Journal Title
XX(X):1–18

DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

A Temporal Planning Approach for Intelligent Flood Response

Fazlul Hasan Siddiqui[1], Md. Monjurul Islam[1,2], and Sabah Binte Noor[1]

Abstract

Effective response to multiple, simultaneously flooded areas requires coordinating appropriate actions in the correct temporal order, under severe resource constraints. Automated planning provides a foundation for addressing this challenge by generating time-aware schedules, given a formal description of available resources, constraints, and goals. This work presents an intelligent flood-response framework that exploits temporal planning and models the complete operational life cycle of flood response. The framework incorporates priority-driven triage, route accessibility and travel costs, resource allocation, and supply management, while also supporting mid-execution re-planning in response to unexpected environmental changes. The framework is formulated both in the Action Notation Modeling Language (ANML) and the Planning Domain Definition Language (PDDL) 2.1, facilitating compatibility with a wider range of temporal planners. Experimental results establish the feasibility and scalability of the proposed framework, showing that flood response scenarios can be effectively modeled and solved using temporal planning, while providing guidance on planner selection.

Keywords

Flood Response, Planning, Reasoning, AI and Society

Introduction

An effective response to floods requires rapid decision-making and coordinated actions (Kapucu and Garayev 2011). Traditional flood management combines structural and non-structural measures. Structural measures include embankments and improved drainage, while non-structural measures include early warning systems (Meyer et al. 2012). However, traditional approaches are increasingly considered inadequate to deal with complex flood events, especially when multiple locations are affected at the same time and response resources have to be allocated under severe time constraints (Rutherford et al. 2024). Automated planning systems offer a viable solution to this problem.

Automated planning systems use heuristic-based search to reason over a formal description of available resources and operational constraints (Ghallab et al. 2004). In this work, we formulate flood response as a temporal planning problem to provide structured coordination and scheduling of response actions under resource constraints, along with dynamic replanning to adapt to environmental changes during execution of the scheduled actions.

Earlier works in disaster management focused primarily on preparedness and prevention, whereas recent studies emphasize response, recovery, and resilience. However, absolute protection is neither achievable nor sustainable due to inherent uncertainties and high costs (Schanze 2006). Therefore, strategic flood risk management principles recommend a portfolio of diverse responses rather than relying on a single measure (Sayers et al. 2013). A comparative study of the severe 1998 and 2020 Yangtze River floods by Jia et al. (2022) showed that the results were significantly improved in 2020 due to the advanced emergency response and risk management capabilities. This study indicates that effective response measures are critical for mitigating flood impacts and strengthening community resilience.

Several studies formalize specific flood response operations as computational problems. Chang et al. (2007) use stochastic programming models to optimize rescue depot locations and supply allocation under uncertainty. They employed a sample average approximation scheme to solve their models. Simonovic and Ahmad (2005) developed a simulation model that captures human behavior during flood emergency evacuation. Their model incorporates key variables such as the number of families at risk, evacuated populations, warning systems, and route inundation. At a broader scale, Ye et al. (2021) propose an interdisciplinary, AI-driven framework for urban flood resilience that integrates urban planning, landscape architecture, and computer science, addressing the persistent gap between research and practice in achieving flood resilience. Whereas Ambily et al. (2024) present a spatial planning framework centered on blue-green infrastructure for prioritizing key infrastructure dimensions.

Other researchers focus specifically on evacuation logistics during extreme events. Yin et al. (2024) examine storm flood

[1]Department of Computer Science and Engineering, Dhaka University of Engineering & Technology, Gazipur, 1707, Bangladesh
[2]Department of Computer Science and Engineering, Bangladesh Army University of Engineering & Technology, Natore, 6431, Bangladesh

Corresponding author:
Sabah Binte Noor, Department of Computer Science and Engineering, Dhaka University of Engineering & Technology, Gazipur, 1707, Bangladesh.
Email: sabah@duet.ac.bd

evacuations in large coastal cities, with particular attention to elderly populations. Their optimization of shelter placement and routing evacuees to nearby rather than distant facilities improved overall evacuation efficiency. Meanwhile, Wang et al. (2024) study pedestrian evacuation during dam-break floods. They utilize a fuzzy VIKOR method to evaluate potential shelters and a dual-objective model to balance evacuation time, cost, and adaptability under evolving post-disaster conditions. While these works address important aspects of flood response, none employ automated planning and scheduling to generate time-aware, coordinated response plans under resource constraints.

The most relevant work in this domain is the RAPID framework by Islam et al. (2025), which introduces a disaster response domain using the Planning Domain Definition Language (PDDL) (Ghallab et al. 1998) with numeric planners to generate disaster response plans. However, RAPID relies on classical PDDL with numeric fluents, and therefore, it cannot model action scheduling or the concurrent overlapping of operations. This study addresses that limitation by introducing a temporal planning approach for temporally coherent flood response.

In this work, we formalize flood response as a coordination problem to optimize critical operational decisions, including the allocation of teams, resources, and vehicles, as well as the timing and sequencing of concurrent actions. We present a temporal planning framework, modeled in two formal languages, the Action Notation Modeling Language (ANML) (Smith et al. 2008) and PDDL 2.1 (Fox and Long 2003), covering the full operational cycle of flood response. The proposed framework incorporates real-world constraints like priority-based triage ordering, route accessibility, vehicle capacity, and scheduling of concurrent operations. Additionally, the framework includes a symbolic milestone system that captures the multi-trip nature of evacuation and supply delivery without the computational overhead of continuous numeric reasoning. Notably, the framework supports mid-execution replanning to adapt to the dynamic changes in the environment. As new data is revealed from the field, it allows us to update the instance to the current observable state and trigger replanning for an adapted schedule. We also provide practical planner selection guidance based on scenario complexity and quality requirements.

Illustrative Example

A representative flood response scenario used to demonstrate the proposed domain is shown in Figure 1. The scenario consists of two safe areas, `safeA` and `safeB`, and two affected areas, `zoneA` and `zoneB`. The location `safeA` is the main deployment base that has a transit vehicle (`bus`) with an evacuation capacity of 20 persons per trip, a freight vehicle (truck) used for resource transportation, a rescue team of 5 members, a medical support team of 5 members, and 100 units of relief goods (food).

Affected area `zoneA` requires rescue operations, medical support, and 100 units of food, whereas `zoneB` requires rescue operations and evacuation of 40 people. Travel time between locations depends on the vehicle. For instance, a bus takes 18 time units to move from `safeA` to `zoneA`, while a truck takes 25, consistent with real-world cases where vehicle type determines travel time.

To illustrate how temporal planning operates on such a scenario, Figure 2 shows a valid action schedule, produced by an automated temporal planner. The plan consists of 25 actions with a makespan of 275 time units. This plan demonstrates how temporal planning coordinates overlapping operations. Specifically, at time 0, three actions execute concurrently: loading relief goods into the truck (`LoadResource`), while the medical support and rescue teams simultaneously board the bus (`BoardTeam`). After that, the bus departs at time 5 and arrives at `zoneA` at time 23, and both teams disembark from the bus and immediately begin rescue operations (`RescueAffectedPeople`) and medical support (`ProvideMedicalSupport`) in parallel at time 28.

In addition, the truck departs at time 50 and arrives at `zoneA` at time 75, and relief goods are then delivered. Meanwhile, the rescue team is transported and is redeployed to `zoneB` via bus, where evacuation and rescue operations are carried out across two evacuation sorties (`Evacuate` at times 77 and 129). Each evacuation sortie is followed by a return trip to `safeA` to disembark evacuees before redeployment. Applying the whole plan efficiently resolves all demands for evacuation, rescue, medical support, and resource delivery across the impacted zones, and several actions concurrently take place, similar to a real-world scheduling scenario.

Background

In this section, we introduce basic notions of automated planning and modeling languages used in this work. We first define the primary automated planning paradigms, including classical, numeric, and temporal planning, and then present an overview of the two modeling languages, PDDL and ANML, used to model the proposed domain.

Automated Planning

Automated planning generates sequences of actions (i.e., plans) which transform an initial state into a desired goal state (Ghallab et al. 2004). Depending on the complexity of the environment of the problem, planning models need to reason about logical conditions, quantitative resources, and temporal constraints. These requirements give rise to three progressively expressive paradigms: classical planning, numeric planning, and temporal planning.

Classical planning (Definition 1) provides the fundamental structural syntax for automated planning. It operates within a deterministic, fully observable, and static environment, meaning the agent has complete knowledge of the system state and the deterministic outcomes of its actions.

Definition 1. *A **classical planning task** is represented as a tuple $\Pi = \langle F, A, \gamma, I, G \rangle$, where:*

- *F is a finite set of propositional facts,*
- *A is a finite set of actions,*
- *$\gamma : A \mapsto \mathbb{Q}_0^+$ is a function assigning a non-negative rational cost to each action,*

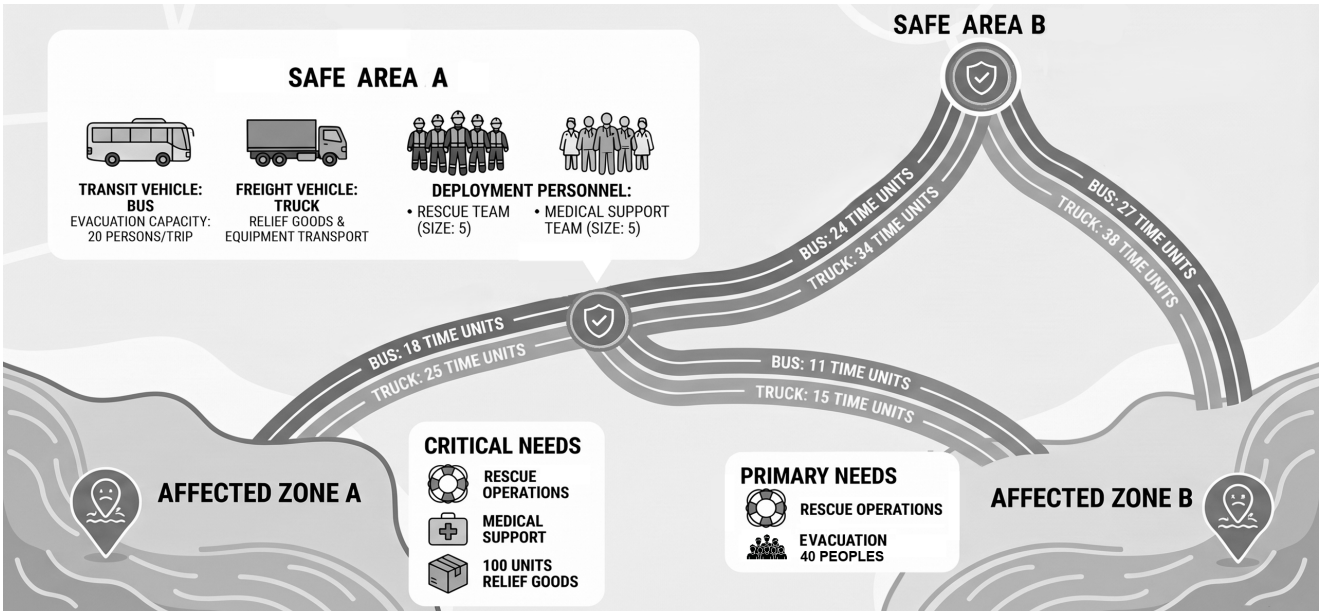

Figure 1. An example scenario with two safe areas (`safeA` and `safeB`) and two affected zones (`zoneA` and `zoneB`). `zoneA` requires rescue operations, medical support, and 100 units of relief goods, while `zoneB` requires rescue operations and evacuation of 40 people. Vehicle-dependent travel times range from 11 to 38 time units.

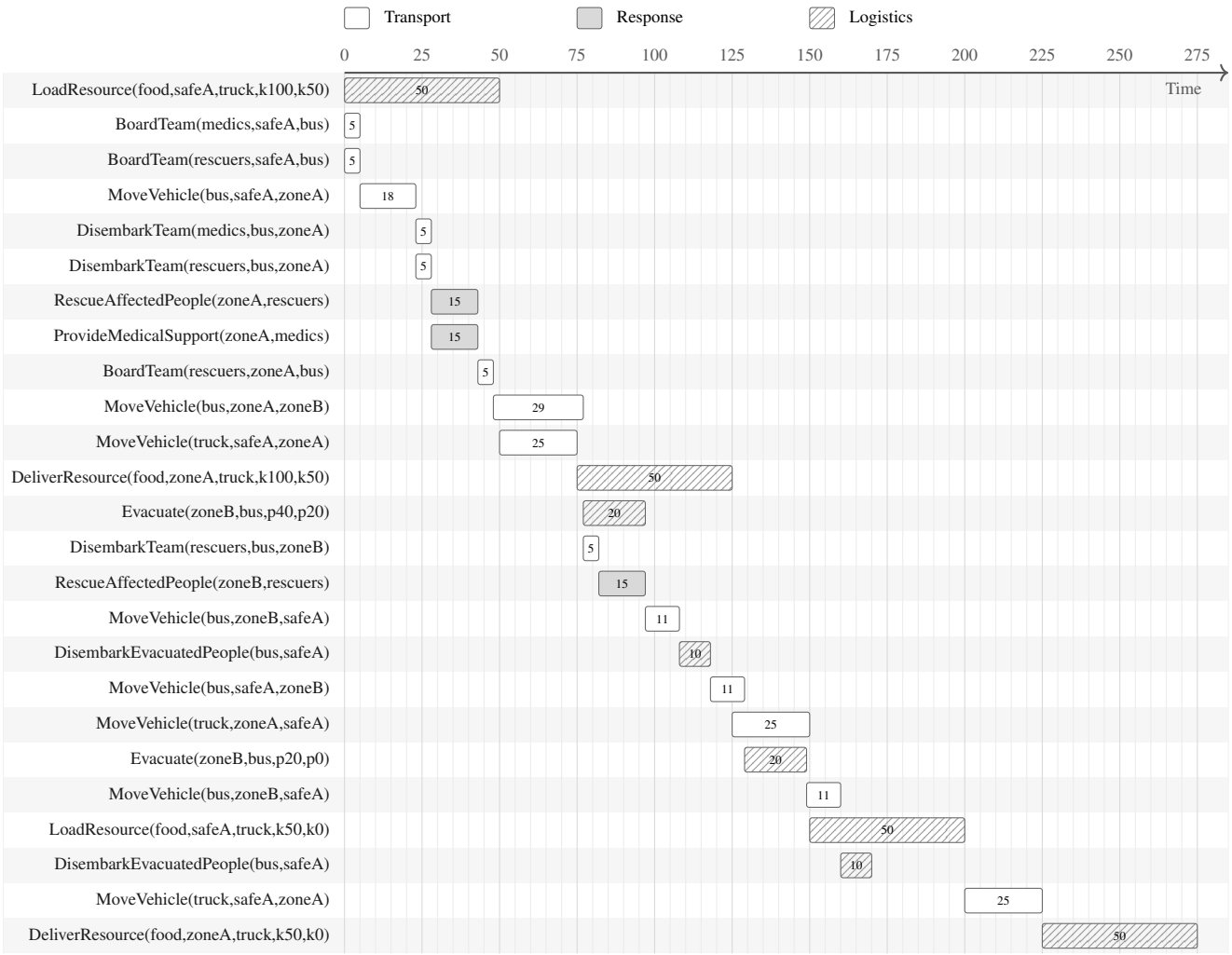

Figure 2. A temporal plan for the example flood response scenario in Figure 1. Each bar represents a durative action, where the horizontal placement indicates the start time and the width with the inscribed number indicates the execution duration in time units. Actions aligned vertically execute concurrently.

- $I \subseteq F$ *represents the initial state, and*
- $G \subseteq F$ *specifies the goal conditions.*

Each action $a \in A$ *is a tuple* $a = \langle pre(a), add(a), del(a) \rangle$*, where* $pre(a), add(a), del(a) \subseteq F$ *denote the action's precondition, add list, and delete list, respectively. The add and delete lists collectively constitute the action effects.*

A solution to a classical planning task is a sequence of actions that transforms the initial state into a state satisfying the goal conditions. An action is applicable only when its preconditions hold in the current state, after which its effects modify the state by adding and removing facts. Definition 2 formally defines a **plan**.

Definition 2. *A* ***plan*** π *for a classical planning task* Π *is an ordered sequence of actions* $\langle a_1, a_2, \ldots, a_n \rangle$*. The plan* π *is valid if and only if* $pre(a_1) \subseteq I$ *and the sequential application of actions* $a_1, a_2, \ldots, a_n$ *successfully yields a state satisfying the goal conditions* G*.*

Classical planning cannot represent quantitative resources such as fuel, supply quantities, or vehicle capacities because states are purely propositional. Many real-world problems require reasoning about quantities that change during execution. Numeric planning directly extends classical planning by introducing numeric state variables and arithmetic expressions into the planning model (Fox and Long 2003).

Definition 3. *A* ***numeric planning task*** *is defined as a tuple* $\Pi_n = \langle F, X, A, \gamma, I, G \rangle$*, where* F*,* A*, and* γ *retain their definitions from the classical planning paradigm (Definition 1), and* X *is a finite set of numeric variables bounded over the rational numbers* $\mathbb{Q}$*,* $I \subseteq F \cup X$ *is the initial state, and* $G \subseteq F \cup X$ *specifies the goal conditions.*

Actions of a numeric planning task may modify both propositional facts and numeric variables through arithmetic updates. Despite this added expressiveness, numeric planning still assumes that actions occur instantaneously. Real-world operations involve activities with duration, deadlines, and concurrency requirements that numeric planning tasks fail to encode. Temporal planning addresses these limitations by explicitly incorporating time into the planning process (Fox and Long 2003).

Definition 4. *A* ***temporal planning task*** *is represented as a tuple* $\Pi_t = \langle F, X, A^d, \gamma, I, G \rangle$*, where* F*,* X*,* γ*,* I*, and* G *are defined as in a numeric planning task (Definition 3), and* A^d *is a finite set of* durative actions*. Each action* $a \in A^d$ *is a tuple as given in* (1)*.*

$$a = \langle\, pre^s(a),\ pre^o(a),\ pre^e(a),\ eff^s(a),\ eff^e(a),\ dur(a) \,\rangle \tag{1}$$

where $pre^s(a)$*,* $pre^o(a)$*,* $pre^e(a) \subseteq F \cup X$ *are the* at-start*,* over-all*, and* at-end *preconditions respectively,* $eff^s(a)$*,* $eff^e(a) \subseteq F \cup X$ *are the* at-start *and* at-end *effects, and* $dur(a) \in \mathbb{R}^+$ *is the action duration.*

Temporal planning extends numeric planning by modeling actions as durative activities whose conditions and effects may occur at different points during execution. At-start preconditions must hold at the moment execution begins, over-all preconditions must remain true throughout the open interval of the action's duration, and at-end preconditions must hold upon completion. Effects are applied at their respective time points, modifying the state by adding and removing facts. This enables planners to reason about overlapping actions, synchronization constraints, and time-dependent resource usage. Additionally, Timed Initial Literals (TILs) allow facts to change independently at predefined time points.

The solution to a temporal planning task is a timed schedule of actions, i.e., a temporal plan (Definition 5).

Definition 5. *A* ***temporal plan*** π_t *for a temporal planning task* Π_t *is a timed schedule of durative actions represented as a tuple* $\langle (t_1 : a_1), (t_2 : a_2), ..., (t_n : a_n) \rangle$*, where each* $t_i \in \mathbb{R}_0^+$ *is a time point of the start of some durative action* $a_i \in A_d$*. The schedule* π_t *is valid iff all action preconditions are satisfied at their required time points, and the goal conditions are achieved upon completion of the schedule.*

Plan quality in temporal planning is most commonly measured by *makespan* (Definition 6), the total time required to execute a schedule.

Definition 6. *The* ***makespan*** *of a temporal plan* $\pi_t = \langle (t_1{:}a_1), (t_2{:}a_2), \ldots, (t_n{:}a_n) \rangle$ *is the elapsed time between the start of its earliest action and the completion of its latest action, as specified in* (2)*. A smaller makespan reflects a more temporally compact schedule with greater concurrency among actions.*

$$\text{makespan}(\pi_t) \;=\; \max_{1 \le i \le n} \big(t_i + \text{dur}(a_i)\big) \;-\; \min_{1 \le i \le n} (t_i) \tag{2}$$

Flood response operations require synchronized activities with durations involving constrained time and resources, so temporal planning is the natural fit for this work.

Planning Modeling Languages

Planning modeling languages specify planning domains and problem instances in terms that automated planners can process: defining actions, state variables, constraints, temporal properties, initial states, and goal conditions. Early languages worked with propositional representations and instantaneous actions; modern ones also handle quantitative reasoning, temporal constraints, durative actions, and concurrency. This work uses two temporal planning languages, PDDL and ANML, and the concepts from each that are relevant here are the following.

The Planning Domain Definition Language The Planning Domain Definition Language (PDDL) is a standardized formalism for modeling automated planning domains and problems (Ghallab et al. 1998). It separates reusable domains containing typed predicates, functions, and parameterised action schemas from instance-specific information such as the objects, the initial state, and the goal. PDDL 2.1 extends the classical propositional language with *numeric fluents* and *durative actions* (Fox and Long 2003), enabling quantitative and temporal reasoning.

A durative action in PDDL 2.1 specifies a duration together with temporally annotated conditions and effects. Three qualifiers locate these in time: `at start` conditions and effects apply at the instant execution begins, `over all` conditions must hold throughout the open interval of

the action, and `at end` conditions and effects apply on completion. The Temporal Fast Downward (TFD) planner used in our evaluation consumes PDDL 2.1.

The Action Notation Modeling Language The Action Notation Modeling Language (ANML) is a high-level formalism for modeling planning domains and problems (Smith et al. 2008). Where PDDL is primarily propositional, ANML adopts a variable/value representation in which every variable is an implicit function of time, so that state transitions over an action's execution are expressed directly on a timeline. Types follow a single-inheritance hierarchy and may carry member variables and constants.

Conditions and effects are not separated by keywords but distinguished by their operators: relational operators (e.g. ==) express conditions, while assignment and transition operators (e.g. `:=` and `:->`) express effects. Each condition and effect is scoped by a temporal qualifier relative to the action's `start` and `end` landmarks, and `all` denotes the whole execution window. The Flexible Action and Planning Environment (FAPE) used in our evaluation consumes ANML.

Proposed Temporal Planning Framework

The proposed framework for flood-response operations models disaster-response activities as a temporal planning task involving heterogeneous response teams, transportation assets, operational priorities, and resource-constrained logistics under time-dependent execution. Its objective is to generate temporally consistent schedules that coordinate rescue operations, medical assistance, civilian evacuation, and humanitarian supply distribution across geographically dispersed flood-affected areas. The framework comprises three main components: a *domain definition* that specifies the state representation, constraints, and durative actions of the flood-response environment; an *instance formulation* that defines a specific disaster scenario through its initial and goal states; and *plan generation*, in which an automated temporal planner generates a valid execution schedule. The remainder of this section describes each component.

We use a compact illustrative scenario, presented in Figure 3, as a running example throughout this section. A single safe location S serves as a staging hub containing a transit vehicle T, a freight vehicle F, a rescue team R, a medical team M, and a stock of relief goods (e.g., `food=k300`). From this safe hub, operations are carried out over two affected zones: Zone A (priority 1) and Zone B (priority 2). Each zone has associated evacuation and relief requirements. A strict triage constraint is imposed through the `prior` relation: each response operation in Zone A must be completed before the same operation in Zone B can begin. Vehicle-specific travel times between the safe hub and affected zones are encoded as edge labels. This compact instance captures all core structural elements of the domain: shared vehicles, competing zones, temporal routing constraints, and resource-dependent service execution.

Task Formulation

We present the proposed model using a propositional temporal-planning formulation (Definition 7), following the general formulation introduced in Definition 4. We first characterize the state space, then the action model, and finally the validity conditions a solution plan must satisfy. The flood-response planning task is stated as follows.

Definition 7. *A **flood-response planning task** is a temporal planning problem $\Pi_f = \langle F_f, X_f, A_f^d, \gamma_f, I_f, G_f \rangle$ defined over a flood-response domain, where:*

- *F_f is the finite set of grounded propositional facts over the predicate set $\mathcal{P}$, specified in (3).*

$$\mathcal{P} = \left\{ \begin{array}{l} \texttt{prior}, \texttt{vehicle\_at}, \texttt{team\_at}, \\ \texttt{team\_in\_vehicle}, \texttt{evac\_level}, \\ \texttt{needs\_rescue}, \texttt{resource\_needed}, \\ \texttt{needs\_medical\_support}, \texttt{evacuating}, \\ \texttt{resource\_available}, \texttt{cargo\_loaded}, \\ \texttt{cargo\_full}, \texttt{route\_accessible}, \\ \texttt{valid\_evac\_step}, \texttt{valid\_supply\_step} \end{array} \right\} \tag{3}$$

- *X_f is the finite set of ground numeric fluents (all* rigid*, i.e. never modified by an action effect) over the set $\mathcal{X} = \{\texttt{travel\_time}, \texttt{team\_size}, \texttt{person\_count\_diff}, \texttt{package\_count\_diff}\}$.*

- *A_f^d is the finite set of ground durative actions over the set of nine action schemas $\mathcal{A}$, specified in (4), instantiated over the typed objects of a given problem instance.*

$$\mathcal{A} = \left\{ \begin{array}{l} \textsc{BoardTeam}, \textsc{DisembarkTeam}, \\ \textsc{MoveVehicle}, \textsc{Evacuate}, \\ \textsc{DisembarkEvacuatedPeople}, \\ \textsc{RescueAffectedPeople}, \\ \textsc{LoadResource}, \textsc{DeliverResource}, \\ \textsc{ProvideMedicalSupport} \end{array} \right\} \tag{4}$$

- *γ_f is the action cost function, where $\gamma_f = 1$ for all $a \in A_f^d$.*

- *$I_f \subseteq F_f \cup X_f$ is the initial state, grounded over $\mathcal{P} \cup \mathcal{X}$ to describe the initial configuration of vehicles, teams, routes, operational requirements, and milestone levels for a given flood scenario.*

- *$G_f \subseteq F_f$ is the goal condition grounded over $\mathcal{P}$, as formalized in (5).*

$$G_f = \left\{ \begin{array}{l} \neg\texttt{needs\_rescue}(l), \neg\texttt{evacuating}(v), \\ \neg\texttt{needs\_medical\_support}(l), \\ \texttt{evac\_level}(l, \texttt{p0}), \\ \texttt{resource\_needed}(l, r, \texttt{k0}) \end{array} \right\} \tag{5}$$

for all $l \in \mathcal{L}_A$, $r \in \mathcal{R}$, and $v \in \mathcal{V}_T$, where $\mathcal{L}_A$ denotes the set of affected locations, $\mathcal{R}$ is the set of resource types, $\mathcal{V}_T$ is the set of transit vehicles, and `p0` and `k0` are the base milestone values denoting full evacuation and full resource satisfaction.

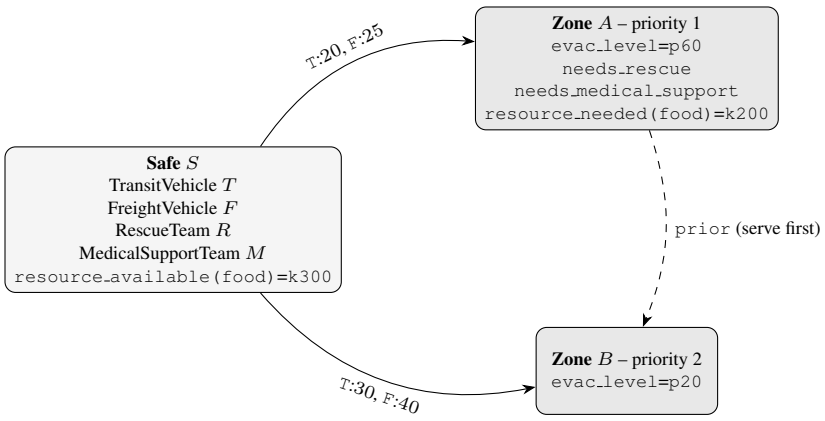

Figure 3. A compact flood-response instance used as a running example. A single safe location S stages a transit vehicle T, a freight vehicle F, a rescue team R, and a medical support team M, together with a stock of relief goods (`food=k300`). Zone A outranks Zone B via `prior`. Edge labels give vehicle-specific travel times.

Modeling Decisions

The following modeling decisions are intended to balance representational expressiveness with planning tractability and motivate the predicates and action schemas introduced in the following subsections.

Symbolic milestones instead of numeric fluents. Evacuation and supply progress are represented using ordered symbolic milestones rather than continuous numeric fluents. This design avoids large numeric search spaces while preserving multi-trip operational behavior and capacity-aware progress tracking.

Priority-constrained execution. Area-specific priorities are encoded directly within the preconditions of rescue, medical support, evacuation, and resource-delivery actions. Consequently, every generated plan automatically respects operational priorities without requiring additional validation or post-processing.

Temporal resource locking. Vehicles and teams are treated as temporally exclusive resources. During transportation, they are temporarily unavailable for other actions, thereby preventing conflicting concurrent assignments.

Native concurrency. The temporal-planning formalism naturally permits concurrent execution of independent rescue, medical, transportation, and logistics operations when resource and ordering constraints are satisfied.

World Representation

This subsection defines the representational structure of the planning domain, including the type hierarchy, the state predicates, and the symbolic milestone system used to model operational progress.

Type System The proposed domain uses a typed object hierarchy to organize operational entities involved in planning. The domain is defined over four root types: `Location`, `Vehicle`, `Team`, and `Resource`. These root types are further specialized, as presented in (6), to capture the operational structure of flood-response activities as follows:

$$\begin{aligned} \texttt{Location} &\supseteq \{\texttt{AffectedLocation}, \texttt{SafeLocation}\} \\ \texttt{Vehicle} &\supseteq \{\texttt{TransitVehicle}, \texttt{FreightVehicle}\} \\ \texttt{Team} &\supseteq \{\texttt{RescueTeam}, \texttt{MedicalSupportTeam}\} \end{aligned} \tag{6}$$

- **Safe areas and affected zones:** The `Location` type represents all areas within the scenario and is specialized into two distinct subtypes: `AffectedLocation` and `SafeLocation`.
- **Transportation Assets:** The `Vehicle` type models transportation assets and is specialized into two subtypes: `TransitVehicle` for personnel and evacuee transport and `FreightVehicle` for relief cargo delivery.
- **Response Teams:** The `Team` type represents deployable response units and is partitioned into two subtypes: `RescueTeam` and `MedicalSupportTeam`.
- **Consumable Resources:** `Resource` denotes consumable humanitarian supplies such as food, water, and medical kits. Its dynamic quantities are represented through symbolic milestones and state predicates across locations.

A distinguished constant `NULL` represents temporary unassignment of entities during transitional operations such as movement, boarding, loading, and unloading.

State Representation The predicates collectively define the operational world state of the flood-response environment. They are organized into spatial configuration, vehicle status, location requirements, operational progress, and rigid constants.

- **Spatial configuration:** The spatial information of vehicles and teams is captured by `vehicle_at(v,l)`, `team_at(t,l)`, and `team_in_vehicle(t,v)`, denoting vehicle locations, team deployment, and team boarding status, respectively.
- **Vehicle status:** Vehicle operations are captured by `evacuating(v)`, `route_accessible(v,l1,l2)`, `cargo_full(v)`, and `cargo_loaded(v,r)`, indicating evacuation activity, cargo status, and route feasibility.

- **Location requirements and priority:** Operational requirements are expressed using `needs_rescue(l)` and `needs_medical_support(l)`. A priority relation `prior(l,pl)` enforces that location `pl` must be served before `l`, restricting concurrent servicing of lower-priority areas.

- **Operational progress and milestones:** Evacuation and supply delivery progress are modeled using ordered symbolic milestones rather than numeric fluents. For instance, two ordered domains can be defined: `PersonCount` with $\texttt{p60} \succ \texttt{p40} \succ \texttt{p20} \succ \texttt{p0}$, and `PackageCount` with $\texttt{k200} \succ \texttt{k100} \succ \texttt{k0}$, where `p0` and `k0` denote goal satisfaction. Three predicates operate over `evac_level(l,p)` (remaining evacuees), `resource_needed(l,r,k)` (remaining demand), and `resource_available(l,r,k)` (available stock at safe locations). Each relevant action advances the corresponding milestone toward its terminal.

- **Constants:** A set of numeric fluents governs action durations and remains fixed for each instance. Numeric fluent `team_size(t)` defines team's boarding and disembarkation durations, `travel_time(v,l1,l2)` defines vehicle movement durations, while `person_count_diff(p_1,p_2)` and `package_count_diff(k_1,k_2)` scale evacuation and delivery durations according to corresponding workload. Vehicle's capacity constraints are enforced via `valid_evac_step(v,p_1,p_2)` and `valid_supply_step(v,k_1,k_2)`, which restrict allowed milestone transitions.

Action Schemas

The proposed domain is defined over nine durative actions. A set of preconditions and effects specifies each action, annotated using temporal qualifiers that indicate when they apply: *at start* (execution onset), *over all* (maintained throughout execution), and *at end* (completion). The complete action schemas are presented in Tables 1–3.

Transport Actions The transport actions, BOARDTEAM, DISEMBARKTEAM, and MOVEVEHICLE (Table 1), govern the movement of teams and vehicles across the locations. BOARDTEAM and DISEMBARKTEAM are symmetric operations that transfer a team into and out of a vehicle while maintaining the vehicle's location as an invariant throughout execution. Both actions scale in duration with `team_size`.

MOVEVEHICLE models physical relocation between locations and requires route accessibility between origin and destination. The vehicle remains in transit for the entire duration, and travel time is parameterized by the specific vehicle and route. Multi-hop travel is achieved by consecutive move actions through intermediate locations.

Response Actions Response actions are RESCUEAFFECTEDPEOPLE and PROVIDEMEDICALSUPPORT (Table 2). These actions are executed on-site by dedicated teams and require that the corresponding team remains present throughout execution as an *over all* invariant. Both actions are governed by a shared triage constraint expressed through the `prior` predicate, which prevents service at a location unless all higher-priority locations have already been fully serviced for the same task type. For instance, Zone A must be cleared before Zone B in Figure 3. Each action has a fixed duration of constant time units and, upon completion, removes the corresponding demand predicate at the location. Because rescue and medical teams are distinct resources, these actions may be executed concurrently at the same location when available.

Logistics Actions Logistics actions (Table 3) implement evacuation and supply operations across the same spatial network under shared priority constraints. EVACUATE moves civilians in discrete capacity-bounded steps governed by `valid_evac_step`, updating evacuation milestones and marking vehicles as actively evacuating. To enable concurrency, key milestone updates occur at action start. DISEMBARKEVACUATEDPEOPLE drops off evacuees at safe locations and releases vehicles for reuse.

Supply operations are handled by LOADRESOURCE and DELIVERRESOURCE, which shuttle goods between depots and affected zones while updating resource demand milestones. Both evacuation and supply actions use discretized progress variables rather than arithmetic quantities, ensuring compatibility with symbolic planning representations and enabling repeated execution until completion.

Dependency Among Action Schemas

The actions form a structured causal network in which the effects of one action establish the preconditions of others. This dependency structure is summarized in Figure 4. Solid arrows denote causal dependencies, where the source action produces a fact required by the target action. Nonetheless, the target action need not depend on the source action in every execution, since the required fact may already hold in the initial configuration. Dashed arrows represent priority constraints that prevent execution in lower-priority zones until higher-priority zones are served by the same action. Three interacting operational cycles emerge from this structure.

- **Deployment Cycle**: moves teams from the safe hub into affected zones via vehicle boarding, movement, and disembarkation, enabling on-site response.

- **Evacuation Cycle**: alternates between loading civilians, transporting them to safety, and disembarking them at secure locations.

- **Supply Cycle**: manages the flow of relief goods from depots to affected zones through repeated loading and delivery operations.

The MOVEVEHICLE action participates in all three cycles and therefore constitutes the primary shared scheduling resource in the domain. Priority constraints are enforced across response and logistics actions, ensuring strict triage ordering across zones.

Instance Formulation

The domain specification defines the available object types, predicates, and action schemas, but it does not determine which concrete entities participate in a particular disaster-response scenario. This information is provided by a problem

Table 1. Transport action schemas. Parameter shorthands: t denotes a team; l, $from$, and to denote locations; v_T denotes a transit vehicle; and T denotes a vehicle.

Action	Preconditions		Effects	
[Duration]	*ann.*	*predicate(s)*	*ann.*	*predicate(s)*
BOARDTEAM (t, l, v_T)	*Start*	team_at(t, l)	*Start*	¬team_at(t, l)
[team_size(t)]	*Over-all*	vehicle_at(v_T, l), ¬evacuating(v_T)	*End*	team_in_vehicle(t, v_T)
DISEMBARKTEAM (t, v_T, l)	*Start*	team_in_vehicle(t, v_T)	*Start*	¬team_in_vehicle(t, v_T)
[team_size(t)]	*Over-all*	vehicle_at(v_T, l)	*End*	team_at(t, l)
MOVEVEHICLE $(v, from, to)$	*Start*	vehicle_at$(v, from)$	*Start*	¬vehicle_at$(v, from)$
[travel_time$(v, from, to)$]	*Over-all*	route_accessible$(v, from, to)$	*End*	vehicle_at(v, to)

Table 2. Response action schemas. Parameter shorthands: l_A denotes an affected location; pl denotes a higher-priority affected location; and C denotes a constant value.

Action	Preconditions		Effects	
	ann.	*predicate(s)*	*ann.*	*predicate(s)*
RESCUEAFFECTEDPEOPLE $(l_A, rescuers, pl)$	*Start*	prior(l_A, pl), ¬needs_rescue(pl), needs_rescue(l_A)	–	–
	Over-all	team_at$(rescuers, l_A)$	*End*	¬needs_rescue(l_A)
PROVIDEMEDICALSUPPORT $(l_A, medics, pl)$	*Start*	prior(l_A, pl), ¬needs_medical_support(pl), needs_medical_support(l_A)	–	–
	Over-all	team_at$(medics, l_A)$	*End*	¬needs_medical_support(l_A)

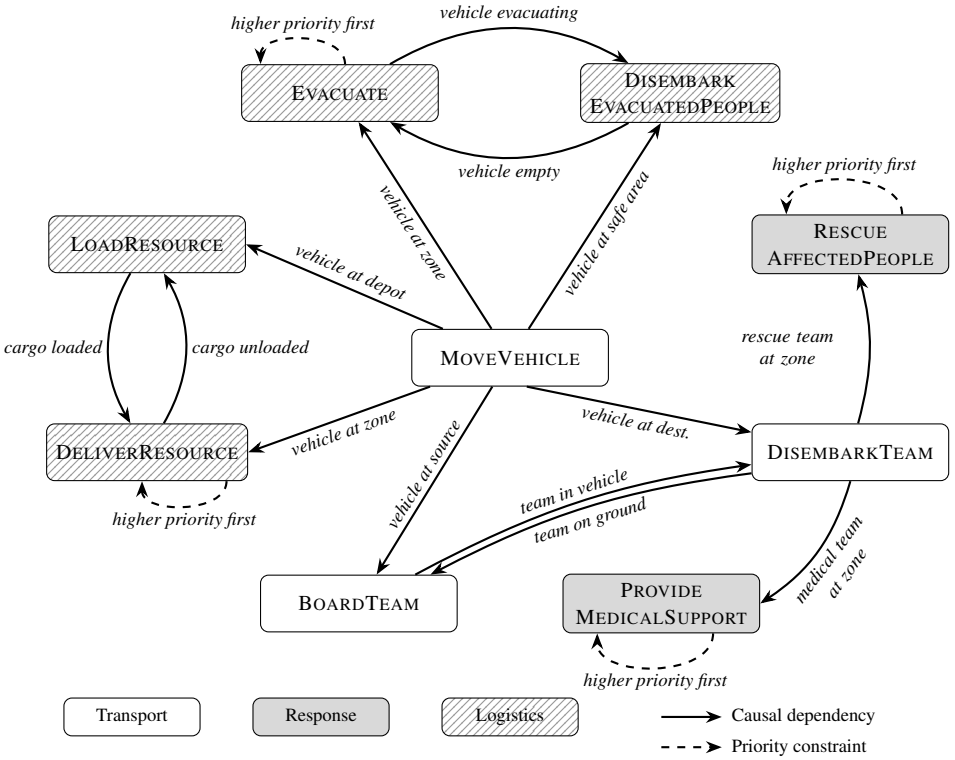

Figure 4. Dependency among the actions. Transport actions are shown in plain white, response actions in light gray, and logistics actions with a diagonal hatch pattern. Solid arrows indicate causal dependencies; however, the required fact may also be available in the initial configuration. Dashed arrows indicate priority constraints.

Table 3. Logistics action schemas. Parameter shorthands: l_A denotes an affected location; v_T denotes a transit vehicle; pf denotes the initial evacuation milestone; pt denotes the final evacuation milestone; pl denotes a higher-priority affected location than l_A; l_S denotes a safe location; r denotes a resource; v_F denotes a freight vehicle; kf denotes the initial resource milestone; kt denotes the final resource milestone; and C denotes a constant value.

Action	**Preconditions**		**Effects**	
`[Duration]`	*ann.*	*predicate(s)*	*ann.*	*predicate(s)*
EVACUATE (l_A, v_T, pf, pt, pl)	*Start*	`prior`(l_A, pl), `evac_level`$(pl,$ `p0`$)$, `evac_level`(l_A, pf), `¬evacuating`(v_T)	*Start*†	`¬evac_level`(l_A, pf), `evac_level`(l_A, pt), `evacuating`(v_T)
`[person_count_diff`(pf, pt)`]`	*Over-all*	`vehicle_at`(v_T, l_A), `valid_evac_step`(v_T, pf, pt)	–	–
DISEMBARKEVACUATED-PEOPLE (v_T, l_S)	*Start*	`evacuating`(v_T)	–	–
`[C]`	*Over-all*	`vehicle_at`(v_T, l_S)	*End*	`¬evacuating`(v_T)
LOADRESOURCE (r, l, v_F, kf, kt)	*Start*	`resource_available`(l, r, kf), `¬cargo_loaded`(v_F, r), `¬cargo_full`(v_F)	*Start*	`¬resource_available`(l, r, kf), `resource_available`(l, r, kt)
`[package_count_diff`(kf, kt)`]`	*Over-all*	`vehicle_at`(v_F, l), `valid_supply_step`(v_F, kf, kt)	*End*	`cargo_loaded`(v_F, r), `cargo_full`(v_F)
DELIVERRESOURCE $(r, l_A, v_F, kf, kt, pl)$	*Start*	`prior`(l_A, pl), `evac_level`$(pl,$ `p0`$)$, `resource_needed`$(pl, r,$ `k0`$)$, `resource_needed`(l_A, r, kf), `cargo_loaded`(v_F, r)	*Start*	`¬resource_needed`(l_A, r, kf)
`[package_count_diff`(kf, kt)`]`	*Over-all*	`vehicle_at`(v_F, l_A), `valid_supply_step`(v_F, kf, kt)	*End*	`resource_needed`(l_A, r, kt), `¬cargo_loaded`(v_F, r), `¬cargo_full`(v_F)

† Effect applied at action start, enabling concurrent scheduling.

instance, which specifies the initial state (I_f) and goal condition (G_f) of the planning task (Π_f) in Definition 7.

A problem instance instantiates the domain by declaring a finite set of objects, assigning values to all static predicates and lookup relations (e.g., route accessibility and travel times), and defining the initial distribution of resources, vehicles, response teams, and disaster-response demands. It also specifies the desired goal condition, thereby determining the objectives that must be achieved by a valid plan.

Table 4 specifies the planning problem corresponding to the representative flood-response scenario in Figure 3, instantiating the objects, initial state (I_f), and goal condition (G_f) of Definition 7. The goal condition (G_f) is achieved only when both zones have been fully evacuated (`p0`), all rescue, medical, and resource requirements in Zone (A) have been fulfilled, and no transit vehicle remains engaged in an ongoing evacuation operation. Together, the domain specification and a problem instance define a temporal planning task (Π_f) that can be solved directly by an automated planner.

Plan Validity

A solution to Π_f is a valid temporal plan $\pi_f = \langle (t_i : a_i) \rangle_{i=1}^{n}$ in the sense of Definition 5: it must be temporally consistent, satisfy each action's *at-start*, *over-all*, and *at-end* conditions at the corresponding instants, and the state reached after executing all actions must satisfy the goal G_f of Definition 7. Beyond these requirements, the flood-response task imposes one domain-specific condition, *priority ordering*. For every pair of affected locations ℓ, ℓ' with `prior`$(\ell, \ell') \in I_f$, no response action may start at ℓ while the corresponding requirement at the higher-priority location ℓ' remains outstanding. Because this is enforced as a hard *at start* precondition in every affected action schema (Tables 2 and 3) rather than as an optimisation objective, every admissible plan satisfies it by construction: no planner, regardless of its search strategy, can return a valid plan that violates it, and no post-processing is required.

Dynamic Replanning

Our flood response formulation in Definition 7 assumes that the instance has complete and correct information about the world. In practical flood-response scenarios, critical information such as route accessibility, infrastructure status, and the number of stranded individuals may not be known a priori. Thus, plans generated prior to deployment may become partially invalid as new information emerges during execution. To address this operational reality, the proposed framework supports dynamic replanning. Whenever a discrepancy between the planner's model and the actual environment is revealed, the current planning problem is updated, and a new schedule is generated from the latest observable state. This enables the framework to adapt ongoing operations while preserving actions that have already been executed.

Table 4. Planning task for the instance of Figure 3, instantiating the objects, initial state (I_f), and goal (G_f) of Definition 7.

Category	Predicates / values
Objects	
Location	`S` (SafeLocation); `A`, `B` (AffectedLocation)
Vehicle	`T` (TransitVehicle); `F` (FreightVehicle)
Team	`R` (RescueTeam); `M` (MedicalSupportTeam)
Resource	`food`
PersonCount	`p0, p20, p40, p60`
PackageCount	`k0, k100, k200, k300`
Initial state (I_f)	
Deployment	`vehicle_at(T,S)`, `vehicle_at(F,S)`, `team_at(R,S)`, `team_at(M,S)`
Priority	`prior(B,A)`
Demands	`needs_rescue(A)`, `needs_medical_support(A)`, `resource_needed(A,food,k200)`
Status	`evac_level(A,p60)`, `evac_level(B,p20)`
Supplies	`resource_available(S,food,k300)`
Static	`travel_time(T,S,A)`=20, `travel_time(T,S,B)`=30, `travel_time(F,S,A)`=25, `travel_time(F,S,B)`=40, `...` // Static `team_size`, `person_count_diff`, `package_count_diff`, `valid_evac_step`, and `valid_supply_step` values are omitted for brevity
Goal (G_f)	
Demands	`¬needs_rescue(A)`, `¬needs_medical_support(A)`, `resource_needed(A,food,k0)`
Status	`evac_level(A,p0)`, `evac_level(B,p0)`, `¬evacuating(T)`

Algorithm 1 Dynamic Replanning Procedure

Require: Original Instance $\mathcal{I}$, Original Plan π_0, Surprise Time T, Fact Changes $\mathcal{C}$, Goal Changes $\mathcal{G}$, Planner $\mathcal{P}$
Ensure: Replanned Schedule π_r

$S \leftarrow \textsc{ParseInitialState}(\mathcal{I})$
$\pi_0^{<} \leftarrow \textsc{SortByTime}(a \in \pi_0 \mid a.\text{time} < T)$
▷ **Phase 1: Reconstruct execution state**
for each action $a \in \pi_0^{<}$ **do**
  $S \leftarrow \textsc{ApplyActionEffects}(S, a)$
end for
▷ **Phase 2: Incorporate newly observed information**
$S \leftarrow \textsc{UpdateState}(S, \mathcal{C})$
$G \leftarrow \textsc{UpdateGoals}(\mathcal{I}.\text{goal}, \mathcal{G})$
▷ **Phase 3: Construct replanning problem**
$\mathcal{I}'.\text{header} \leftarrow \mathcal{I}.\text{header}$
$\mathcal{I}'.\text{objects} \leftarrow \mathcal{I}.\text{objects}$
$\mathcal{I}'.\text{init} \leftarrow \textsc{ConstructInitFacts}(S)$
$\mathcal{I}'.\text{goal} \leftarrow G$
▷ **Phase 4: Generate adapted schedule**
$\pi_r \leftarrow \textsc{Plan}(\mathcal{P}, \mathcal{I}')$
return π_r

Algorithm 1 represents the replanning process. Given an original planning instance $\mathcal{I}$ and its corresponding plan π_0, a surprise occurring at time T triggers state reconstruction by simulating all actions whose start times precede T. Newly observed facts and revised goals are then incorporated into the reconstructed state to produce an updated planning instance. Finally, the temporal planner is invoked on the updated instance to generate a revised schedule π_r that reflects the newly observed conditions.

To evaluate the scalability of the proposed domain, we generated benchmarks of increasing complexity. The benchmark-generation method and complexity characteristics are described in the following section.

Benchmark Generation and Replanning Setup

This section describes the construction of both the primary benchmark used for evaluating planning performance and the replanning benchmark used to assess adaptability under dynamic environmental changes.

Primary Benchmark Generation

We design a problem benchmark consisting of 50 problem instances to evaluate the scalability, runtime performance, and plan quality of the proposed domain. Each instance is encoded in PDDL 2.1 and ANML to evaluate across fundamentally different temporal planning paradigms: PDDL-based heuristic search planners and ANML-based timeline-centric planners.

The instances are organized into five tiers of increasing complexity, as summarized in Table 5. Each tier consists of 10 problems. All the instances from 1 to 50 correspond to an increase in the complexity of the flood instance scenario. The number of grounded action spaces reaches as large as 84,579 actions. This reflects the combinatorial nature of the binding problem. As the number of agents and locations increases, the number of possible action groundings grows multiplicatively rather than additively.

Table 5. Characteristics of instances grouped by complexity tier; each tier includes 10 problem instances, for a total of 50.

Tier	Locations	Vehicles	Teams	Estimated Actions
1	2 – 6	2 – 3	2 – 5	60 – 554
2	7 – 11	3 – 6	5 – 7	632 – 2560
3	12 – 15	6 – 8	7 – 10	2917 – 7190
4	17 – 25	8 – 12	10 – 13	8566 – 30632
5	26 – 32	12 – 17	14 – 17	32570 – 84579

Replanning Benchmark Generation

To evaluate the effectiveness of the proposed replanning approach, we generated systematically perturbed instances from the PDDL instances of the primary benchmark. We model a single re-planning step as the regeneration of a plan from the world state observed at the moment a discrepancy is revealed.

Surprise Time Selection and Perturbation Types Let π_0 denote the original plan, and the *surprise time* T is fixed at the start time of a random action near the midpoint of the plan. Placing the surprise at the midpoint ensures that a substantial portion of the plan has already been committed before the disruption occurs, while enough remains for the perturbation to demand meaningful recovery. One of two perturbation types is injected into each instance to trigger the need for re-planning:

- **Route Blockage:** A route-accessibility fact is removed for all vehicles. Before committing the removal, a reachability check verifies that every `AffectedLocation` remains reachable from at least one `SafeLocation` in the residual undirected graph. Instances where blocking would disconnect the graph fall back to the evacuee increase perturbation.
- **Evacuee Increase:** The `evac_level` of a randomly affected zone is raised to a randomly selected, strictly higher level drawn from the set of evacuation milestones. The goal condition is modified to require the zone to reach `p0`.

Replanning Instance Preparation

We only used 49 solved instances from the primary PDDL benchmark for the replanning experiment. Among those, odd-indexed instances are selected for the route blockage perturbation strategy, while even-indexed instances are selected for the evacuee increase perturbation strategy. For each instance from the primary PDDL benchmark, Algorithm 2 is applied along with the corresponding plan, random surprise time, and changes induced by the selected perturbation. One odd-indexed instance failed the required connectivity check and was reclassified as an evacuee increase, yielding a final distribution of 25 evacuee increase and 24 route blockage perturbations.

Experimental Results and Analysis

This section evaluates the results of two automated planners on both the primary benchmark and the replanning benchmark of the proposed domain. The two planners are FAPE and TFD. The evaluation addresses several questions concerning both initial plan generation and plan adaptation during execution: i) How reliably does each planner find solutions across instances of increasing complexity? ii) When both planners succeed, how do the resulting plans compare in quality, both in terms of the number of steps and the total time required to execute them? iii) How does each planner's runtime grow as the problems get larger, and at what point does growth become impractical? iv) When unexpected changes occur during execution, how much effort is required to revise existing plans, and how does the quality of the plans differ? This section addresses the findings.

Evaluation Metrics

Three standard metrics are used in the primary evaluation: Coverage, Runtime, and Plan quality.

- **Coverage** is defined as the fraction of benchmark instances for which the planner returns a valid plan within the time limit.
- **Runtime** is the time from the start of the search until the planner returns a result, measured in seconds.
- **Plan quality** is assessed along two dimensions: *plan length* and *makespan*. Plan length is the number of individual actions in the returned plan, and makespan (Definition 6) is the total time required to execute it.

Let $\pi_0^<$ denote the subsequence of π_0 containing all actions with start time $< T$, and π_r denote the plan produced for the corresponding re-planning instance, then–

- $M_r = \text{makespan}(\pi_0^<) + \text{makespan}(\pi_r)$: *Total replan makespan*, combining the makespan already spent before T with the plan π_r's own makespan.
- $\Delta M = (M_r - M_0)/M_0$: *Makespan change* relative to the original plan's makespan $M_0 = \text{makespan}(\pi_0)$.
- $\Delta|\pi|$: *Plan-length change*, comparing the length of π_0 against the combined length of the committed pre-surprise actions ($\pi_0^<$) and the plan π_r.

Algorithm 2 Replanning Instance Generation

Require: Instance $\mathcal{I}$, Original plan π_0, surprise time function $\mathcal{T}(\cdot)$, perturbations $\mathcal{P}$
Ensure: Replanning instance $\mathcal{I}'$

$T \leftarrow \mathcal{T}(\pi_0)$ ▷ **Select surprise time near plan midpoint**
$S \leftarrow \textsc{ReconstructState}(\mathcal{I}, \pi_0, T)$
$p \leftarrow \textsc{Sample}(\mathcal{P})$ ▷ **Select perturbation type**
if $p =$ Route Blockage **then**
  $S \leftarrow \textsc{RemoveRouteFacts}(S)$
  if disconnected(S) **then** $p \leftarrow$ Evacuee Increase
end if
if $p =$ Evacuee Increase **then**
  $S \leftarrow \textsc{IncreaseEvacuationDemand}(S)$
  $G \leftarrow \textsc{UpdateGoal}(\mathcal{I}.G)$
else
  $G \leftarrow \mathcal{I}.G$
end if
$\mathcal{I}' \leftarrow \textsc{ConstructInstance}(S, G)$
return $\mathcal{I}'$

- r_{rt}: *Runtime ratio*, re-plan time divided by original planning time.

The experiments were conducted on a machine equipped with a 12th Gen Intel Core i7-12700 CPU, 16GB of 3200 MT/s RAM, and an M.2 NVMe SSD, running Ubuntu 24.04 LTS. Both planners were run with a maximum time limit of 3,600 seconds. An instance is counted as solved if the planner returns a valid plan within the time limit.

Results on Primary Benchmark

Planners' coverage and runtimes are summarized in Table 6, clustered by tiers. Overall, TFD finds solutions faster than FAPE across tiers, and its runtime grows far more slowly as problem size increases. Figure 5 visualizes the cumulative coverage curve. The TFD's coverage reaches 98% well before the 20-second mark. The FAPE's curve grows more slowly, plateauing at 86% instances around 2,430 seconds.

Figure 6 shows the comparison between the planners' runtimes on the 42 instances that both solved, on a log-log scale, with points differentiated by tiers. The dashed diagonal line represents equal runtime. Every point in this plot lies above the diagonal, which confirms that TFD is consistently faster. This difference is not surprising based on how these two planners work. TFD's heuristic-based forward search leads toward the goal faster. In contrast, the cost of FAPE's maintaining a constraint network and propagating temporal and causal constraints grows with the size of the action space.

Coverage and runtime do not directly reflect the quality of the generated plans. Table 7 presents the distribution of plan quality in terms of the plan length and makespan across the solved instances. FAPE produces substantially shorter plans with a median of 39 steps, compared to TFD's median of 57, consistent with FAPE's planning approach. However, plan length is a somewhat coarse quality measure. A plan with n steps is not worse than a shorter one if it executes steps concurrently, resulting in a smaller makespan.

Makespan provides a more operationally meaningful measure, as it captures the total duration of execution under concurrency. Figure 7 plots the FAPE makespan against the TFD's for all the jointly solved instances, with the diagonal line representing equal makespan. It shows a tendency for FAPE's makespans to be usually higher than TFD's. The divergence is mostly visible at the high end. There are also several instances, points scattered below the diagonal in the lower-left region, where FAPE produces a shorter makespan than TFD. The overall picture of plan quality is that neither planner clearly dominates the other.

Coverage and runtime statistics do not directly reveal how each planner behaves as the problem grows continuously. The scalability analysis presented in Figure 8 addresses this by plotting each solved instance's runtime against the size of its grounded action space, as the grounded action determines both the branching factor of the search and the size of the constraint network.

The points representing the TFD in Figure 8 scatter linearly on the log-log axes. The overall growth rate is almost polynomial with respect to the size of the action space. The trend is consistent as the TFD planner relies on a heuristic to keep the effective branching factor low. On the other hand, points representing the FAPE in Figure 8 form a noisier scatter with noticeable variance. This variance reflects the sensitivity of FAPE's constraint propagation to the specific structure of each problem, the order in which flaws are encountered and resolved, the depth of the causal chains, and the density of temporal constraints.

The key distinction is not only speed; it is the variability of FAPE's runtime within this range compared to TFD. The runtime of TFD at any given problem size is polynomial and predictable, but the runtime of FAPE is not. For operational deployment, a predictable runtime is as important as a fast runtime, because an emergency planner is more reliable if it finishes tasks in under a few seconds.

Results on Replanning Benchmark

The benchmark of replanning instances encoding dynamic changes mid-execution is used as input to the TFD. No plan repair or reuse mechanism is used; each perturbed residual problem is solved from scratch. TFD successfully solved all 49 replanning instances. Table 8 summarizes the aggregate results. Route blockages generally produced only modest

Table 6. Planner coverage and runtime statistics by complexity. Solved refers to instances solved within the time limit of 3,600 seconds.

Tier	Planner	Solved	Runtime (s)			
			Min	Median	Max	Mean
1	FAPE (ANML)	10/10	0.421	0.737	1.709	0.806
	TFD (PDDL)	10/10	0.001	0.003	0.007	0.003
2	FAPE (ANML)	10/10	1.213	1.811	3.096	1.845
	TFD (PDDL)	10/10	0.005	0.010	0.017	0.010
3	FAPE (ANML)	9/10	6.378	9.242	28.363	11.938
	TFD (PDDL)	9/10	0.029	0.046	0.094	0.050
4	FAPE (ANML)	9/10	23.521	87.077	333.685	145.208
	TFD (PDDL)	10/10	0.067	0.161	0.433	0.213
5	FAPE (ANML)	5/10	686.735	1487.472	2429.159	1388.416
	TFD (PDDL)	10/10	0.655	1.377	19.583	3.868
All	FAPE (ANML)	**43/50**	0.421	6.635	2429.159	194.951
	TFD (PDDL)	**49/50**	0.001	0.046	19.583	0.845

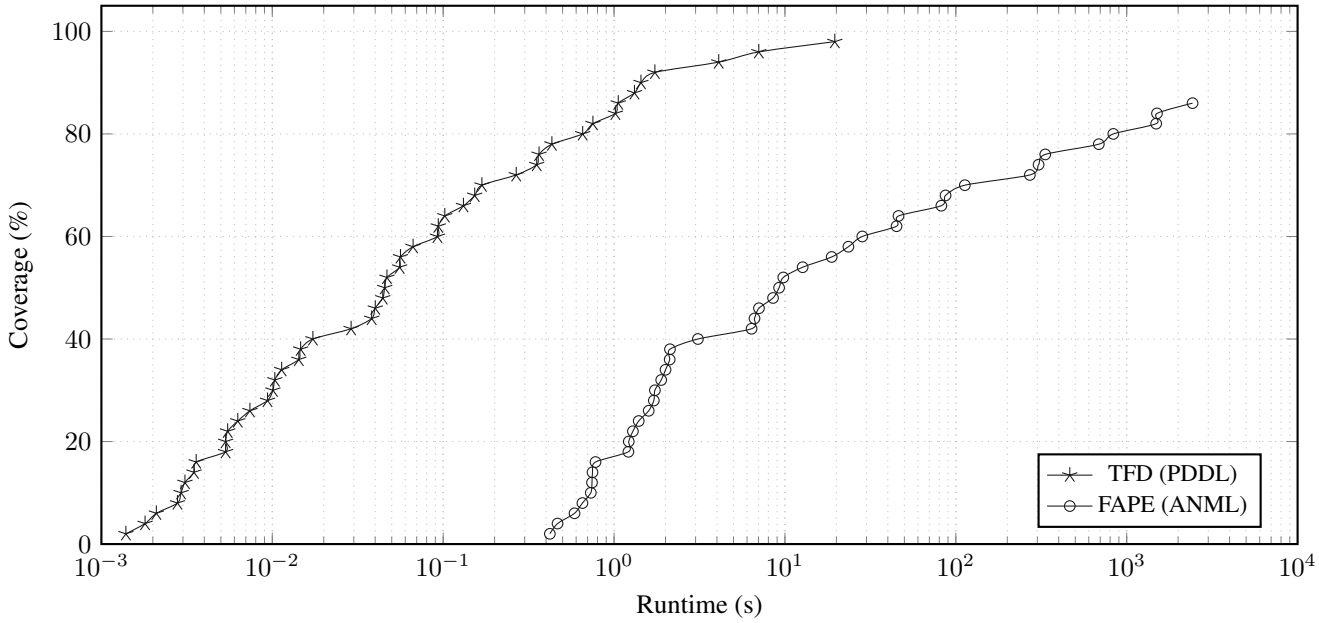

Figure 5. Percentage of instances solved (coverage) within a given runtime. TFD achieves coverage of 98% within 19.583 seconds, while FAPE plateaus at 86%.

Table 7. Plan quality comparison over the instances solved by planners. Plan Length denotes the number of plan steps, and Makespan denotes the total temporal span of execution, with smaller values indicating increased parallelism.

Planner	Plan Length				Makespan			
	Min	Mean	Median	Max	Min	Mean	Median	Max
All Instances								
FAPE (ANML)	14	49	39	140	170.0	627.4	533.0	1695.0
TFD (PDDL)	16	71	57	222	222.0	668.5	476.4	3223.8
Jointly Solved								
FAPE (ANML)	14	49	39	140	170.0	630.6	536.5	1695.0
TFD (PDDL)	16	55	46	161	222.0	483.4	469.1	1519.2

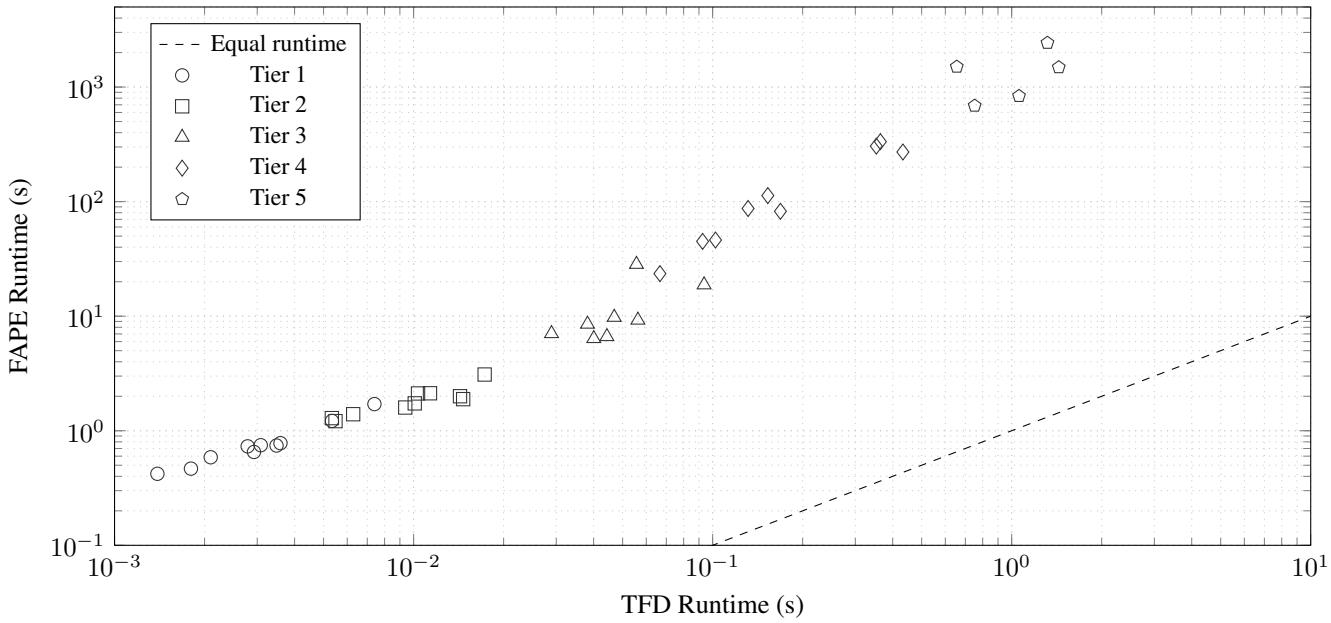

Figure 6. Log-log scatter plot of runtime for the instances solved by both planners; shapes distinguish complexity tiers. Points above the dashed diagonal indicate cases where FAPE is slower than TFD.

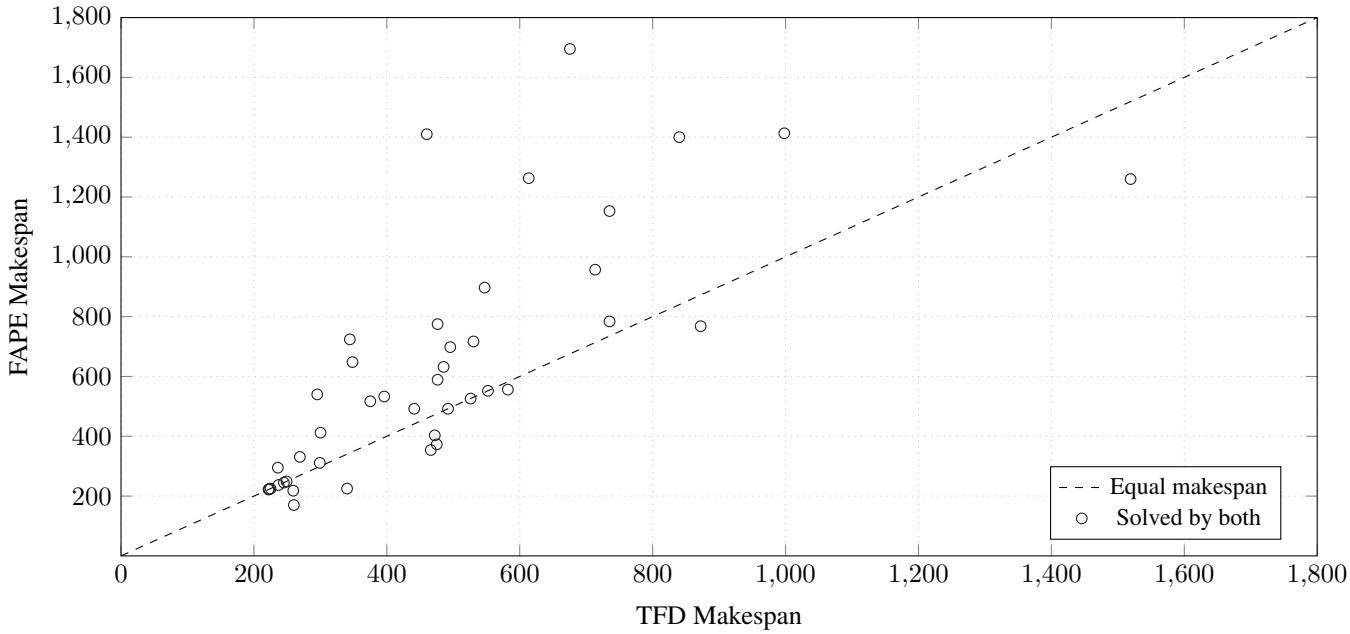

Figure 7. Plan makespan comparison for the 42 instances solved by both planners. Most points lie above the diagonal, indicating that FAPE generally produces plans with a higher makespan compared to TFD.

changes in the generated plans, with mean increases of 3.2% in makespan and 3.1% in plan length. Evacuee increases resulted in substantially larger modifications, yielding mean increases of 15.7% in makespan and 30.1% in plan length.

Across all instances, replanning remained computationally efficient, with a mean runtime ratio of $r_{\mathrm{rt}} = 0.79$ (median 0.71), indicating that replanning was typically faster than solving the corresponding original problem. The comparison between original planning and replanning runtimes is presented in Figure 9 on a shared log scale. Overall, 38 of 49 re-plans were completed in less time than the original. The reason is straightforward: the planner inherits a partially executed state, so the subproblem it must search is smaller than the original one.

Figure 10 compares replanning runtime on a logarithmic scale to the number of scheduled actions remaining in $\pi_0^{\geq}$, which is the subsequence of π_0 containing all actions with start time $\geq$ surprise time (T). A clear positive trend emerges, indicating that as the number of remaining actions grows linearly, the re-planning runtime increases exponentially. Despite this exponential growth trajectory, the absolute runtimes remain highly efficient, proving the system's viability for real-time operational deployment.

The effect of replanning on plan quality is summarized by the per-instance makespan change ΔM in Figure 11. Route

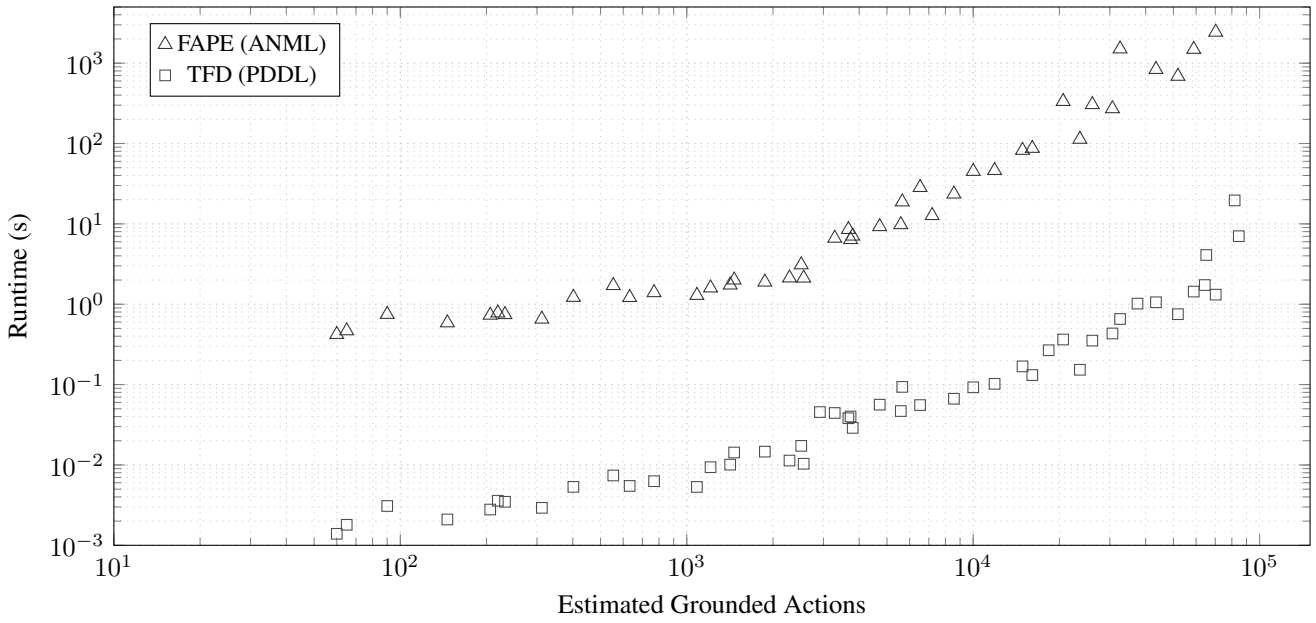

Figure 8. Log-log scatter plot of the scalability of both planners with increasing problem size, measured by the size of the grounded action space.

Table 8. Aggregate replanning statistics by perturbation type. Values are means with medians in parentheses. $\Delta M(\%)$ = percentage change in total makespan; $\Delta|\pi|(\%)$ = percentage change in plan length; r_{rt} = re-plan runtime divided by original runtime.

Perturbation	**Number of Instances**	$\Delta M(\%)$	$\mathbf{\Delta}\|\pi\|(\%)$	$\mathbf{r_{rt}}$
Evacuee Increase	25	15.7 (6.6)	30.1 (24.3)	0.78 (0.66)
Route Blockage	24	3.2 (0.9)	3.1 (3.1)	0.80 (0.73)
Overall	**49**	**9.6 (4.2)**	**16.9 (7.7)**	**0.79 (0.71)**

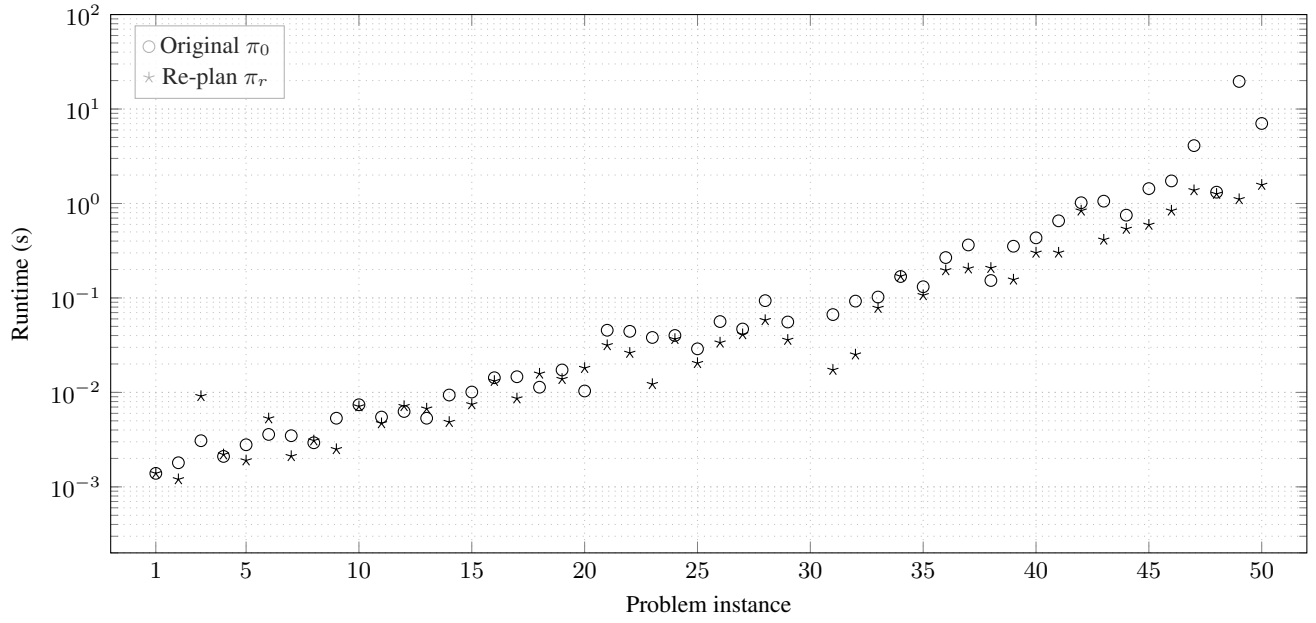

Figure 9. Original planning runtime versus re-planning runtime (log scale) for all instances, ordered by problem index. Overall, 38 of 49 re-plans were completed faster than the original plan.

blockage instances are largely unaffected: the mean increase is only 3.2% (median 0.9%), with 19 of 24 instances below 7%, because rerouting typically requires only localised vehicle reassignment.

Evacuee increase perturbations are far more variable: the mean rises to 15.7% with a standard deviation of 33.7%, reflecting the wide range of survivor-count increases imposed across instances. Notably, several large instances yield *negative* ΔM: replanning from a mid-execution state allows

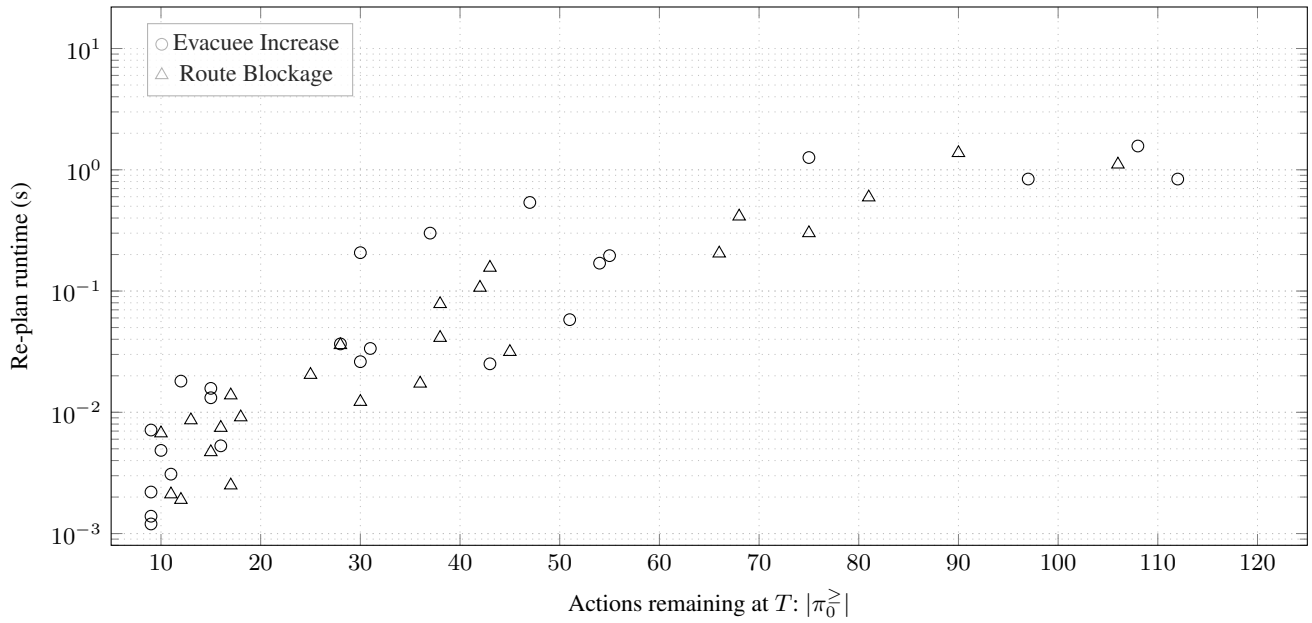

Figure 10. Re-planning runtime against residual plan length $|\pi_0^{\geq}|$ at T (log-scale). The two Re-planning costs are driven by how much of the plan remains to be reconstructed, not by what kind of perturbation triggered it.

the re-planner to discover a tighter residual schedule than the original plan's tail, resulting in a lower total makespan.

Discussion

The experimental results on the primary benchmark show that TFD is faster and more predictable than FAPE. The runtime gaps between the planners are large enough to matter in real-world deployment. Though FAPE is slower and fails on harder instances, it produces plans with fewer sequential steps, and its runtimes in small-scale instances corresponding to neighborhood-level incidents are acceptable. But for large-scale scenarios, corresponding to city-level incidents, TFD appears to be the more viable option. FAPE's lower makespan at the smaller tiers is an advantage also. A plan that completes in t time units rather than t' with $t' > t$ may correspond to dozens of additional households served before rising water levels close the last available route.

The replanning experiment under mid-execution perturbations extends the findings. Even under dynamic conditions, TFD remains highly robust and successfully solves all replanning instances. The results further confirm that replanning is generally faster than initial planning. In some cases, replanning from a partially executed state produced plans with shorter makespans than the corresponding tails of the original plans. One possible reason is that the planner identifies alternative coordination patterns that were not selected during the initial planning phase.

For operational deployment, the most robust approach is to consider the TFD and FAPE in combination, depending on requirements. For initial planning, the experimental results provide strong support for selecting TFD as the default planner because of its computational efficiency and consistent performance. FAPE is a viable choice when the scenario is small-scale, plan quality is the priority, and there is time to do better. Mid-execution replanning is a different situation entirely. The operation is already moving, and what matters is getting a usable plan back quickly. Under such conditions, TFD appears to be the more suitable choice.

Conclusion

Can automated planning be applied to produce temporally coherent response plans for floods? This was the central question addressed in this work. To answer it, a flood-response framework was developed that captures the full operational cycle of coordinated disaster response, including rescue, medical support, evacuation, and supply delivery. The proposed framework supports key operational considerations such as prioritization of affected zones, route accessibility, action concurrency, and other real-world constraints. The framework also enables mid-execution replanning under dynamic environmental changes, enhancing its applicability. To validate the framework, experiments on initial response planning and mid-execution replanning were conducted using a benchmark set of instances of the proposed framework's temporal planning domain. Results indicate that automated planning can generate temporal flood-response plans while supporting dynamic adaptation during mid-execution surprises. However, the findings are based on experiments on the benchmark instances, and additional validation using real-world flood response data and operational workflows remains necessary to assess practical deployment feasibility.

Future work could extend the proposed framework to support multi-objective optimization, such as makespan and resource efficiency, together with post-processing mechanisms that enable decision-makers to explore trade-offs among competing objectives. In addition, developing an interface that translates planner output into human-readable schedules or dispatch orders would move the proposed framework from a research prototype toward something usable in an emergency operations center.

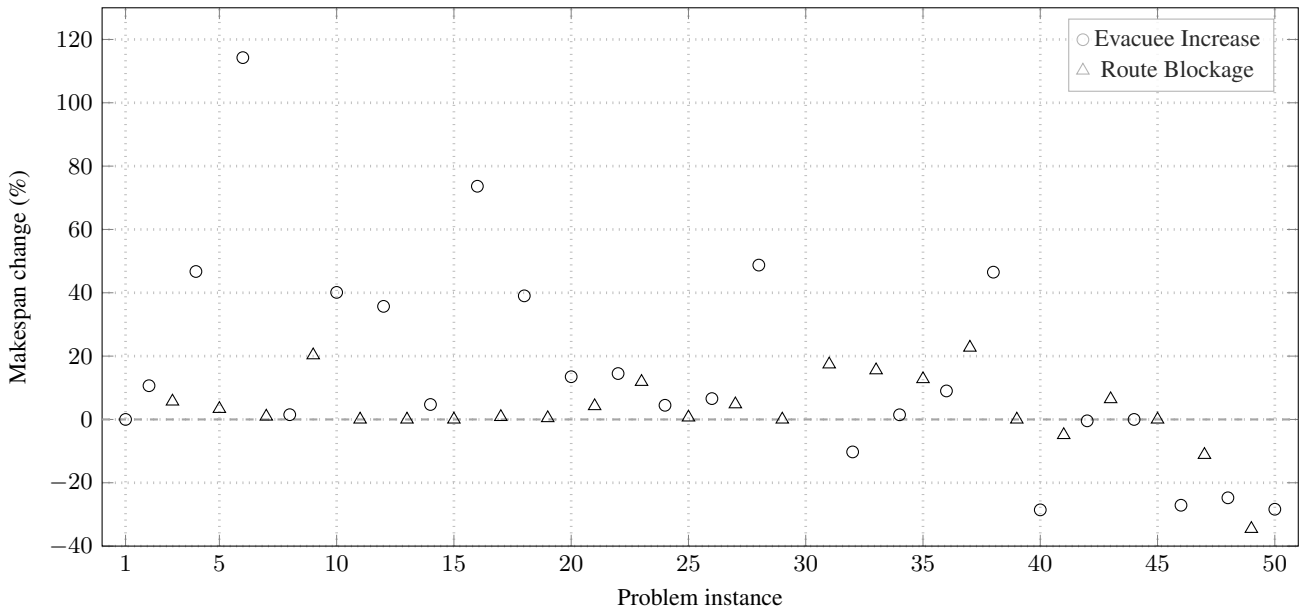

Figure 11. Percentage change in total makespan after replanning, relative to π_0's original makespan. Route blockage perturbations mostly cluster near zero throughout. Evacuee increase perturbations are far more variable. The dashed horizontal line marks zero change.

Author Contributions

All authors contributed to the conceptualization of the study and the development of the research methodology. Fazlul Hasan Siddiqui secured the project funding, administered the project, provided the necessary resources, and supervised the research. Md. Monjurul Islam conducted the investigation, developed the software, performed the visualization of the results, and prepared the original draft of the manuscript. Fazlul Hasan Siddiqui and Sabah Binte Noor validated the methodology and research findings. All authors contributed to the review and editing of the manuscript and approved the final version for publication.

Statements and Declarations

Funding

This work was supported by the University Grants Commission of Bangladesh through the Office of the Director (Research and Extension), DUET, Gazipur [grant number DUET-TRF/2025-2026/13].

Data Availability

The data supporting the findings of this study are available from the corresponding author upon reasonable request.

Code Availability

The source code used in this work is publicly available at the repository https://github.com/islammonjurul/fresched.

References

Ambily P, Chithra N and Mohammed Firoz C (2024) A framework for urban pluvial flood resilient spatial planning through blue-green infrastructure. *International Journal of Disaster Risk Reduction* 103: 104342.

Chang MS, Tseng YL and Chen JW (2007) A scenario planning approach for the flood emergency logistics preparation problem under uncertainty. *Transportation research part E: logistics and transportation review* 43(6): 737–754.

Fox M and Long D (2003) Pddl2. 1: An extension to pddl for expressing temporal planning domains. *Journal of artificial intelligence research* 20: 61–124.

Ghallab M, Howe A, Knoblock C, McDermott D, Ram A, Veloso M, Weld D and Wilkins D (1998) Pddl—the planning domain definition language. *Technical Report, Tech. Rep.* .

Ghallab M, Nau D and Traverso P (2004) *Automated Planning: theory and practice*. Elsevier.

Islam MM, Noor SB and Siddiqui FH (2025) Rapid: Resilient automated planning for intelligent disaster response. In: *2025 2nd International Conference on Next-Generation Computing, IoT and Machine Learning (NCIM)*. IEEE, pp. 1–6.

Jia H, Chen F, Pan D, Du E, Wang L, Wang N and Yang A (2022) Flood risk management in the yangtze river basin—comparison of 1998 and 2020 events. *International Journal of Disaster Risk Reduction* 68: 102724.

Kapucu N and Garayev V (2011) Collaborative decision-making in emergency and disaster management. *International Journal of Public Administration* 34(6): 366–375.

Meyer V, Priest S and Kuhlicke C (2012) Economic evaluation of structural and non-structural flood risk management measures: examples from the mulde river. *Natural Hazards* 62(2): 301–324.

Rutherford G, Kirkpatrick J, Davison A and Prahalad V (2024) Can a relational cross-scalar approach to management improve environmental disaster responses? a case study of an unprecedented flood in new south wales, australia. *Australasian Journal of Environmental Management* 31(4): 431–447.

Sayers P, Yuanyuan L, Galloway G, Penning-Rowsell E, Fuxin S, Kang W, Yiwei C and Le Quesne T (2013) Flood risk management: A strategic approach.

Schanze J (2006) Flood risk management–a basic framework. In: *Flood risk management: Hazards, vulnerability and mitigation measures*. Springer, pp. 1–20.

Simonovic SP and Ahmad S (2005) Computer-based model for flood evacuation emergency planning. *Natural Hazards* 34: 25–51.

Smith DE, Frank J and Cushing W (2008) The anml language. In: *The ICAPS-08 Workshop on Knowledge Engineering for Planning and Scheduling (KEPS)*, volume 31.

Wang W, Li Y, Zhang Y and Wu Z (2024) Pedestrian evacuation planning under dam-break flood disaster considering road risk and road pedestrian demand. *International Journal of Disaster Risk Reduction* 104: 104355.

Ye X, Wang S, Lu Z, Song Y and Yu S (2021) Towards an ai-driven framework for multi-scale urban flood resilience planning and design. *Computational Urban Science* 1: 1–12.

Yin J, Yang Y, Yu D, Lin N, Wilby R, Lane S, Sun B, Bricker J, Wright N, Yang L et al. (2024) Strategic storm flood evacuation planning for large coastal cities enables more effective transfer of elderly populations. *Nature Water* 2(3): 274–284.