle s, k - 1, V \cup \{a\} \rangle$.

These nodes are added to the open list to be processed in subsequent iterations, effectively building the branches of the decision tree.

5.4.4 Policy Validation

To ensure the correctness and reliability of the generated resilient policy π^* , we developed a formal policy validator. The primary objective of this tool is to verify that the policy tree synthesized from the extended certificates $\mathcal{R}_\uparrow$ is sound: specifically, it proves that the strategy guarantees goal achievement for any failure sequence within the budget K . Beyond operational efficiency, the transition from a linear plan to a resilient policy provides significant advantages for formal verification. While a standard plan only identifies a single sequence of actions, the policy explicitly maps every recovery branch, effectively acting as a self-contained formal proof of the system's resilience.

Algorithm 3 Policy Validation Algorithm

```

1: function VALIDATEPOLICY( $p, \pi^*, G$ )
2:    $\langle s, k, V \rangle \leftarrow p$ 
3:    $P \leftarrow \text{dom}(\pi^*)$ 
4:   if  $s \models G$  then                                      $\triangleright$  Termination Condition
5:     return True
6:   if  $p \notin P$  then                                      $\triangleright$  Coverage Check
7:     fail "Policy Incomplete: Node not found."
8:    $a \leftarrow \pi^*(p)$ 
9:   if  $a \notin A(s)$  then                                    $\triangleright$  Action Applicability
10:    fail "Invalid Policy: Action not applicable."
11:    $isValid \leftarrow \text{VALIDATEPOLICY}(\langle s[a], k, V \rangle, \pi^*, G)$ 
12:   if  $k > 0 \wedge isValid$  then                              $\triangleright$  Closure under failures
13:      $isValid \leftarrow \text{VALIDATEPOLICY}(\langle s, k - 1, V \cup \{a\} \rangle, \pi^*, G)$ 
14:   return  $isValid$ 
15: end function

```

The validation process, detailed in Algorithm 3, performs an exhaustive recursive traversal of the policy tree starting from the initial node $p_0 = \langle s_0, K, \emptyset \rangle$. The verification steps are:

- **Goal Reachability:** The validator checks if the current state satisfies the goal (Line 4). This represents the successful termination condition for each branch.
- **Existence and Coverage:** Line 6 verifies that every reachable context is covered by the policy. If a node encountered during traversal is not present in the mapping, the policy is flagged as incomplete.
- **Action Applicability:** The tool ensures that the action a prescribed by the policy respects the domain constraints in state s (Line 9).
- **Completeness of Transitions:** All potential outcomes are verified:
 1. **Success Branch:** Line 11 validates the path following the successful execution of a .
 2. **Failure Branch:** If $k > 0$, the validator checks the recovery path (Lines 12–13), ensuring that even if a fails, the agent is directed to a valid recovery strategy.

The implementation of this validator provides a rigorous safety check, bridging the gap between theoretical resilience and practical reliability. By simulating all execution scenarios within the failure budget K , the tool confirms that the RESPLAN2 algorithm has synthesized a complete, closed-loop strategy. Ultimately, the validator ensures that for every sequence of modeled failures, a guaranteed recovery path exists, transforming the resilient plan into a formally verified execution policy.

Implementation Details

The policy validation tool has been fully implemented in Python, utilizing the Unified Planning (UP) framework [28]. The library is central to the validation process, as it provides the necessary structures to manage state representations and action semantics. Specifically, the validator leverages the UP library through the following components:

- **PDDL Parser:** This component allows the system to read domain and problem files, building the data structures to manage the environment and agent information.
- **Simulator:** Successor states are computed through the UP simulator, ensuring that the effects of actions are correctly applied according to the domain definition. The simulator is also used to verify action applicability by checking if an action’s preconditions are satisfied in a given state.

5.5 Theoretical Properties

The new features introduced in RESPLAN 2 do not alter the theoretical guarantees of the original algorithm. RESPLAN 2 retains the core properties of correctness, completeness, and termination within a finite number of iterations, provided the underlying classical planner used in COMPUTEPLAN possesses the same properties.

Theorem 10. *RESPLAN 2 is sound, complete, and terminates after a finite number of iterations.*

Proof (sketch). The early detection mechanism exploiting landmarks only labels a state as non-resilient when the sufficient condition in Theorem 8 is satisfied.

Since this condition is conservative, no resilient state will be incorrectly labeled as non-resilient, thus preserving completeness.

The adaptive replanning strategy connects the current candidate plan prefix to a previously verified resilient suffix. By Theorem 9 and the compatibility condition $\langle s, k, V \rangle \sim \langle \phi, k', D \rangle$, reconnecting a plan prefix to a previously certified suffix preserves soundness. Importantly, the remaining plan prefix is still subject to verification by the algorithm, ensuring correctness.

Additionally, the goal state G is always included in the disjunctive goal set G_ϕ of any subproblem $\langle s, k, V \rangle$ since the original goal G is always explicitly included in the planning goal $G \cup G_\phi$ of every subproblem. This guarantees that an entirely new plan to the goal remains a valid outcome, preserving completeness.

Hence, since RESPLAN is sound, complete, and terminates in a finite number of iterations, as demonstrated in [1] and discussed in Subsection 4.1.4, and since the algorithmic revisions introduced in RESPLAN2 do not affect these properties, we can conclude that these guarantees also hold for RESPLAN2. $\square$

Chapter 6

Experimental Evaluation

6.1 Introduction to the Results

In this chapter, we present our comparative results. We evaluate our RESPLAN algorithm by taking classical planning instances and scaling $K \in \{1, 2, 3, 4\}$, across two classes of domains. For the first class, we select domains from the International Planning Competition (IPC) where redundancy naturally yields alternative ways to achieve the goals, hence K -resilient solutions are plausible. Transportation domains are good candidates: they typically feature multiple vehicles for load/delivery and navigation graphs that permit alternative routes. Concretely, we selected four IPC domains: in DRIVERLOG, drivers must be co-located and the road network is not fully connected; SATELLITE provides partially overlapping instrument sets across satellites; STORAGE introduces hoists for moving crates; and in ZENOTRAVEL aircraft can fly at two speeds and consume fuel.

For the second class, we selected three domains specifically designed to exhibit high resilience: BLOCKSWORLDMMA, ROCKETS, and LOGISTICS. In these do-

mains, we deliberately increased redundancy to ensure that instances remain solvable at higher values of K . For example, in BLOCKSWORLDMA, we added more robotic arms and blocks; in ROCKETS, we increased the number of rockets and cargo opportunities; and in LOGISTICS, we considered multiple cities with additional locations and vehicles, providing alternative routes and carriers.

In the first part of our evaluation, we compare our approaches, RESPLAN and RESPLAN2, against a FOND compilation solved with MYND and FOND-SAT. Both are capable of computing strong policies that preserve the properties required by resilient planning. Following this comparison, we present an ablation study between RESPLAN2 and RESPLAN to isolate the contributions of our algorithmic enhancements.

Finally, we further investigate the structure of the domains: first, we analyze the landmarks for the benchmarks used to evaluate our algorithm; then, we provide an overview of the computation time spent by the algorithm and the quality of the solutions obtained; and lastly, we assess the resilience across all IPC domains to provide a comprehensive evaluation of the framework.

We evaluate four resilience levels with a time limit of 1800 s and 32 GB RAM per instance, on a Xeon Gold 6140M @ 2.3 GHz. As the classical planner, we use *Fast Downward* [17] with the h_{ff} heuristic to compute plans and we compute fact-landmarks using the backchaining method over the Relaxed Planning Graph proposed by Zhu and Givan [35].

We then provide a more in-depth analysis of the effectiveness of the new techniques introduced in RESPLAN2.

Domain	Sol	$K = 1$				$K = 2$				$K = 3$				$K = 4$			
		MN	FS	RP	RP2	MN	FS	RP	RP2	MN	FS	RP	RP2	MN	FS	RP	RP2
Driverlog (#20)	S	2	1	14	14	0	0	1	1	0	0	0	0	0	0	0	0
	U	0	-	0	0	0	-	0	5	0	-	0	13	1	-	0	15
Satellite (#36)	S	1	0	15	17	1	0	4	11	0	0	1	3	0	0	0	0
	U	1	-	1	4	0	-	1	10	0	-	1	15	0	-	1	18
Storage (#30)	S	4	0	10	11	0	0	3	3	0	0	0	0	0	0	0	0
	U	6	-	5	5	9	-	7	8	9	-	6	7	12	-	5	8
Zenotravel (#20)	S	3	1	11	13	1	1	2	2	0	0	0	0	0	0	0	0
	U	1	-	1	1	0	-	1	6	1	-	2	13	2	-	2	13
Subtotal-IPC (#106)	S	10	2	50	55	2	1	10	17	0	0	1	3	0	0	0	0
	U	8	-	7	10	9	-	9	29	10	-	9	48	15	-	8	54
BlocksworldMA (#10)	S	5	6	10	10	0	2	10	10	0	0	4	6	0	0	3	5
	U	0	-	0	0	0	-	0	0	0	-	0	4	0	-	0	4
Logistics (#10)	S	0	1	10	9	0	0	9	9	0	0	7	7	0	0	3	3
	U	0	-	0	0	0	-	0	0	0	-	0	0	0	-	0	0
Rocket (#10)	S	4	4	10	10	1	0	9	9	0	0	3	6	0	0	0	0
	U	0	-	0	0	0	-	0	0	0	-	0	0	0	-	0	4
Subtotal-Res (#30)	S	9	11	30	29	1	2	28	28	0	0	14	19	0	0	6	8
	U	0	-	0	0	0	-	0	0	0	-	0	4	0	-	0	8
Total (#136)	S	19	13	80	84	3	3	38	45	0	0	15	22	0	0	6	8
	U	8	-	7	10	9	-	9	29	10	-	9	52	15	-	8	62

Table 6.1: Coverage results for RESPLAN (RP), RESPLAN2 (RP2), and compilation into FOND using MyND (MN) or FOND-SAT (FS).

6.2 Comparison among RESPLAN2, RESPLAN and FOND

This comparison analysis begins with the coverage results across the two benchmark sets and proceeds to the runtime evaluation.

6.2.1 Coverage results

Table 6.1 summarizes the coverage results, distinguishing between solvable (S) and unsolvable (U) instances to provide a granular analysis of algorithmic per-

formance. Columns MN and FS report the results for the FOND compilations solved by MyND and FOND-SAT, respectively. The specialized resilient planning approach is represented by the original *RESPLAN* (RP) and the enhanced *RESPLAN2* (RP2). Notably, since FOND-SAT is a SAT-based planner, it cannot prove if a problem does not have a solution; such cases are marked with “–” in the table.

Analyzing the IPC domains, K -resilient plans are predominantly found for $K \leq 2$, as these models typically offer limited structural redundancy. Within this solvable regime, resilient planning significantly outperforms FOND-based compilations. At $K=1$, both *RESPLAN* and *RESPLAN2* achieve high coverage (up to 55 solutions) compared to only 10 for MyND and 2 for FOND-SAT. This advantage is maintained at $K=2$, where *RESPLAN2* identifies 17 solutions, whereas the FOND baselines fail to exceed 2.

For unsolvable cases, the original *RESPLAN* shows limitations, proving unsolvability less frequently than MyND as K increases. This gap is due to the depth-first nature of *RESPLAN*, which is optimized for plan construction rather than state-space refutation. However, *RESPLAN2* fundamentally changes this picture: by integrating landmark-based pruning, it overcomes the constraints of its predecessor. As shown in Table 6.1, at $K = 4$ *RESPLAN2* certifies 62 unsolvable instances, far surpassing both MyND (15) and the original *RESPLAN* (8). This confirms that the specialized pruning in *RESPLAN2* allows it to accumulate evidence for impossibility more efficiently than the symbolic search of the FOND compilation, establishing *RESPLAN2* as the most effective tool for both construction and refutation.

Overall, while the original *RESPLAN* was the preferred choice for constructing K -resilient solutions, it relied on the complementarity with FOND planners like MyND to promptly refute infeasible instances. *RESPLAN2* effectively internal-

izes and improves upon this refutation capability. By combining efficient plan construction with landmark-based pruning, *RESPLAN2* eliminates the need for such complementarity, outperforming both *RESPLAN* and *MyND* across the entire spectrum of solvable and unsolvable instances.

In high-resilience domains, such as *BLOCKSWORLDMA*, *ROCKET*, and *LOGISTICS*, many instances admit plans with $K \geq 3$. In these scenarios, the contrast between specialized resilient planning and *FOND* baselines is most evident: for $K = 3$ and $K = 4$, both *MyND* and *FOND-SAT* fail to solve any instance. In contrast, *RESPLAN2* (RP2) demonstrates exceptional scalability, solving up to 22 solvable instances at $K = 3$ and maintaining robust coverage even where the original *RESPLAN* begins to degrade due to the exponential growth of the state space.

6.2.2 Runtime results

Figure 6.1 presents a detailed runtime comparison of *RESPLAN2* against the *FOND* baseline *MyND* and the original *RESPLAN*. By separating solvable instances (purple) from unsolvable ones (green), the plots make the outcome-dependent behavior of the systems explicit. *FOND-SAT* is excluded because its low overall coverage yields too few data points for a meaningful comparison.

The first row compares *MyND* against the baseline *RESPLAN*, illustrating the initial trade-off of the k -resilient search. While *RESPLAN* demonstrates a clear construction advantage on most solvable instances, it struggles against the *FOND* compilation on unsolvable cases. In these instances, the symbolic search of *MyND* often proves impossibility faster than a standard state-space search, as evidenced by the green diamonds situated above the diagonal near the y -axis.

This dynamic changes significantly in the second row, where *RESPLAN2* is compared against *MyND*. Here, *RESPLAN2* demonstrates superior scalability across

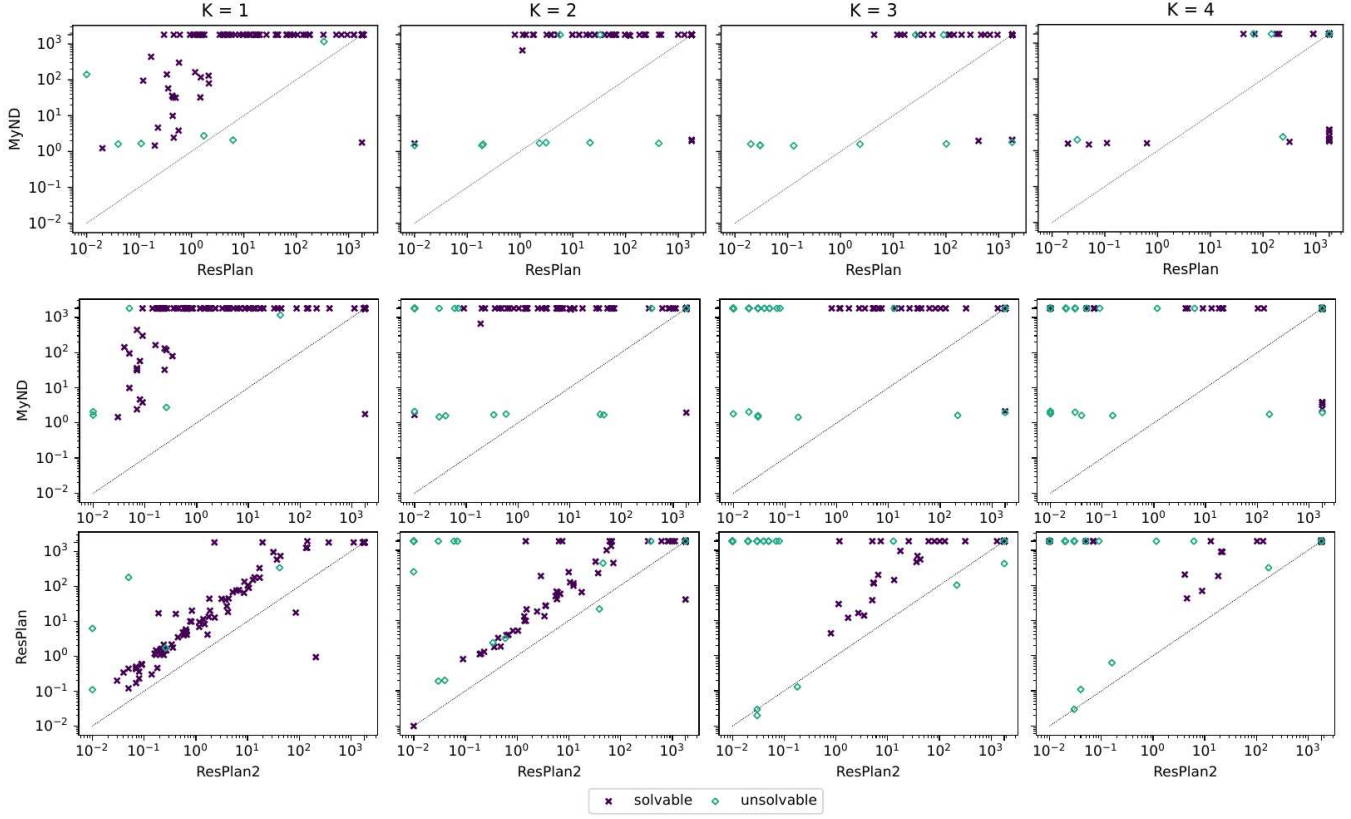


Figure 6.1: Runtime comparison across all benchmarks. The first row compares MyND (y-axis) against RESPLAN (x-axis). The second row compares MyND (y-axis) against RESPLAN2 (x-axis), while the third row shows a direct comparison between RESPLAN (y-axis) and RESPLAN2 (x-axis). Each point represents the execution time in seconds for a specific instance.

all resilience levels and, most importantly, eliminates the previous disadvantage on unsolvable instances. The integration of landmark-based pruning allows RESPLAN2 to dominate the FOND baseline even on non-resilient problems, effectively shifting the cluster of green diamonds below the diagonal. This reflects the algorithm’s ability to accumulate evidence for non-resilient much earlier than the FOND compilation, providing a consistent speedup regardless of whether a K -resilient plan exists.

Finally, the third row provides a direct comparison between RESPLAN2 and RESPLAN, highlighting the impact of our architectural enhancements. While performance at $K = 1$ is similar, the performance gap becomes significant as the resilience requirement increases ($K > 1$). RESPLAN2 effectively eliminates many of the timeouts experienced by the original search, particularly on unsolvable cases at $K = 3$ and $K = 4$. Although a small number of instances at intermediate resilience are solved slightly faster by RESPLAN, due to the minor computational overhead of landmark bookkeeping, this does not affect the overall trend. The data confirms that RESPLAN2 maintains a dominant construction advantage on resilient problems while providing a much faster decision on infeasible ones, resulting in higher overall coverage and efficiency.

6.2.3 Cumulative Coverage over Time

Figure 6.2 illustrates the cumulative number of instances solved over time for both versions of our resilient algorithms and the two baselines. Overall, our specialized approaches demonstrate a clear dominance across all values of K .

At $K = 1$, RESPLAN2 and RESPLAN exhibit nearly identical performance, both significantly outperforming the baselines by solving approximately 60 instances in less than 10 seconds. This confirms that for low resilience values, the additional computational overhead of landmark-based pruning in RESPLAN2 is negligible compared to the efficiency of the core resilient search.

As K increases, the superiority of RESPLAN2 becomes most evident:

- **Efficiency and Scalability:** At $K = 2$, while RESPLAN remains highly efficient, RESPLAN2 begins to pull ahead as the search space grows, maintaining a steeper coverage curve.

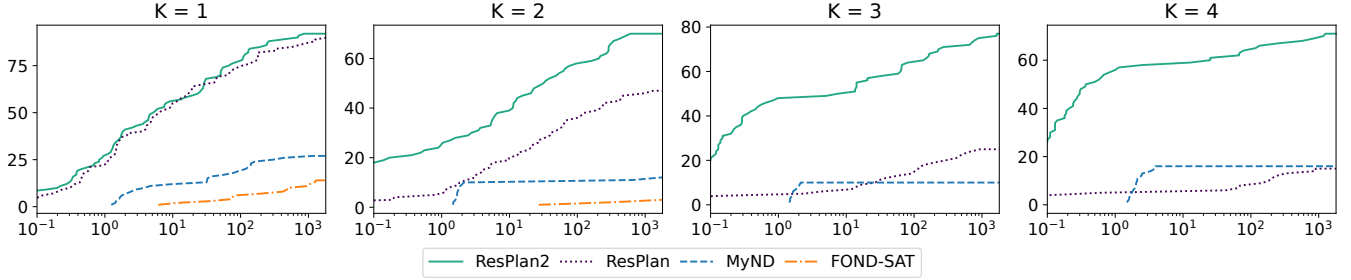


Figure 6.2: Coverage (y-axis) vs runtime (x-axis) for increasing values of K .

- Closing the Refutation Gap:** For $K = 3$ and $K = 4$, we observe that MyND is able to solve a small subset of instances (exclusively by proving unsolvability) in just a few seconds. However, while the original RESPLAN curve previously stagnated below MyND in the very first seconds of search, RESPLAN2 effectively overcomes the limitations of the original algorithm. Thanks to its integrated pruning, RESPLAN2 surpasses the baseline's early refutation speed and continues to certify unsolvability for many instances long after MyND has reached its plateau.
- Baseline Performance:** Across all tested resilience levels, FOND-SAT remains the least performant, struggling to scale as the failure budget K increases.

In summary, the cumulative coverage analysis establishes RESPLAN2 as the most robust tool for K -resilient planning. It successfully combines the fast construction capabilities of the original resilient search with a significantly more powerful refutation mechanism, resulting in an anytime performance that is superior to both its predecessor and state-of-the-art FOND-based compilations.

Domain	Sol	$K = 1$				$K = 2$				$K = 3$				$K = 4$			
		RP	LRP	ARP	RP2	RP	LRP	ARP	RP2	RP	LRP	ARP	RP2	RP	LRP	ARP	RP2
Driverlog (#20)	S	14	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
	U	0	0	0	0	0	4	5	5	0	13	0	13	0	15	0	15
Satellite (#36)	S	15	2	2	2	4	2	7	7	1	1	2	2	0	0	0	0
	U	1	3	0	3	1	9	0	9	1	14	0	14	1	17	0	17
Storage (#30)	S	10	1	1	1	3	0	0	0	0	0	0	0	0	0	0	0
	U	5	0	0	0	7	1	0	1	6	1	0	1	5	3	1	3
Zenotravel (#20)	S	11	2	3	2	2	0	0	0	0	0	0	0	0	0	0	0
	U	1	0	0	0	1	5	0	5	2	11	0	11	2	11	0	11
BlocksworldMA (#10)	S	10	0	0	0	10	0	0	0	4	0	2	2	3	0	2	2
	U	0	0	0	0	0	0	0	0	0	4	0	4	0	4	0	4
Logistics (#10)	S	10	-1	-1	-1	9	0	0	0	7	0	0	0	3	0	0	0
	U	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Rocket (#10)	S	10	0	0	0	9	0	0	0	3	0	3	3	0	0	0	0
	U	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	4
Subtotal (#136)	S	80	4	5	4	38	2	7	7	15	1	7	7	6	0	2	2
	U	7	3	0	3	9	20	5	20	9	43	8	43	8	54	1	54
Total Coverage	C	87	94	92	94	47	69	59	74	24	68	39	74	14	68	17	70

Table 6.2: Coverage difference with respect to the baseline *RESPLAN* (RP) for Landmark enhancement (LRP), Adaptive Planning enhancement (ARP) and *RESPLAN2* (RP2)

6.2.4 Ablation Study

Table 6.2 presents a comprehensive comparison of coverage across the baseline *RESPLAN* and all configurations of the enhanced *RESPLAN2* system: *RESPLAN* with landmark-based pruning only (LRP), *RESPLAN* with adaptive replanning only (ARP), and the full implementation combining both features (RP2). This comparison analyzes the behavior of the individual enhancements and their combination across solvable (S) and unsolvable (U) instances for resilience levels $K \in \{1, 2, 3, 4\}$. Absolute coverage is reported for the baseline *RESPLAN*, whereas for the other variants only the difference relative to *RESPLAN* is shown. Presenting deltas emphasizes the marginal contribution of each component, reduces redundancy, and facilitates comparisons across values of K .

- **Landmark-based pruning (LRP).** Coverage improves markedly on unsolvable instances. The advantage becomes more pronounced as K increases, since unsolvable cases tend to be more frequent and the cost of exhaustively certifying unsolvability grows quickly with the size of the search space. Pruning discards large portions of the search space early, yielding significant gains.
- **Adaptive replanning (ARP).** The replanning strategy yields a modest but consistent improvement on solvable instances by avoiding repeated exploration after failures. The only exception occurs in *LOGISTICS* at $K = 1$, where coverage decreases by one instance. This domain was handcrafted in prior work to admit resilient plans; as a result, the need for replanning is reduced and the overhead of the adaptive strategy may not yield a net benefit.

RESPLAN augmented with landmark-based pruning (LRP) achieves higher coverage than *RESPLAN* with adaptive replanning (ARP) because pruning overcomes the main algorithmic limitation: to declare an instance non-resilient, the search must often explore the entire space and certify that no sufficient set of resilient plans exists.

The highest coverage is obtained by the full *RESPLAN2* configuration (RP2), which combines landmark-based pruning and adaptive replanning.

6.2.5 Summary

In conclusion, while the original *RESPLAN* excelled at plan construction but relied on *FOND* baselines for efficient refutation, *RESPLAN2* successfully bridges this gap. By internalizing powerful pruning techniques, *RESPLAN2* establishes

itself as a superior, all-in-one framework that dominates both construction and refutation tasks across all resilience levels.

6.3 Landmark Structural Analysis

To provide a quantitative justification for the effectiveness of the pruning technique in RESPLAN2, we conducted a structural analysis of the benchmark domains. This analysis focuses on landmarks and their respective achievers, which are the actions capable of satisfying them.

Table 6.3 summarizes these properties for each domain, reporting the average number of landmarks per instance and the average number of achievers for the critical landmark (the one with the fewest achievers in the instance). Additionally, the table shows the percentage of instances where the number of achievers for the critical landmark is less than 3 and 4, respectively. These data highlight the structural vulnerability of the instances to action failures and provide an immediate indication of which instances cannot be 3-resilient (those with fewer than 3 achievers for a critical landmark) or 4-resilient.

Domain	Avg. Landmark For Instance	Avg. Achievers On Critical Landmark	Instance with less than 3 achievers on critical landmark	Instance with less than 4 achievers on critical landmark
Driverlog	30.0	3.4	20.0%	65.0%
Satellite	68.3	7.2	16.7%	22.2%
Storage	17.6	45.4	3.3%	10.0%
Zenotravel	20.3	4.3	30.0%	60.0%
BlocksworldMA	11.0	4.2	0.0%	40.0%
Logistics	7.6	5.0	0.0%	0.0%
Rocket	6.8	5.2	0.0%	0.0%

Table 6.3: Structural properties of landmarks across domains. Achiever counts refer to the minimum number of distinct actions capable of satisfying the most constrained landmark in each instance.

Key Observations

The data highlight how the domain topology dictates an upper bound on the obtainable resilient plans, thereby determining the success of our specialized techniques:

- **Plan Depth and Sequential Complexity:** The *Avg. Landmark Per Instance* column reflects the complexity of each domain. For instance, SATELLITE exhibits a notably high number of landmarks (68.3), suggesting that plans consist of long chains of mandatory steps. Despite a high average redundancy (7.2 achievers), we observe a drastic drop in coverage between $K=1$ and $K=2$. This suggests that in complex domains, even when structural redundancy is theoretically high, the probability of encountering a bottleneck increases with the number of sequential constraints, making it significantly harder to find a fully resilient solution as K increases.
- **Identification of achievers for the critical landmark:** The *Avg. Achievers On Critical Landmark* column represents the maximum value of failure to find a valid K -resilient plan. This information provides an intrinsic property of the domain: its resilience capacity. In domains such as DRIVERLOG (3.4), BLOCKSWORLDM (4.2), and ZENOTRAVEL (4.3), the presence of landmarks with few achievers suggests the impossibility of finding solutions with high resilience values. Indeed, based on the results obtained, most instances prove to be non-resilient for $K \geq 4$ (and in many cases, as early as $K = 3$).
- **Correlation with Pruning Speed:** The percentage columns indicate the probability that an instance will be pruned “instantaneously” for a given value of K . In ZENOTRAVEL, 60% of instances exhibit critical bottlenecks for

$K \geq 3$. This structural fragility explains the dramatic increase in refutation speed: RESPLAN2 eliminates entire search subtrees as soon as a landmark's achievers fall below the K threshold.

- **Structural vs. Algorithmic Resilience:** High-resilience domains like ROCKET never show fewer than 5 achievers for critical landmarks. In these cases, the domain is structurally wide enough to survive most failures; the performance advantage of RESPLAN2 thus stems from the integration of Adaptive Replanning rather than pruning, allowing for the efficient management of the numerous available alternative paths.

6.4 Computational Overhead and Runtime Distribution

To quantify the overhead introduced by the proposed enhancements, we analyze the distribution of computing time across the different phases of the RESPLAN2 algorithm (Figure 6.3). This analysis directly addresses the need to evaluate the computational cost of landmark generation and certificate management.

The monitored phases are:

- **Search Time:** The time spent by the classical planner for (re)planning.
- **RCheck:** Verification of state resilience using certificates.
- **Fill Open List:** Regeneration of states and certificates via regression after each replan.
- **Certificate Selection:** Selection of suitable resilience certificates to be used as disjunctive goals.

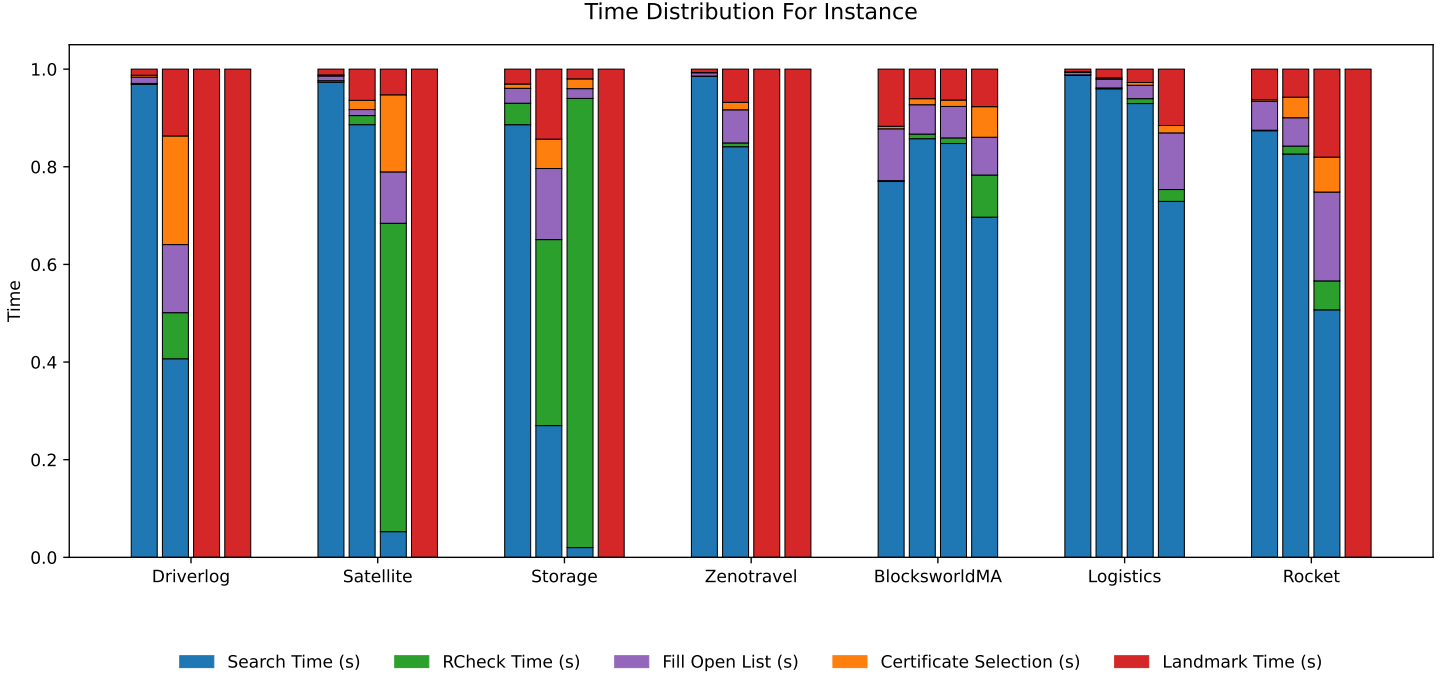


Figure 6.3: Runtime distribution by domain and phase (Search, RCheck, Fill Open List, Certificate Selection, Landmark) for $K = 1$ to $K = 4$.

- **Landmark Time:** Extraction and evaluation of landmarks for pruning purposes.

The normalized results in Figure 6.3 show that the time dedicated to landmark generation (Landmark Time, in red) is negligible across all domains. This confirms that the effectiveness of pruning does not compromise overall performance. In complex domains such as SATELLITE, a relative increase in RCheck time is observed, which is consistent with the high number of sequential landmarks requiring constant verification. Conversely, in fragile IPC domains, the system shifts the computational load from plan search to landmark-based verification as K increases. This confirms that the algorithm successfully avoids the exploration of non-resilient branches by identifying structural bottlenecks early

in the search process.

It is important to emphasize that the time dedicated to landmark extraction and the subsequent structural analysis is negligible. As illustrated in the runtime comparisons, the performance of RESPLAN2 remains nearly identical to the base version at low resilience levels ($K = 1$), where pruning is less frequent. This demonstrates that the landmark-based mechanism does not introduce a significant computational burden. Instead, it functions as an efficient pre-processing step that provides immediate refutation for non-resilient instances without hindering the plan construction speed when a solution exists.

6.5 Solution Quality Analysis

In this section, we evaluate the impact of k -resilient plans with respect to optimal plans (or standard classical plans when finding the absolute optimum becomes computationally prohibitive). This analysis quantifies the cost of resilience, the additional execution steps required to guarantee goal achievement under a failure budget compared to a deterministic baseline ($K = 0$).

The results in Table 6.4 report the plan lengths for each instance, assuming a unit cost for every action. In the “Opt” column, the values represent the deterministic baseline: values in black are optimal, while values in red refer to plans generated using the same configuration as the RESPLAN algorithm, providing a direct reference for comparison in larger instances. A symbol “X” indicates that the problem has been proven to be non-resilient at that specific K level, otherwise “–”

As shown by the data, while we did not necessarily expect resilient plans to match the deterministic optimum, the results are remarkably close across the majority of instances. The most significant finding, however, is the stability of

plan lengths as the failure budget K increases. For most problems, the length of a 1-resilient plan is identical to that of a 4-resilient plan.

This stability suggests that resilience is an intrinsic characteristic of the domain's structure rather than a mere side effect of plan length. A plan that is successfully built to be k -resilient often already exploits the maximum redundancy available in the environment, making it inherently resilient for higher values of K up to a structural threshold.

Domain	Prob.	Opt	$K = 1$	$K = 2$	$K = 3$	$K = 4$
DRIVERLOG	p01	7	8	X	–	X
	p02	15	22	X	X	X
	p03	12	13	X	X	X
	p04	16	16	X	X	X
	p05	18	21	X	X	X
	p06	11	13	–	X	X
	p07	13	15	–	X	X
	p08	19	23	–	X	X
	p09	24	40	–	X	X
	p10	17	21	21	X	X
	p11	22	29	–	X	X
	p12	38	52	–	X	X
	p13	28	45	–	X	X
	p14	41	45	–	X	X
SATELLITE	p03	11	11	X	X	X
	p04	17	26	X	X	X
	p05	15	21	21	21	–
	p07	21	28	28	X	X
	p08	20	27	27	27	X
	p09	27	37	37	37	X
	p10	33	35	35	X	X
	p11	35	36	X	X	X
	p13	57	78	–	X	X
	p14	40	47	47	–	–
	p15	47	64	–	–	X
	p16	48	62	62	–	–
	p17	43	49	49	–	–
	p18	32	36	36	X	X
	p19	61	68	68	–	–
	p21	74	86	–	–	–
	p23	100	145	–	–	–
STORAGE	p05	8	9	X	X	X
	p06	8	9	X	–	X
	p08	12	14	–	–	–
	p09	11	14	14	–	–
	p11	17	20	–	–	–
	p12	16	20	20	–	–
	p14	18	26	–	–	–
	p15	18	24	–	–	–
	p16	21	24	24	–	–
	p17	33	33	–	–	–
	p18	41	41	–	–	–
Domain	Prob.	Opt	$K = 1$	$K = 2$	$K = 3$	$K = 4$
ZENOTRAVEL	p01	1	1	1	–	–
	p03	6	6	X	X	X
	p04	8	9	X	X	X
	p05	11	12	X	X	X
	p06	11	16	X	X	X
	p07	15	32	X	X	X
	p08	11	16	16	X	X
	p09	21	23	–	X	X
	p10	22	24	–	X	X
	p11	14	22	–	X	X
	p12	21	23	–	X	X
	p13	26	32	–	X	X
	p15	51	53	–	–	–
BLOCKSWORLDMA	p01	3	3	3	X	X
	p02	2	2	2	X	X
	p03	3	3	3	X	X
	p04	4	4	4	X	X
	p05	5	5	5	5	5
	p06	6	6	6	6	6
	p07	6	6	6	6	6
	p08	9	9	9	9	9
	p09	11	13	13	13	13
	p10	18	20	20	20	–
LOGISTICS	p01	3	3	3	3	3
	p02	6	6	6	6	–
	p03	6	6	6	6	6
	p04	6	6	6	6	6
	p05	12	12	12	12	–
	p06	13	14	14	14	–
	p07	13	14	14	14	–
	p08	12	12	12	–	–
	p09	21	21	21	–	–
ROCKET	p01	4	4	4	4	X
	p02	6	7	7	–	X
	p03	6	6	6	6	X
	p04	7	7	7	7	X
	p05	6	7	7	7	–
	p06	6	6	6	6	–
	p07	9	10	10	10	–
	p08	11	12	12	–	–
	p09	17	19	19	–	–
	p10	26	31	–	–	–

Table 6.4: Solution Quality Analysis: comparison of plan lengths ($K \in \{1, \dots, 4\}$) against optimal, or with h_{ff} marked in red. Table split into two halves for readability.

The results in Table 6.4 highlight the efficiency of the RESPLAN2 algorithm in preserving solution quality. In the BLOCKSWORLDM domain, where optimal deterministic costs were computed as a reference, the k -resilient plans frequently match the optimal plan length. This indicates that RESPLAN2 successfully identifies inherent structural redundancies in the domain topology, providing a safety budget without any additional execution cost.

6.6 Policy Analysis

The introduction of the resilient policy alongside the resilient plan allows for the possibility to study all scenarios that could happen in case of failure. While the original RESPLAN2 algorithm provided only the primary resilient path, the policy synthesis explicitly contains all recovery strategies.

This approach enables a more in-depth evaluation of the solution's properties: specifically, the longest path that guarantees reaching the goal despite failures (Worst Plan), the length of the K -resilient plan (Res Plan), and the total number of actions included in the policy.

Table 6.5 reports these values across different domains, highlighting the overhead required to ensure success under adversarial conditions. These results provide several insights into the relationship between resilience and plan execution. Since the synthesized plans are not necessarily optimal, the analysis focuses on the general behavior of the recovery strategies.

A key observation is that the number of failures K is not strictly correlated with a fixed increase in the worst-case plan length. A single action failure does not imply the addition of just one recovery step; instead, it can lead to a significant detour depending on the state and domain constraints.

To illustrate this non-linearity, consider a vehicle trip analogy: if a tire gets

Domain	K	Avg. Actions	Avg. Res Plan	Avg. Worst Plan	Overhead
DRIVERLOG	1	118.8	23.2	47.7	2.06x
	2	219.0	21.5	57.0	2.65x
SATELLITE	1	223.1	47.7	64.9	1.36x
	2	411.8	30.2	49.0	1.62x
	3	223.0	23.0	50.0	2.17x
STORAGE	1	89.1	19.2	33.2	1.73x
	2	205.7	17.3	35.3	2.04x
ZENO	1	91.0	23.1	34.6	1.50x
	2	92.0	7.0	17.0	2.43x
BLOCKSWORLDMA	1	26.1	7.1	14.2	2.00x
	2	58.0	7.0	19.4	2.77x
	3	164.0	9.8	31.0	3.15x
	4	188.8	7.8	27.6	3.54x
LOGISTICS	1	52.4	10.4	18.9	1.81x
	2	144.4	10.4	27.6	2.64x
	3	227.0	8.7	32.3	3.70x
	4	208.0	5.0	34.3	6.87x
ROCKET	1	58.2	10.7	19.6	1.83x
	2	142.7	8.6	29.7	3.47x
	3	184.3	6.7	36.5	5.48x

Table 6.5: Average Policy Complexity and Plan lengths by domain and K value.

flat, it can be replaced with a spare, adding only a few steps to the plan. However, if a second failure occurs ($K = 2$) and no more spare tires are available, the agent is forced into a complete change of strategy, such as calling a tow truck to reach a repair shop. In this case, the second failure makes reaching the destination significantly more arduous and longer than the first.

This behavior is clearly reflected in the benchmarks: as the resilience budget K increases, the worst-case plan length often grows substantially. In some

scenarios, providing a guarantee against multiple failures requires the policy to include alternative paths that are several times longer than the nominal plan. This demonstrates that while the policy ensures goal reachability, the cost of this guarantee in terms of execution steps can escalate rapidly as the required degree of protection rises, reflecting the non-linear "price of resilience" in constrained environments.

6.7 Evaluating Resilience of IPC Benchmarks

Table 6.6 shows the number of instances, grouped by IPC edition from 1998 to 2023, evaluated with resilience values $K = 1$ and $K = 2$. The table reports coverage (in percentage) for both RESPLAN and RESPLAN2. For each IPC, both solvable and unsolvable instances are considered.

With this analysis, we aim to understand to what extent resilience is an intrinsic property of the domains used in the competitions and as benchmarks by the research community.

From the table, a consistent trend emerges: the number of instances that turn out to be resilient according to RESPLAN and RESPLAN2 decreases (with the exception of IPC 2023). In particular, starting from IPC 2002, the percentages of solved problems drop sharply, with IPC 2000 and IPC 2011 reporting 0 solved instances. This suggests that, as IPC domains evolved, problem complexity has shifted toward finding a single valid and complex plan rather than supporting a broader exploration of alternatives, leading to a smaller number of cases that can be effectively solved in the resilient setting. For $k = 1$, IPC 1998 stands out with a relatively higher resiliency rate of 20% solved instances, while the best rate is observed in 2002, with 33.6%. By contrast, IPC 2023, although still low, shows a slight improvement with 11.0%. It is interesting to note that the percentages

Domains	Type	$K = 1$		$K = 2$	
		RP (%)	RP2 (%)	RP (%)	RP2 (%)
IPC 1998	s	17.7	20.0	9.1	9.1
(#164 – $D = 5$)	u	25.0	53.3	23.8	61.0
IPC 2000	s	0.0	0.0	0.0	0.0
(#90 – $D = 3$)	u	58.9	100.0	58.9	100.0
IPC 2002	s	27.0	33.6	2.5	3.3
(#122 – $D = 6$)	u	4.1	31.1	4.9	43.4
IPC 2004	s	7.5	7.5	7.5	7.5
(#133 – $D = 5$)	u	76.7	76.7	76.7	77.4
IPC 2006	s	13.3	13.3	4.7	4.7
(#150 – $D = 5$)	u	14.7	20.7	16.7	24.0
IPC 2011	s	0.0	1.7	0.0	0.0
(#120 – $D = 6$)	u	0.0	33.3	0.0	76.7
IPC 2014	s	0.0	0.0	0.0	0.0
(#100 – $D = 5$)	u	0.0	2.0	0.0	11.0
IPC 2018	s	0.0	0.0	0.0	0.0
(#200 – $D = 9$)	u	0.0	79.5	0.0	90.0
IPC 2023	s	11.0	11.0	5.0	5.0
(#100 – $D = 4$)	u	35.0	35.0	35.0	35.0

Table 6.6: Percentage coverage results for IPC domains from 1998 to 2023, comparing RESPLAN (RP) and RESPLAN2 (RP2) for $k = 1$ and $k = 2$. Coverage is shown for both solved (s) and unsolvable (u) instances.

of unsolvable problems in earlier domains, particularly IPC 1998 and IPC 2004, are significantly higher than in the later domains. This indicates that earlier benchmarks may have represented a more manageable challenge compared to the complexities introduced in recent years. For $k = 2$, the percentages remain low, with IPC 2023 reaching only 5.0% solved instances, and the best year being 1998 with 9.0%. The decrease in solved problems over the years, particularly in IPC 2011 and IPC 2014, correlates with a rise in unsolvable instances, reflecting a trend where, as problem complexity increases, the likelihood of finding solutions diminishes. High unsolvability rates reveal limited resilience, especially in IPC 2000, which reports 100% unsolvability for the problems analyzed. This

trend indicates that some problem sets, particularly those defined in the more recent IPCs, may be inherently more challenging or even beyond the current capabilities of existing solving techniques. The high unsolvability percentages in domains such as IPC 2018 (90.0%) and IPC 2023 (35.0%) further support the idea that the evolution of these problems has led to more complex, and in some cases unsolvable, scenarios.

Our analysis of the IPC domains demonstrates that resilience is not an intrinsic property that emerges spontaneously. Rather, it must be intentionally engineered through a careful analysis of domain structures and potential failure points. Resiliency is indeed a challenging feature to achieve and necessitates careful consideration during the design phase of a domain or physical system. Incorporating resiliency involves a thorough understanding of potential failure points and the implementation of strategies that allow the system to adapt and recover from disruptions. This process often requires significant investment in time and resources to ensure that the system is robust enough to handle various unforeseen challenges. From a practical standpoint, however, the benefits of achieving resiliency are substantial. A resilient system can enhance reliability, reduce downtime, and ultimately lead to better overall performance. As organizations increasingly rely on complex systems, the ability to maintain functionality despite adverse conditions becomes invaluable.

Chapter 7

Conclusions

In this thesis, we described the main research activities carried out during my three years of PhD and the experimental results obtained in the field of robust automated planning. The main contribution of this work is the development of a formal framework and efficient algorithms to address the Resilient Planning problem. The primary objective was to bridge the gap between classical planning, which assumes ideal execution, and real-world applications where actions can fail in unpredictable ways.

The first major milestones of this research were the formal modeling of the Resilient Planning problem and the introduction of RESPLAN, the first complete algorithmic solution for generating k -resilient plans. By building upon a recursive definition of state resilience, RESPLAN ensures goal achievement as long as the number of failures remains within a given bound k . The experimental analysis shows that RESPLAN significantly outperforms state-of-the-art FOND-based approaches in terms of scalability and coverage, particularly in domains where redundancy is an intrinsic feature.

To address the computational limitations of the initial approach, specifically

the high cost of proving non-resilience and the lack of knowledge reuse, we developed RESPLAN2. This enhanced version integrates two novel techniques:

- **Landmark-based Pruning:** by leveraging fact landmarks, the algorithm can identify non-resilient states at an early stage, avoiding the exhaustive exploration of unsolvable subproblems.
- **Adaptive Planning:** through the use of resilience certificates based on regressed state formulas, RESPLAN2 can reconnect the search to previously verified resilient suffixes, drastically reducing the number of required planner invocations.

The experimental results demonstrate that RESPLAN2 offers a substantial improvement in runtime efficiency and problem coverage, effectively handling higher resilience levels where the original algorithm and traditional FOND planners frequently encountered timeouts.

Finally, as a third contribution, we extended the framework to support the generation of Resilient Policies. Unlike a fixed plan, a policy provides a closed-loop strategy that guides the agent through every possible recovery branch without the need for real-time replanning. We also presented a formal Policy Validator to ensure the correctness of these strategies, proving that for every sequence of modeled failures, a guaranteed recovery path exists. The analysis of the “Worst-case Plan” length within these policies revealed that the cost of resilience can grow non-linearly with the failure budget K , providing valuable insights for the design of robust physical systems.

Despite the significant advancements, several directions for future work remain open. First, the pruning power of landmarks could be further enhanced by investigating disjunctive or action landmarks in the context of resilience. Second, we aim to bridge the gap with FOND planning formalisms to handle a broader

range of failure types. This extension would allow us to move beyond the current failure model, which assumes the state remains unchanged, and address scenarios involving non-deterministic failure outcomes.

Bibliography

- [1] Diego Aineto, Alessandro Gaudenzi, Alfonso Gerevini, Alberto Rovetta, Enrico Scala, and Ivan Serina. Action-failure resilient planning. In *ECAI 2023*, pages 44–51. IOS Press, 2023.
- [2] Arslan Ahmed Amin and Khalid Mahmood Hasan. A review of fault tolerant control systems: Advancements and applications. *Measurement and Control*, 52(9-10):1139–1154, 2019. doi: 10.1177/0020294019853361.
- [3] Mohannad Babli, Óscar Sapena, and Eva Onaindia. Plan commitment: Re-planning versus plan repair. *Engineering Applications of Artificial Intelligence*, 123:106275, 2023.
- [4] Christer Bäckström and Bernhard Nebel. Complexity results for SAS+ planning. *Computational Intelligence*, 11(4):625–655, 1995.
- [5] Tom Bylander. The computational complexity of propositional strips planning. *Artificial Intelligence*, 69(1):165–204, 1994. ISSN 0004-3702. doi: [https://doi.org/10.1016/0004-3702\(94\)90081-7](https://doi.org/10.1016/0004-3702(94)90081-7). URL <https://www.sciencedirect.com/science/article/pii/0004370294900817>.
- [6] Alessandro Cimatti, Marco Roveri, and Paolo Traverso. Automatic obdd-

- based generation of universal plans in non-deterministic domains. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 875–881. AAAI Press / The MIT Press, 1998.
- [7] Alessandro Cimatti, Marco Pistore, Marco Roveri, and Paolo Traverso. Weak, strong, and strong cyclic planning via symbolic model checking. *Artif. Intell.*, 147(1-2):35–84, 2003.
- [8] Carmel Domshlak. Fault tolerant planning: Complexity and compilation. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 23, pages 64–72, 2013.
- [9] Wolfgang Dvořák et al. Fond planning for all: Transforming fond into sat. In *Proceedings of the 33rd International Conference on Automated Planning and Scheduling (ICAPS)*, 2023.
- [10] Richard E. Fikes and Nils J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 1971.
- [11] Maria Fox, Alfonso Gerevini, Derek Long, and Ivan Serina. Plan stability: Replanning versus plan repair. In *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling (ICAPS-2006)*, pages 212–221. AAAI, 2006. URL <http://www.aaai.org/Library/ICAPS/2006/icaps06-022.php>.
- [12] Hector Geffner and Blai Bonet. *A Concise Introduction to Models and Methods for Automated Planning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2013.
- [13] Tomas Geffner and Hector Geffner. Compact policies for fully observable

- non-deterministic planning as SAT. In *ICAPS*, pages 88–96. AAAI Press, 2018.
- [14] Alfonso Gerevini and Ivan Serina. Efficient plan adaptation through re-planning windows and heuristic goals. *Fundamenta Informaticae*, 102(3-4): 287–323, 2010.
- [15] Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated Planning and Acting*. Cambridge University Press, 2016. ISBN 978-1-107-03727-4.
- [16] Frederik W. Heger and Sanjiv Singh. Robust robotic assembly through contingencies, plan repair and re-planning. In *2010 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3825–3830, 2010. doi: 10.1109/ROBOT.2010.5509274.
- [17] Malte Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.
- [18] Malte Helmert. Concise finite-domain representations for PDDL planning tasks. In *Artificial Intelligence*, volume 173, pages 503–535, 2009.
- [19] Malte Helmert and Carmel Domshlak. Landmarks, critical paths and abstractions: What’s the difference anyway? In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS)*, 2009.
- [20] J. Hoffmann, J. Porteous, and L. Sebastia. Ordered landmarks in planning. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS 2004)*, 2004. URL <https://www.ijcai.org/Proceedings/04/Papers/090.pdf>.
- [21] Richard Howey, Derek Long, and Maria Fox. Val: Automatic plan validation, continuous effects and mixed initiative planning using pddl. In *16th IEEE*

- International Conference on Tools with Artificial Intelligence*, pages 294–301. IEEE, 2004.
- [22] Rune M Jensen, Manuela M Veloso, and Randal E Bryant. Fault tolerant planning: Toward probabilistic uncertainty models in symbolic non-deterministic planning. In *ICAPS*, pages 335–344, 2004.
- [23] Ted G. Lewis. The mathematics of catastrophe. *AppliedMath*, 2(3):480–500, 2022. doi: 10.3390/appliedmath2030028.
- [24] Francisco Maciel et al. PlanSys2: A planning system for ROS2. In *2021 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 1–6, 2021. doi: 10.1109/ICARSC52212.2021.9440954.
- [25] Robert Mattmüller, Manuela Ortlieb, Malte Helmert, and Pascal Bercher. Pattern database heuristics for fully observable nondeterministic planning. In *ICAPS*, pages 105–112. AAAI, 2010.
- [26] Drew McDermott, Malik Ghallab, Adele E. Howe, Craig A. Knoblock, Ashwin Ram, Manuela M. Veloso, Daniel S. Weld, and David E. Wilkins. Pddl-the planning domain definition language. 1998. URL <https://api.semanticscholar.org/CorpusID:59656859>.
- [27] Leonardo Messa and André Grahl Pereira. Foundations of native heuristic search for fond planning. In *Proceedings of the 33rd International Conference on Automated Planning and Scheduling (ICAPS)*, pages 310–318, 2023.
- [28] Andrea Micheli, Arthur Bit-Monnot, Gabriele Röger, Enrico Scala, Alessandro Valentini, Luca Framba, Alberto Rovetta, Alessandro Trapasso, Luigi Bonassi, Alfonso Emilio Gerevini, Luca Iocchi, Felix Ingrand, Uwe Köckemann, Fabio Patrizi, Alessandro Saetti, Ivan Se-

- rina, and Sebastian Stock. Unified planning: Modeling, manipulating and solving ai planning problems in python. *SoftwareX*, 29: 102012, 2025. ISSN 2352-7110. doi: <https://doi.org/10.1016/j.softx.2024.102012>. URL <https://www.sciencedirect.com/science/article/pii/S2352711024003820>.
- [29] Christian J. Muise, Sheila A. McIlraith, and J. Christopher Beck. Improved non-deterministic planning by exploiting state relevance. In *ICAPS*. AAAI, 2012.
- [30] Ramon Fraga Pereira, André Grahl Pereira, Frederico Messa, and Giuseppe De Giacomo. Iterative depth-first search for FOND planning. In *ICAPS*, pages 90–99. AAAI Press, 2022.
- [31] S. Richter, M. Helmert, and M. Westphal. Landmarks revisited. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS 2010)*, 2010. URL <https://cdn.aaai.org/AAAI/2010/ICAPS2010/ICAPS20100149.pdf>.
- [32] Alberto Rovetta, Diego Aineto, Alfonso Gerevini, Enrico Scala, and Ivan Serina. *Improving Resilient Planning Through Landmarks and Regressed State Formulas*. 10 2025. ISBN 9781643686318. doi: 10.3233/FAIA251369.
- [33] X. Wu, D. Zhang, and X. Zheng. Resilience planning model of flexible resources collaborative under typhoon disaster. In *2024 IEEE International Conference on Energy Internet (ICEI)*, pages 373–379, 2024. doi: 10.1109/ICEI63732.2024.10917096.
- [34] Sung Wook Yoon, Alan Fern, and Robert Givan. Ff-replan: A baseline for probabilistic planning. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling, (ICAPS-2007)*. AAAI, 2007.

-
- [35] Lin Zhu and Robert Givan. Landmark extraction via planning graph propagation. *ICAPS Doctoral Consortium*, pages 156–160, 2003.