# Plan Commitment: Replanning versus Plan Repair

Mohannad Babli[a,*], Óscar Sapena[a], Eva Onaindia[a]

[a]*Department of Computer Systems and Computation, Polytechnic University of Valencia, Valencia, 46022, Spain*

**Abstract**

While executing its plan in a dynamic environment where multiple agents are operating, an autonomous agent may suffer a failure due to discrepancies between the expected and actual context and thus must replace its obsolete plan. In its endeavour to fix the failure and reach its original goals, the agent may unknowingly disrupt other agents executing their plans in the same environment. We present a property for plan repair called plan commitment to ensure a responsible repair policy among agents that aims to minimise the negative impact on others. We present arguments to support the claim that plan commitment is a valuable property when an agent may have made bookings and commitments to others. We then propose C-TFLAP, an implementation of a plan repair heuristic that allows adapting a failed plan to the new context while committing as much as possible to the original plan. We demonstrate empirically that: (1) our plan repair achieves more committed plans than plan-stability repair when an agent has made bookings and commitments to others, and (2) compared to typical replanning and plan-stability repair, it can reduce the revisions among agents when failures are avoidable and can decrease the time-loss otherwise. In addition, to demonstrate extensibility, we integrate context-aware knowledge extension with committed repairing to increase the agent's chances of repairing.

*Keywords:* Automated planning, Plan repair, Plan distance

## 1. Introduction

Automated planning is the model-based approach for autonomous control devoted to studying the reasoning side of acting [22]. Planning is a crucial deliberative capability for autonomous agents to synthesise plans of action that achieve their goals in a variety of contexts like e-learning [19], space applications [13], robotics [31] and assisted living homes [3].

The successful application of automated planning requires looking into the side of acting, that is, examining the results of executing a plan in the real world [21]. Broadly speaking, there are two general ways of addressing planning and execution. The first one is to use models for planning under uncertainty that capture the partial observability and inherent uncertainty of the environment so that the calculated plans account, to some extent, for potential incidents and exogenous events that can occur during execution [10, 45, 39]. The second way lies in using classical planning models that assume a static environment only changing when some action is taken, a full observability and determinism of the actions [23, 42, 6, 32, 48]. Under this paradigm, the effects of unexpected situations during the plan execution are tackled at execution time. The first scheme uses more expressive but also more expensive planning models that can only partially account for the uncertainty in the world, whilst the second scheme relies on efficient replanning or plan repair methods. In this work, we adopt the second scheme and develop a novel plan repair strategy among multiple agents sharing the same execution environment.

A planning agent is an entity with the capacity for reasoning and acting. Ideally, a given planning agent would be responsible for synthesising and executing plans and replanning to account for changes to the world state that the agent cannot foresee. However, there are many such agents in the real world, each with different objectives, yet all tied together by a common execution environment they share. We follow the central argument presented in [47] that the single-agent planning community needs to heed the changes to the world state by generating a new (single-agent) plan that remains consistent with the larger community of agents sharing the world.

We aim to ensure the robust execution of an agent's plan in a community of agents so that it can be adapted to unforeseen situations arising from discrepancies between what an executing agent expects and what it observes. More importantly, ensure that the new adapted plan does not propagate negative consequences to other agents operating in the same environment.

Replanning consists of generating a new plan from scratch without considering the original plan by calling a planner with the new encountered world situation. Replanning is a popular method concerned with allowing an agent that suffered a failure to generate a new plan to reach its goals quickly. However, it derives a new solution without considering the existing plan. By contrast, plan repair fixes the existing plan, tries to retain its parts and structure and adapts it to the new context whilst causing minimal perturbations. In theory, in the worst case, modifying an existing plan is not more efficient than a complete replanning (from the current state) [37]. However, in practice, much prior research effort [7, 12, 34, 20] indicates that plan repair can be more effective than replanning when the plan duration (the makespan of the plan) is not the main criterion to

*Corresponding author
*Email address:* mobab@doctor.upv.es (Mohannad Babli)

optimise.

Since the plan-stability concept was defined in 2006, plan repair has proved that it could provide new plans with equivalent computation speed and with fewer revisions than replanning in the face of disruptions [23, 17]. Albeit there is a wealth of research on plan repair, there is arguably less work on repair strategies that aim to minimise the negative impact among multiple agents sharing the same execution environment, where there is a degree of privacy, reduced communication, or when an agent has bookings and commitments to others that may be costly to undo; for brevity, we will call such a setting the conservative shared setting.

Herein, we identify a new metric, plan commitment, which we claim is important in a community of agents operating in a conservative shared setting. In this situation, if an agent is solely focused on achieving its goals within the minimum time, it may propagate negative consequences to other agents and disrupt the community. Therefore, the repaired plan should be as close as possible to the original plan from the perspective of commitments made to other agents whilst achieving the original goals. When we refer to commitments in the context of a repaired plan, we are specifically referring to reusing the resources that were used in the original plan. The results of this work show that in comparison to replanning and plan-stability repair, an agent that is using plan commitment to repair its plan failures can reduce the adverse effects on other agents, i.e., can cause fewer failures to other agents in a community of agents and can reduce their time-loss if the failures are inevitable.

Our approach builds on the temporal forward partial-order TFLAP planner [43] but exploits the new plan commitment quality as a search strategy by seeding the search with the existing plan and using the new heuristic to guide the search for the most committed repair plan. Our contributions are the plan commitment property, the corresponding heuristic that guides the planner's search to achieve the most committed plan repair, and the resulting planner, which we call C-TFLAP, that repairs failures while trying to minimise the negative impact on others by generating the most committed plan when a failure occurs.

This paper is structured as follows. The next section presents the related work on handling plan execution failures in the literature. Section 3 defines the preliminaries and formalises the problem. The plan commitment property is defined in Section 4, and in Section 5, we show the implementation details of C-TFLAP. Then, we present an experimental evaluation and discuss the results in Section 6. Finally, Sections 7 and 8 show future extensions and draw some conclusions.

## 2. Related Work

The integration of planning and acting is a critical issue in designing deliberation systems, thus resulting in a range of design options for them. The first view relies entirely on planning. The second view depends mostly on execution without much efficient help from planning, whereas the third view is balanced between planning and acting [28].

For our purposes, this section examines some approaches in two categories. The first category uses predictive models while planning (before execution). The second category deals with failures without predictive models during execution (online interleaved with execution).

For the *first category*, planners need to integrate predictive models with mission specifications and provide a plan or policy as output, transforming perceived states (or beliefs) into lower-level orders, guaranteeing the mission's success and potentially meeting some optimality. Partially Observable Markov Decision Process (POMDP), contingent planning and conformant planning are ways to plan under uncertainty. In POMDP, agents use observations to form a belief of the current state, expressed as a probability distribution over the states, to find a policy prescribing which action is optimal for each belief state [5]. Contingent planning generates conditional plans given uncertainty about initial state and action effects, where the right branch will be selected during the execution through sensing [27]. The conformant planning problem can be formulated as a path-finding problem in the belief space where a sequence of actions maps a given initial belief state into a target belief [40]. Other approaches use a tri-valued logic to account for the unknown items of the world and follow an online anytime planning algorithm that allows combining planning and execution simultaneously [44]. Generally, the low predictability of most real-world environments, the immense difficulty in specifying necessary predictive models, and the computational complexity of planning under uncertainty still constrain the effectiveness of such approaches.

The *second category* of systems resulted from addressing the previous concerns by decomposing the deliberation burden into planning and acting phases, using restrictive assumptions in either phase and shifting the focus towards acting. This second category includes various approaches such as replanning, explanation-based plan repair and adaptation, replanning within limited time windows, refinement and unrefinement and iterative repair. In addition, several related works that tackle execution failures for multiple agents are reviewed for the second category. Subsequently, the remainder of this subsection delves into a comprehensive examination of several techniques and methods that fall within the scope of the second category.

Re-invoking a planner (replanning) is straightforward. However, it disregards the resources used in the existing plan and is only eager to generate a new plan to reach the original goals. On the other hand, plan repair involves adapting the existing plan to the actual execution context and making it executable again with minimal perturbation. Plan stability repair has been empirically shown to produce more stable plans than replanning [17]. CHEF [24] and GERRY [50] are two early approaches to plan repair. In CHEF, plan failures are described as causal explanations of why they occurred and are used to access abstract replanning strategies, which are then turned into changes to the faulty plans. GERRY starts with a complete but flawed schedule and then uses a scheduling and rescheduling constraint-based iterative repair to improve solutions. The plan adaptation approach [20] combines modifying the existing plan by replanning within limited temporal windows and using a local search for action graphs. The refinement and unrefinement approach [34] repairs plans by removing the obstructing actions, adding

new actions to achieve the goals, and adapting planning heuristics to remove constraints.

The Hierarchical Task Network (HTN) RepairSHOP [49] and the PANDA hybrid planner [8] consider failed traces during plan adaptation, replay the reasoning that led to the initial decision and select a different path. The forwarding dock state approach [12] collects the state information about the agent being monitored at different time intervals, including the current state, the near future state, and upcoming actions and tries to find a path from the regressed state to the current state through a breadth-first search. Regression in planning, which dates back to PLANEX [16], is a process of finding a path that applies actions backwards from the goals to the current or initial state. A regressed state is a partial expected state which contains the minimal set of conditions that must hold in the world at a specific time instance for the rest of the plan to be executable during execution and for the goals to be satisfied. SHOPFIXER [23] uses a graph of causal links and task decompositions to identify a minimal subset of the plan that must be fixed.

The robust plan execution with unexpected observations approach employs a runtime action selection policy that dynamically selects causally-valid actions that are likely to reach the goal [35]. This effectively moves the causal reasoning online instead of limiting the executor to blindly following the plan's causal structure. Although there are some similarities in purpose, our work addresses a different problem than the robust plan execution with unexpected observations approach. Their approach focuses on minimising the number of failures experienced by a single agent and reducing its need for replanning. In contrast, our work involves a community of agents where each agent must adjust its plan in the event of failure without causing disruptions for the entire community. Our objective is to minimise the need for plan revisions that impact the community as a whole. Another difference is that their approach provides solutions for a single planning problem with multiple robots possibly having concurrent actions. In contrast, in our case, different agents have different planning problems with different private goals but use some shared resources of the environment. In our work, agents operate in the same execution environment and use some shared resources. No collaboration or organisation between agents is needed. Agents devise and execute their plans to achieve their private goals. An agent communicates information about shared resources after it generates its plan, and no further communication is made to assume reduced communication settings. Our approach is independent of how agents communicate (agents can communicate with each other directly, through a mediator or in any other way). If an agent's plan fails (because of a change in the environment or incompatible information received from another agent), it repairs its plan (as a single agent), aiming to minimise the impact on other agents.

It is worth mentioning that there are other approaches, such as the situational evaluation and awareness (SEA) framework [9], which integrate both replanning and plan repair as valid options depending on the characteristics of the failure.

There have been several approaches to solving execution failures for multiple agents, such as centralised or distributed multi-agent planning (MAP) and intra-agent planning. The hybrid planning distributed approach [7] uses refinement and unrefinement. It repairs the plan by iteratively removing actions to amend the global plan of multi-robot missions involving heterogenous robots cooperating autonomously to achieve a common task. The repair approach in [33] either attempts to preserve a suffix of the existing plan and prefixes it with a newly computed plan or preserves the plan's remainder and closes the gap between the state resulting from the failed plan execution and the original goal state. Their decentralised algorithms produce a synchronous multi-agent plan or plan repair that is decomposed among agents, assuming some transparency or collaboration.

In the architecture for iterative plan repair in hierarchical multi-agent systems (HIPR) [36], the authors base their work on two ideas that agents are organised hierarchically based on attributes like location/administration and that few agents are affected by the failure. The agents are assumed to be collaborating to solve a task by decomposing global plans into local plans for individual agents. There are operational agents at the leaf level to provide operational capabilities in their work. The agents at higher levels, called managing agents, are responsible for deciding the operational agents' actions and coordination among them. This requires additional communication for coordination signalling between agents and extra computations to determine dependencies between agents, the contingency-free plan segments, and the agents affected by a hazard.

The partial satisfaction planning (PSP) intra-agent repair approach [47, 14] describes three replanning paradigms distinguished by the constraints they try to satisfy during the replanning process. These paradigms are replanning as restart, replanning to reduce computation and replanning for multi-agent scenarios. The authors in that approach refer to these constraints also as commitments. However, in their approach, the agent must publish its plan before putting it to execution, communicating it to other agents, who in turn must inform that agent of the constraints they wish to book. Hence, it knows in advance what constraints it must try to maintain if a failure occurs during execution. Consequently, all these constraints, possibly for multiple agents, must be coded and maintained as a list of new soft goals and will be added to the new problem after the failure.

The closest approach from the literature to our work is plan-stability repair [17], which defines a plan measure called the plan-stability distance to keep the repaired plan as close as possible to the original plan in terms of actions. In contrast, our approach attempts to maintain the repaired plan close to the original plan in terms of resources booked in the original plan.

## 3. Preliminary Notions

Our approach builds upon elements that comprise the Planning Domain Definition Language (PDDL) 2.2 specifications [15]. Specifically, we will use the following components of a domain:

1. A finite set of types $T$ is defined in a hierarchical structure in the PDDL domain, such that every object in the

planning problem belongs to a single type that is a leaf node in the hierarchy. The type hierarchy is expected to be organised in a reasonable and meaningful way, reflecting logical relationships between objects and their properties in the problem domain.
2. A finite set of variables $V$ that are propositional (boolean) or numeric; each $v \in V$ having a set of typed arguments $arg(v)$.
3. A finite set of operators $OP$ that are ungrounded PDDL durative actions.

The following components are used for the problem:

1. A finite set of objects $O$, each belonging to one of the types defined in $T$.
2. A finite set of state variables constituting the initial state $I$. State variables, also called fluents, are variables from $V$ whose arguments are instantiated with objects from $O$ and bound to a value, either boolean or numerical.
3. A finite set of Timed Initial Literals *TILs*: state variables denoting time-stamped changes to the world.
4. A finite set of hard goals $G$; each $g \in G$ is a fluent that the planner must achieve.

Hence for our purposes, a domain is a tuple $Dom = (T, V, OP)$, and problem is a quadruple $Prob = (O, I, TILs, G)$. The problem may vary depending on the particular instance to solve. The domain encapsulates the behaviour dynamics as a fixed description of what the planner can do to achieve $G$ starting from what is known: $I$ and $TILs$. The finite set of actions $A = \{a_1, a_2, \ldots\}$ are grounded from the operators $OP$ using objects in $O$, modelling all possible actions.

Upon receiving the domain and the problem, a planner generates a temporal plan $\pi$ as a collection of actions to satisfy $G$, where each action has a start time and a duration. During the execution of the plan, discrepancies between the expected state and the actual state of the environment may be detected, which can make the execution of the rest of the plan impossible. At this point, we have a dynamic planning problem that we define as a tuple $(G, \pi, O', I', TILs')$ where:

- $\pi$ is the original plan, designed to achieve goals $G$ according to the initial beliefs ($I$ and *TILs*).
- $(I', TILs')$ describe the new observed situation, when the failure occurred.
- We distinguish $O$ from $O'$ as new objects may be detected.

Our planner, C-TFLAP, receives the domain *Dom* and the dynamic planning problem $(G, \pi, O', I', TILs')$ as input, and our goal is to generate the most committed plan to the original plan $\pi$ that can reach the goals $G$ from the new observed situation.

## 4. Plan Commitment

This section defines a novel distance named *plan commitment*, explains how it differs from action-based measures and provides arguments and examples to support its significance for a community of agents in a conservative shared setting.

Two of the most influential plan measures from the literature were designed to maintain plan stability and foster plan diversity:

- Plan stability distance $d(\pi', \pi)$ [17] between two plans $\pi'$ and $\pi$ is defined as the number of actions that appear in a plan $\pi'$ and not in a plan $\pi$, plus the number of actions that appear in $\pi$ and not in $\pi'$. The lower the plan stability distance value is, the more stable and closer it is to the reference plan $\pi$. A plan $\pi'$ achieves better plan stability than a plan $\pi''$ if $d(\pi', \pi) < d(\pi'', \pi)$. One of the plan stability arguments is that it is intended to maintain stability between agents, making their interaction potentially predictable and easier. However, that informal argument was not tested empirically in a community of agents.
- Action distance $ad(\pi', \pi)$ [38, 46] between two plans $\pi'$ and $\pi$, is defined as the ratio of the number of non-shared actions between the new plan $\pi'$ and the old plan $\pi$ to the total number of actions appearing in one of them.

These distances are sensitive to changes in the level of actions and are therefore valuable plan measures. However, we show later in this paper that plans that are equally distant from the original plan (actions wise according to these measures) may not lead to the same disturbance levels. Despite the plans being equally distant from the original plan, some of them cause fewer disturbances among agents, leading to fewer failures (revisions) and less wasted time in the community of agents.

We propose a new plan metric we call "plan commitment" distance $c(\pi', \pi)$, which measures the distance of a new plan $\pi'$ to an original (old) plan $\pi$ in terms of objects. The lower the plan commitment distance, the closer the new plan is to the original. A plan $\pi'$ achieves better plan commitment than a plan $\pi''$ if $c(\pi', \pi) < c(\pi'', \pi)$. In essence, the plan commitment distance measure is similar to plan stability and action distance measures; however, it differs from the previously mentioned distances measures in two aspects:

- Object-sensitive: plan commitment distance works on the level of objects rather than actions.
- Type-sensitive: plan commitment distance takes into consideration the types of objects that are used in repairing.

### *4.1. Plan Commitment as a Distance*

After an execution failure occurs, we do not have a complete repair plan upfront, and we will generate a new repair plan. To get a new plan $\pi'$ that resembles the original one, $\pi$, we use our proposed distance metric based on the commitment-distance concept. First of all, we define two operators (intersection and union) that will allow us to compare how similar two actions are with respect to this metric:

**Definition 4.1** (Intersection between two actions)**.** We define the intersection operator $\tilde{\cap}$ between two actions $a'$ and $a$ as a measure of their common characteristics. These characteristics

include the objects in the action parameters, the parameter types and the operator names. The algorithm for calculating $a' \tilde{\cap} a$ can be seen in Algorithm 1.

**Algorithm 1** Intersection operator between two actions

**Input:** the new action $a'$ and the original action $a$
**Output:** the intersection value

```
1:  ▷ The intersection of action a' and a
2:  procedure a' ∩̃ a
3:      intersectedObjects ← 0
4:      ▷ Parameters of action a
5:      for each o ∈ params(a) do
6:          counter ← 0
7:          ▷ Parameters of action a'
8:          for each o' ∈ params(a') do
9:              if o = o' then                          ▷ Identical objects
10:                 counter ← 1
11:             else if type(o) = type(o') then ▷ Identical types
12:                 counter ← max(counter, 0.5)
13:         intersectedObjects += counter ▷ o and its closest object in a'
14:     if op(a') = op(a) then ▷ Instances of the same operator
15:         intersectedObjects += 1
16:     ▷ Returns the value of the intersection
17:     return intersectedObjects
```

**Definition 4.2** (Union between two actions)**.** We define the union operator $\tilde{\cup}$ between two actions $a'$ and $a$ as a measure of the total number of distinct characteristics that can be found in these actions. These characteristics include the objects in the action parameters and the operator names. This union function also prevents the distance from being negative when an object appears multiple times as a parameter in an action, as is the case in some domains. The calculation of $a' \tilde{\cup} a$ is shown in Algorithm 2.

Our goal now is to be able to assess the appropriateness of an action $a'$, in comparison to the original plan $\pi$, before proceeding to add it to the repaired plan $\pi'$. For this, we define a commitment distance function between an action $a'$ and the original plan $\pi$. This function takes into account the actions' parameters (objects) and their types and their operators' names to favour repair actions that use the same objects over those that use different objects of the same type and those that use different objects of different types.

**Definition 4.3** (Commitment distance between an action and a plan)**.** We define the commitment distance between an action $a'$ and a plan $\pi$, $\delta(a', \pi)$, as the distance that separates $a'$ from the closest action of $\pi$, as Equation 1 shows. shows. $\delta(a', \pi)$, has a value between 0 and 1. Lower values indicate greater similarity.

$$\delta(a', \pi) = \min_{a_i \in \pi} \left[ 1 - \frac{a' \tilde{\cap} a_i}{a' \tilde{\cup} a_i} \right] \quad (1)$$

**Algorithm 2** Union operator between two actions

**Input:** the new action $a'$ and the original action $a$
**Output:** the union value

```
1:  ▷ The union of action a' and a
2:  procedure a' ∪̃ a
3:      h ← φ                          ▷ Define a list that allows duplicates
4:      ▷ Parameters of the new action
5:      for each o' ∈ params(a') do
6:          h.append(o')                          ▷ Add the object name
7:      ▷ Parameters of the original action
8:      for each o ∈ params(a) do
9:          if o ∉ h then          ▷ If name of object o does not exist
10:             h.append(o)                       ▷ Add the object name
11:     h.append(op(a'))          ▷ Add the operator of action a'
12:     if op(a) ∉ h then              ▷ If operator of a is different
13:         h.append(op(a))                       ▷ Add it to the list
14:     return |h|                    ▷ Returns the length of the list
```

We consider the commitment distance $\delta(a', \pi)$ to be the key distance in our work as it will be used during heuristic evaluation to find good quality repair plans. Finally, we proceed to the definition of the commitment of a new plan $\pi'$ (assuming it is either given, generated by plan repair or by replanning) to the original plan $\pi$.

**Definition 4.4** (Commitment distance between two plans)**.** The commitment distance of a plan $\pi'$ to a plan $\pi$, $c(\pi', \pi)$, is the sum of commitment distances between the actions of $\pi'$ and $\pi$, divided by the number of actions of $\pi'$ (see Equation 2).

$$c(\pi', \pi) = \frac{\sum_{a'_i \in \pi'} \delta(a'_i, \pi)}{|\pi'|} \quad (2)$$

In general, plans that utilise the original plan's objects will have the best commitment value (closest to zero). Plans that do not utilise the original objects but utilise different objects of the same type will have a slightly worse commitment value. Plans with different objects of different types will have a worse commitment value.

### *4.2. The Value of Plan Commitment*

We present two informal arguments to demonstrate the significance of the plan commitment metric in autonomous systems. The first argument focuses on the impact of plan commitment on human interaction. The second argument, which we test empirically, focuses on the impact of plan commitment on interactions among agents.

- Significance for human interaction: users are confused by changes to plans in response to upsets. This is subjective to each user; however, trivially, change interferes with autonomy, can confuse users and make them feel they have lost control. The significance of plan commitment in human interaction is related to the negative effects of plan

changes on user satisfaction and trust. Such changes can cause confusion and a loss of autonomy, leading to dissatisfaction and mistrust in the autonomous system. By addressing user confusion and preserving autonomy through plan commitment, the design and implementation of autonomous systems can better serve and meet user needs.

- Significance for other agents: in a reduced communication environment, an agent may communicate information to other agents only at the beginning when it generates its plan; these agents, in turn, use that information when generating their own plans to reach their goals. An agent repairing its plan using plan commitment can cause fewer disruptions to other agents. Plan commitment makes the agents more predictable and aims for smooth interactions among agents. Our approach favours repairing plans using the same objects where possible or using different objects of the same types if the same objects are no longer available; otherwise, different objects of different types just to satisfy the goal. We test this informal argument empirically and show that it can reduce the number of revisions among agents and can reduce the time-loss caused by inevitable failures, as the results describe in Section 6.

Sub-optimal metrics or heuristics in planning usually do not have properties that can be formally proved; they can work better or worse depending on each specific case. For this reason, like many other metrics in planning, such as stability [17], diversity [38], dependency satisfaction [47], and causal similarity [30], we have opted for an experimental evaluation, from which we can extract statistics and behaviour trends of our approach in comparison with other existing ones.

To prevent bias, 60 trials were performed by randomising 20 problems for two agents in three benchmark domains in a conservative shared setting. The domains are a multi-agent logistics system, a multi-rover system and a multi-robot navigation system. The problems are generated by modifying the initial state, the goals, the planner's starting seeds and the failures. The modifications were generated randomly. To ensure that we were not artificially enhancing the commitments by relying on how a planner explores its search space, the original plans were calculated by a different planner. In our experiments, the original plans were generated by OPTIC, replanning was done via LPG-TD, stability repair was done via LPG-ADAPT and commitment repair was done via C-TFLAP.

We consider a motivating example domain for logistics of a package-delivery problem with drivers, trucks and packages that must be delivered to locations. The actions of the domain allow drivers to walk between locations, load packages to vehicles, board, drive, disembark from vehicles and unload packages. First, we present a simple single-agent base scenario to demonstrate the calculations of the commitment distance. More importantly, to show that two plans can be equally distanced from the original plan according to the plan-stability distance and the action distance (explained previously in this section) yet the commitment distance can distinguish them according to the resources used in each plan.

Fig. 1 shows an original plan $\pi_0$ for a single-agent simple logistics planning task in which `package0` must be delivered to `location2`, and two candidate plans $\pi_0'$ and $\pi_0''$ to fix a failure due to `truck0` becoming unavailable.

$a_1$ : 0.000 : (*load package*0 *driver*0 *truck*0 *location*0)[17.000]
$a_2$ : 17.001 : (*board driver*0 *truck*0 *location*0)[10.000]
$a_3$ : 27.002 : (*drive driver*0 *truck*0 *location*0 *location*2)[300.000]
$a_4$ : 327.003 : (*disembark driver*0 *truck*0 *location*2)[10.000]
$a_5$ : 337.004 : (*unload package*0 *driver*0 *truck*0 *location*2)[17.000]

(a) An original plan $\pi_0$.

$a_1'$ : 0.000 : (*load package*0 *driver*0 **truck2** *location*0)[17.000]
$a_2'$ : 17.001 : (*board driver*0 **truck2** *location*0)[10.000]
$a_3'$ : 27.002 : (*drive driver*0 **truck2** *location*0 *location*2)[300.000]
$a_4'$ : 327.003 : (*disembark driver*0 **truck2** *location*2)[10.000]
$a_5'$ : 337.004 : (*unload package*0 *driver*0 **truck2** *location*2)[17.000]

(b) Candidate plan $\pi_0'$.

$a_1''$ : 0.000 : (*load package*0 **driver2 truck2** *location*0)[17.000]
$a_2''$ : 17.001 : (*board* **driver2 truck2** *location*0)[10.000]
$a_3''$ : 27.002 : (*drive* **driver2 truck2** *location*0 *location*2)[300.000]
$a_4''$ : 327.003 : (*disembark* **driver2 truck2** *location*2)[10.000]
$a_5''$ : 337.004 : (*unload package*0 **driver2 truck2** *location*2)[17.000]

(c) Candidate plan $\pi_0''$.

Figure 1: The temporal plans $\pi_0$,$\pi_0'$,$\pi_0''$ in a simple single-agent logistics planning task. The number preceding the name of each action is its start time, and the number at the end (in square brackets) is its duration.

Table 1 shows how the commitment distance $c(\pi_0', \pi_0) = (\delta(a_1', \pi) + \delta(a_2', \pi) + \delta(a_3', \pi) + \delta(a_4', \pi) + \delta(a_5', \pi))/5 = 0.270$ is calculated.

| | $a_1$ | $a_2$ | $a_3$ | $a_4$ | $a_5$ | |
|---|---|---|---|---|---|---|
| $a_1'$ | 1 – 4.5/6 | 1 – 2.5/7 | 1 – 3/8 | 1 – 2/8 | 1 – 3/8 | $\delta(a_1', \pi) = 0.25$ |
| $a_2'$ | 1 – 2.5/7 | 1 – 3.5/5 | 1 – 3/7 | 1 – 2/7 | 1 – 2/8 | $\delta(a_2', \pi) = 0.30$ |
| $a_3'$ | 1 – 2.5/8 | 1 – 2.5/7 | 1 – 4.5/6 | 1 – 2.5/7 | 1 – 2.5/8 | $\delta(a_3', \pi) = 0.25$ |
| $a_4'$ | 1 – 2/8 | 1 – 2/7 | 1 – 3/7 | 1 – 3.5/5 | 1 – 2.5/7 | $\delta(a_4', \pi) = 0.30$ |
| $a_5'$ | 1 – 3/8 | 1 – 2/8 | 1 – 3/8 | 1 – 2.5/7 | 1 – 4.5/6 | $\delta(a_5', \pi) = 0.25$ |

$c(\pi_0', \pi_0) = 0.270$

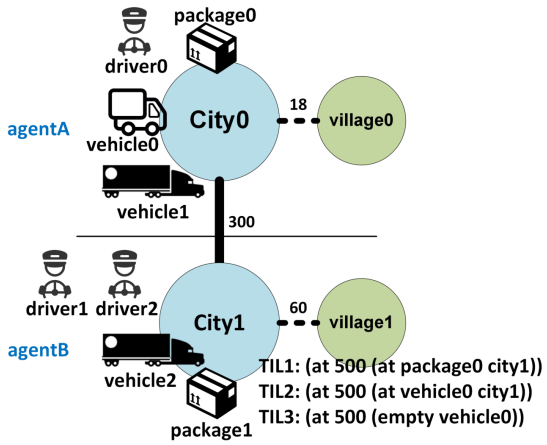
Table 1: Calculating the commitment distance of $\pi_0'$ to $\pi_0$

In action-sensitive distances such as the plan stability and the action distances, when one object changes in an action, the whole action is considered different. Consequently, once one object changes, action-sensitive distances cannot further distinguish the quality of the plans, no matter whether the change is small or large. Plan commitment distinguishes cases that the plan stability and action distances cannot distinguish. According to the makespan metric, the stability and action distances measures $\pi_0'$ and $\pi_0''$ are equally good and equally distanced from $\pi_0$. Both makespans are equal to 354. Stability distances are $d(\pi_0', \pi_0) = d(\pi_0'', \pi_0) = 10$ and action distances are $ad(\pi_0', \pi_0) = ad(\pi_0'', \pi_0) = 1$. On the other hand, according to plan commitment $c(\pi_0', \pi_0) = 0.270$, $c(\pi_0'', \pi_0) = 0.457$, as $\pi_0''$ has two objects different than $\pi_0$ in several actions (`truck2` and `driver2`), whereas $\pi_0'$ has only one object different (`truck2`).

Such plan distances (including ours) are to make the interaction between the agents potentially predictable and more manageable and promote minimum perturbation. Therefore, they must be looked at in a community of agents' context and not

solely in one agent. Our approach is independent of the way agents communicate and share information. However, for the purpose of explaining the example, we will assume that an external entity tracks the movements of the packages, keeping a repository where the agents can consult this information. The rest of the information is private unless an agent decides to share it with another agent. Initially, each agent has private goals and builds a private plan to achieve them. If they do not have enough information to generate the plan, they wait until this information is available in the repository or received from other agents. If this information represents facts expected to occur in the future, they will be modelled as Timed Initial Literals (TILs). There follows a representative yet a straightforward example of the importance of plan commitment in preventing an agent, repairing its plan, from disrupting another agent in a more elaborate logistics problem.

Fig. 2 shows a community of two agents and their plans for the logistics task.




(a) Logistics problem illustration.

$a_1$ : 0.000 : (*load package*0 *driver*0 *vehicle*0 *city*0)[17.000]
$a_2$ : 17.001 : (*board driver*0 *vehicle*0 *city*0)[10.000]
$a_3$ : 27.002 : (*drive-van driver*0 *vehicle*0 *city*0 *city*1)[300.000]
$a_4$ : 327.003 : (*disembark driver*0 *vehicle*0 *city*1)[10.000]
$a_5$ : 337.004 : (*unload package*0 *driver*0 *vehicle*0 *city*1)[17.000]

(b) `agentA` original plan $\pi_a$.

$a_1$ : 0.000 : (*load package*1 *driver*1 *vehicle*2 *city*1)[17.000]
$a_2$ : 17.001 : (*board driver*1 *vehicle*2 *city*1)[10.000]
$a_3$ : 27.002 : (*drive-truck driver*1 *vehicle*2 *city*1 *city*0)[300.000]
$a_4$ : 327.003 : (*disembark driver*1 *vehicle*2 *city*0)[10.000]
$a_5$ : 337.004 : (*unload package*1 *driver*1 *vehicle*2 *city*0)[17.000]
$a_6$ : 500.001 : (*load package*0 *driver*2 *vehicle*0 *city*1)[17.000]
$a_7$ : 517.002 : (*board driver*2 *vehicle*0 *city*1)[10.000]
$a_8$ : 527.003 : (*drive-van driver*2 *vehicle*0 *city*1 *village*1)[60.000]
$a_9$ : 587.004 : (*disembark driver*2 *vehicle*0 *village*1)[10.000]
$a_{10}$ : 597.005 : (*unload package*0 *driver*2 *vehicle*0 *village*1)[17.000]

(c) `agentB` original plan $\pi_b$.

Figure 2: A community of two agents and their plans for the logistics task.

In Fig. 2a, we have two agents (`agentA`, `agentB`), two cities (`city0`,`city1`), two villages (`village0`, `village1`), a fleet of vehicles that can be shared between agents, and contains two types: vans (`vehicle0`) and trucks (`vehicle1`, `vehicle2`). `driver0` belongs to `agentA`, and `driver1`, `driver2` both belong to `agentB`. An essential restriction in this domain that may cause a disturbance between agents is that a van can traverse any road (cities and villages), whereas a truck can only traverse between cities; Fig. 2a shows this restriction as bold and dotted connections, respectively.

`agentA` has a private goal (`at package0 city1`). `agentA` constructs a private plan $\pi_a$ (shown in Fig. 2b) and shares a piece of public information that the end location of `vehicle0` at time instance 500 is `city1`. `agentB` knows that, at time instance 500, `package0` will be at `city1`, `vehicle0` will be empty and at `city1`. These facts are represented through Timed Initial Literals (TILs).

In addition, `agentB` has two private goals (`at package1 city0`) and (`at package0 village1`). Therefore, `agentB` constructs its private plan $\pi_b$ (shown in Fig. 2c), making use of `vehicle2` to deliver `package1` to `city0` and using `vehicle0` (the vehicle of type `van` `agentA` used) to deliver `package0` to `village1`. As seen in these settings, `agentA` constructed its plan to reach its own private goals and communicated the shared information (in this case about `vehicle0`) after planning. No further communication is made to assume reduced communication settings, as explained in Section 1. `agentB` used that information to build its plan to reach its own private goals.

We are interested in showing two scenarios as examples of the advantages of plan commitment in comparison with replanning and plan-stability repair:

1. The first scenario shows that plan commitment can reduce the number of failures (revisions) among agents.
2. The second scenario shows that plan commitment can decrease the time-loss resulting from inevitable failures.

Scenario 1: `agentA` suffers a failure due to the discrepancies of `driver0` is no longer being available, `vehicle0` being at a close location `village0` instead of `city0`, and a new driver, `driver3` is available at `city0`. Whether `agentA` performs replanning or plan-stability repair (same result in this case), it will generate $\pi''_a$, which uses `vehicle1` (shown in Fig. 3b) to achieve the original goal. Consequently, `agentA` will cause a failure to `agentB` at time instance 500.001 as it can no longer execute the $a_6$ of $\pi_b$ resulting in the situation shown in Fig. 4b. `agentB` will perform replanning or repairing, which will cause it to suffer a time-loss of 368 time units, in addition to a negligible processing time for invoking repairing or replanning (or even worse as a 708 time units if `driver1` is no longer available and `driver2` is used). This 368 lost time is due to the fact that `agentB` was unable to use the truck (`vehicle1`) on the road between `city1` and `village0`, and had to retrieve the van when it was discovered to be in the wrong location at time instance 500.001. On the other hand, if `agentA` had chosen to repair its plan respecting plan commitment using $\pi'_a$, which utilises the van `vehicle0` (shown in Fig. 3a) `agentB` would have found everything as expected as shown in Fig. 4a, and would have succeeded in executing $\pi_b$. The plan commitment $c(\pi'_a, \pi_a) = 0.40$ whereas $c(\pi''_a, \pi_a) = 0.57$, the commitment repaired plan $\pi'_a$ is more committed. Although plan-stability repair (through LPG-ADAPT) has not generated a plan like $\pi'_a$, let

us assume that it did; still, it is going to favour plan $\pi''_a$ since $d(\pi'_a, \pi_a) = 9 + 5 = 14$ and $d(\pi''_a, \pi_a) = 5 + 5 = 10$ and `agentB` will suffer a failure.

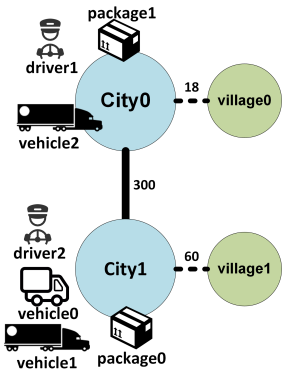
$a'_1$ : 0.002 : (*walk* **driver3** *city*0 *village*0)[50.000]
$a'_2$ : 50.010 : (*board* **driver3** *vehicle*0 *village*0)[10.000]
$a'_3$ : 60.020 : (*drive-van* **driver3** *vehicle*0 *village*0 *city*0)[18.000]
$a'_4$ : 78.030 : (*disembark* **driver3** *vehicle*0 *city*0)[10.000]
$a'_5$ : 88.040 : (*load package*0 **driver3** *vehicle*0 *city*0)[17.000]
$a'_6$ : 105.050 : (*board* **driver3** *vehicle*0 *city*0)[10.000]
$a'_7$ : 115.060 : (*drive-van* **driver3** *vehicle*0 *city*0*city*1)[300.000]
$a'_8$ : 415.070 : (*disembark* **driver3** *vehicle*0 *city*1)[10.000]
$a'_9$ : 425.080 : (*unload package*0 **driver3** *vehicle*0 *city*1)[17.000]

(a) $\pi'_a$ result of commitment plan repair for `agentA`.

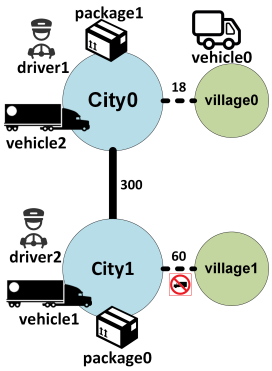
$a''_1$ : 0.003 : (*load package*0 **driver3 vehicle1** *city*0)[17.000]
$a''_2$ : 17.005 : (*board* **driver3 vehicle1** *city*0)[10.0000]
$a''_3$ : 27.008 : (*drive-truck* **driver3 vehicle1** *city*0 *city*1)[300.000]
$a''_4$ : 327.010 : (*disembark* **driver3 vehicle1** *city*1)[10.000]
$a''_5$ : 337.013 : (*unload package*0 **driver3 vehicle1** *city*1)[17.000]

(b) $\pi''_a$ result of replanning or plan-stability repair for `agentA`.

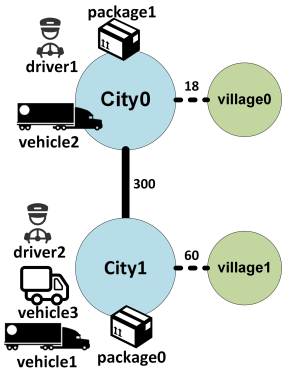
Figure 3: The candidate plans $\pi'_a$ and $\pi''_a$.



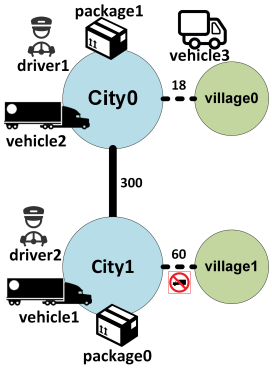

(a) `agentA` used $\pi'_a$ of scenario 1.




(b) `agentA` used $\pi''_a$ of scenario 1.




(c) `agentA` used $\pi'_a$ of scenario 2.




(d) `agentA` used $\pi''_a$ of scenario 2.

Figure 4: Snapshots of `agentB` state at time instance 500.

Scenario 2: `agentA` suffers a failure due to the discrepancies `driver0` and `vehicle0` are no longer available, a new van `vehicle3` is at a close location `village0`, and a new driver, `driver3`, is available at `city0`. Similarly, whether `agentA` performs replanning or plan-stability repair (same result), it will generate $\pi''_a$, which uses `vehicle1` (shown in Fig. 3b) to achieve the original goal. Consequently, `agentA` will cause a failure to `agentB` at time instance 500.001 as it can no longer execute the $a_6$ of $\pi_b$ resulting in the situation shown in Fig. 4d. `agentB` will perform replanning or repairing, which will cause it to suffer a time-loss of 368 time units, in addition to a negligible processing time for invoking repairing or replanning. On the other hand, if `agentA` had chosen to repair its plan respecting plan commitment using $\pi'_a$ (shown in Fig. 5), the resulting situation for `agentB` is demonstrated in Fig. 4c.

$a'_1$ : 0.002 : (*walk* **driver3** *city*0 *village*0)[50.000]
$a'_2$ : 50.010 : (*board* **driver3 vehicle3** *village*0)[10.000]
$a'_3$ : 60.020 : (*drive-van* **driver3 vehicle3** *village*0 *city*0)[18.000]
$a'_4$ : 78.030 : (*disembark* **driver3 vehicle3** *city*0)[10.000]
$a'_5$ : 88.040 : (*load package*0 **driver3 vehicle3** *city*0)[17.000]
$a'_6$ : 105.050 : (*board* **driver3 vehicle3** *city*0)[10.000]
$a'_7$ : 115.060 : (*drive-van* **driver3 vehicle3** *city*0*city*1)[300.000]
$a'_8$ : 415.070 : (*disembark* **driver3 vehicle3** *city*1)[10.000]
$a'_9$ : 425.080 : (*unload package*0 **driver3 vehicle3** *city*1)[17.000]

Figure 5: The commitment repair plan $\pi'_a$.

In that case, `agentB` would suffer a failure; however, its time-loss would have been only the negligible processing time for invoking repairing or replanning, as it can utilise `vehicle3` on the road to `village1` and `vehicle3` is already at `city1` (if `agentA` has used $\pi'_a$ for repairing its failure). The plan commitments $c(\pi'_a, \pi_a) = 0.55$, $c(\pi''_a, \pi_a) = 0.57$, thus $\pi'_a$ is more committed.

## 5. Plan Repair through Plan Commitment

General temporal planners are called "general" because they are able to handle a wide range of planning problems that involve temporal constraints, rather than being specialised for a particular domain or application. General temporal planners aim to find good quality plans, where the quality is considered in most cases as the plan makespan, i.e., plan duration. However, it is possible to define a cost function, called *metric* function, to be optimised in many of them. Unfortunately, it is not possible to define a metric using information not defined in the PDDL input files (domain and problem). In our case, we want the cost function to be the plan commitment function, which depends on the objects of the actions of the original plan $\pi$. Therefore, we need to introduce certain modifications to an existing planner to use the plan commitment to measure plan quality. We have chosen TFLAP [43], a temporal forward-chaining partial-order planner, for the following reasons:

- It has a good performance. It took third place in the temporal track of the last planning competition (IPC'2018[1]). Two planners, TemPorAl [11] and CP4TP [18], exhibited slightly better performance, but they followed a portfolio approach. This means that they integrate different temporary planners within, making it difficult to introduce modifications (each change would have to be implemented for all inner planners).

[1] https://ipc2018.bitbucket.io

- The source code of TFLAP is publicly available [2].
- TFLAP implements a standard A* search, so it is only necessary to make changes in the calculation of the evaluation function, that is, the cost of the path and the heuristic function.
- TFLAP supports planning in temporal settings.

We will call C-TFLAP, the planner resulting after the changes made in TFLAP. In the following subsections, we will describe the working scheme of C-TFLAP and the new heuristic function.

### 5.1. Working Scheme of C-TFLAP

Fig. 6 shows C-TFLAP's working scheme. The main modules of TFLAP are kept practically intact:

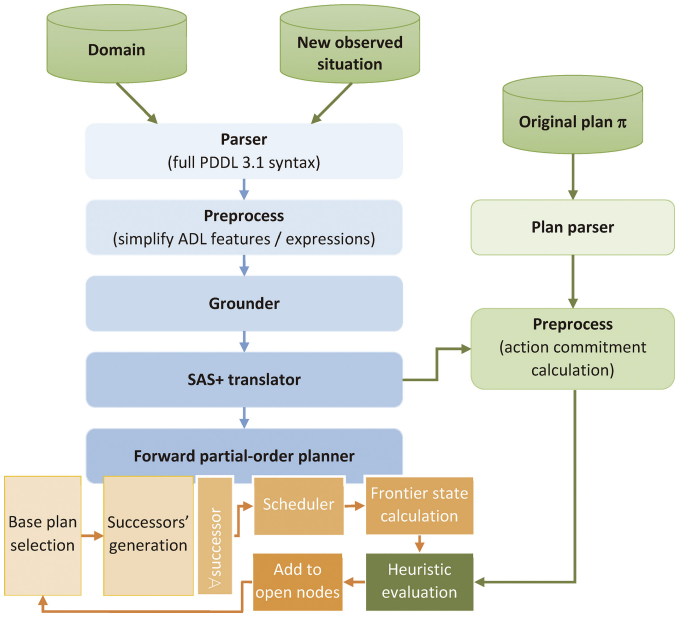



Figure 6: Working scheme of C-TFLAP.

- Initially, the domain and problem files of the planning task are sent to the parser module. Then, the information is preprocessed, grounded and translated to the SAS+ formalism [4], which is a formalism that is used in automated planning and scheduling systems to represent planning problems using multi-valued state variables instead of propositional facts.
- Then, TFLAP uses A* search guided by an evaluation function, with search nodes represented by partial-order plans. The search tree starts with a plan containing one fictitious dummy action representing the initial state. This fictitious action has no conditions and its effects correspond to the fluents of the initial state. The planner applies the following six steps at each iteration of the search process until a solution plan is found:

1. The evaluation function selects the best node, called the base plan, from the set of open nodes. Open nodes refer to a set of nodes representing potential plans or actions that have not yet been fully evaluated or expanded.
2. It generates all possible successors of the base plan. A successor plan adds a new action to the base plan by supporting its conditions and solving all threats. Threats arise if there is a resource conflict which occurs when two actions that can potentially be executed in parallel have inconsistent effects (an effect of one negates an effect of the other), interference (one deletes a precondition of the other), or competing needs (they have inconsistent preconditions).
3. Actions in each successor node are scheduled, determining their start and end times.
4. The node's schedule is used to compute the frontier state, i.e. the state resulting from executing the plan comprised in the node.
5. Successor nodes are evaluated using state-based heuristics computed on their frontier states.
6. Finally, the successor plans are added to the set of open nodes.

The first modification to make in TFLAP is to add the original plan $\pi$ as a planner input. We have developed a parser for this task. Then we proceed to compute the commitment distance of all the possible actions in the planning task to the original plan. For the commitment distance of a single action $a'$ to a plan $\pi$, $\delta(a', \pi)$ see Def. 4.3, Eq. 1. In the next section, we describe in detail how the commitment distance is used during the heuristic evaluation of the nodes to find good-quality repair plans.

### 5.2. Heuristic Evaluation Directed by the Commitment Distance

As we mentioned before, once we generate a new search node, the planner calculates its frontier state, i.e. the state resulting from executing the plan comprised in that node. Formally, we define the frontier state of a plan $\pi'$, $S(\pi')$, as a set of fluents (variable, value) that hold after the execution of $\pi'$.

The heuristic evaluation of a plan $\pi'$, $h_{cd}(\pi')$, is an estimate of the commitment distances of the actions required to reach the goal $G$ from $S(\pi')$. For this, we follow two steps:

1. A relaxed planning graph (C-RPG) is calculated. This graph is based on a *Graphplan*-like expansion where delete effects are ignored.
2. A relaxed plan is drawn by traversing the C-RPG backwards, i.e. starting from the goals. We compute $h_{cd}(\pi')$ as the sum of the commitment distances of the relaxed plan actions.

This heuristic function is calculated similarly to that proposed in [26], but modified to get estimates based on commitment distances.

[2] https://bitbucket.org/osapena/tflap

*RPG based on Commitment Distances (C-RPG)*

C-RPG is a graph where levels organise fluents and actions. The first level (level 0) is composed of the fluents held in the frontier state $S(\pi')$ of the evaluated node. The following level is action level 0, where the actions applicable in that state, i.e. whose preconditions are true in $S(\pi')$, will be located. An action will be placed at a certain level if the conditions required for it to happen are present at that level or any level before it. When an action is placed at a certain level, its outcome will be recorded at a level determined by its commitment distance from the current level.

The graph expansion continues until all the task goals are achieved. The algorithm can be observed in Algorithm 3.

**Algorithm 3** *C-RPG* expansion algorithm for a plan $\pi'$, being $\pi$ the original plan to repair.

**Input:** the plan to evaluate $\pi'$, and the dynamic planning problem $(G, \pi, O', I', TILs')$
**Output:** the fluent and action levels

1: **procedure** C-RPG
2: ▷ Initialising first fluent level with the frontier state
3: $fl \leftarrow S(\pi')$ ▷ Fluents scheduled to be inserted
4: $level(f) \leftarrow \begin{cases} 0 & \texttt{, if } f = (variable, value) \in S(\pi') \\ \infty & \texttt{, otherwise} \end{cases}$
5: ▷ Initially all actions are unreached
6: $level(a) \leftarrow \infty, \forall a \in O'$
7: ▷ New expansion stage while there are unachieved goals
8: **while** $\exists g \in G / level(g) = \infty$ **do**
9: **if** $fl = \emptyset$ **then**
10: fail ▷ Goals are not reachable
11: ▷ Get the scheduled fluent with smallest level value
12: $f \leftarrow argmin(level(f_i)), \forall f_i \in fl$
13: $fl \leftarrow fl - \{f\}$ ▷ Remove it not to be added again
14: ▷ Unreached actions with $f$ as precondition
15: **for each** $a \in O' / level(a) = \infty \wedge f \in prec(a)$ **do**
16: ▷ Check if all action preconditions are met
17: **if** $level(f_i) <= level(f), \forall f_i \in prec(a)$ **then**
18: $level(a) \leftarrow level(f)$ ▷ Action level updated
19: ▷ Level for the action effects
20: $e\_level \leftarrow level(a) + \delta(a, \pi)$
21: **for each** $f_j \in eff(a) / level(f_j) > e\_level$ **do**
22: ▷ Schedule the action effects
23: $fl \leftarrow fl \cup \{f_j\}$
24: $level(f_j) \leftarrow e\_level$

*Extraction of the Relaxed Plan*

Once the C-RPG is built, a relaxed plan is computed by traversing the graph backwards. Algorithm 4 shows how the actions of the relaxed plan are chosen. The heuristic value of plan $\pi'$, $h_{cd}(\pi')$, is then calculated as the sum of the commitment distances of these actions.

*5.3. Two Queues based Search*

The search in C-TFLAP uses two priority queues:

**Algorithm 4** Calculation of a relaxed plan to estimate the commitment distance to reach the goals from $S(\pi')$.

**Input:** the plan to evaluate, $\pi'$, the dynamic planning problem $(G, \pi, O', I', TILs')$ and the fluent and action levels
**Output:** commitment distance estimate

1: **procedure** *evaluate*
2: $h_{cd} \leftarrow 0$ ▷ Heuristic value initialisation
3: ▷ Queue the problem goals that do not hold in $S(\pi')$
4: $cond \leftarrow \{g \in G$ / level(g) $> 0\}$
5: **while** $cond \neq \emptyset$ **do**
6: ▷ Extract the fluent with the highest level value
7: $f \leftarrow argmax(level(f_i)), \forall f_i \in open\_cond$
8: $cond \leftarrow open\_cond - \{f\}$
9: **if** $level(f) = \infty$ **then**
10: **return** $\infty$ ▷ Unreachable goals
11: ▷ Lower cost action that produces $f$
12: $a \leftarrow argmin(level(a_i) + \delta(a_i, \pi)),$
$\forall a_i \in O' / f \in eff(a_i)$
13: ▷ Add the action commit. dist. to the heuristic value
14: $h_{cd} \leftarrow h_{cd} + \delta(a, \pi)$
15: ▷ Queue the action preconditions
16: $cond \leftarrow cond \bigcup \{f_i \in prec(a) / level(f_i) > 0\}$
**return** $h_{cd}$

- pqCD: this queue sorts the search nodes according to the evaluation function $f_{cd}(\pi) = g_{cd}(\pi) + h_{cd}(\pi)$, where $g_{cd}(\pi)$ is the sum of the commitment distances of the actions in $\pi$, and $h_{cd}(\pi)$ is the heuristic value of $\pi$ (described in Section 5.2). This function guides the search effectively towards solutions with small commitment distances. However, these solutions are sometimes too long because they include many redundant actions that, due to their considerable similarity with the original plan's actions, help reduce the average commitment distance. For this reason, we use a second queue.

- pqFF: this queue sorts the nodes according to the evaluation function used in the FF planner [26]: $f_{ff}(\pi) = g_{ff}(\pi) + h_{ff}(\pi)$, where $g_{ff}(\pi)$ is the number of actions in $\pi$ and $h_{ff}(\pi)$ is the length of a relaxed plan extracted from a traditional relaxed planning graph. This function leads the planner to short solution plans.

C-TFLAP uses pqCD and pqFF alternatively. Initially, both queues contain only the empty initial plan, and the active queue is pqCD. Then, the planner repeats the following steps until a solution is found:

1. The plan $\pi$ with the lower value according to its corresponding evaluation function is extracted from the active priority queue.
2. All successor plans of $\pi$ are computed, evaluated and inserted in both queues.
3. The other queue now becomes the active priority queue.

Combining these queues provides a good compromise between the commitment distance and the plan length. We have chosen this method as it is a common approach in planning used in the state-of-the-art planner LAMA [41] and Fast Downward [25]. Two heuristics are used with separate queues, thus exploiting the strengths of the utilised heuristics in an orthogonal way. We could try other methods to improve our planner's performance in future work.

## 6. Experimental Evaluation and Discussion

First, we describe the setup of the experiments, then provide a brief description of the test domains, and finally show the experiments, followed by a discussion of the results.

### *6.1. Setup of Experiments*

To validate the proposed approach for commitment plan repair, we selected three application domains: a multi-agent logistics system, a multi-rover system and a multi-robot navigation system. The domains were modelled in the PDDL 2.2 and are tailored carefully to exhibit the interactions between agents and the possible perturbation one agent can cause to others. For each domain, 20 trials were generated randomly for two agents in a conservative shared setting. In these 60 trials, the initial locations, the goals, the planner's starting seeds (for LPG-TD and LPG-ADAPT), and the failures were generated randomly. The tests were based on a pre-calculated plan given by the planner OPTIC. Then replanning, plan-stability repair and commitment plan repair are done using LPG-TD, LPG-ADAPT and C-TFLAP, respectively. This ensures that we are not artificially enhancing the results by relying on how a planner explores its search space. Plan execution was performed using the simulation system introduced by [1]. All the tests were run on an Intel(R) Core(TM) i7-6700HQ @2.60GHz CPU and 16.0 GB RAM.

### *6.2. Brief Description of the Test Domains*

*Logistics*

A community of two agents for the logistics planning task was explained in Section 4.2, where each agent has a goal to deliver a package to a specified location. The logistics planning task has drivers, cities, villages, and vans that can traverse any road, and trucks only travel between cities.

*A multi-rover system*

We designed a community of two agents (`agentA` and `agentB`) for the test problems from multi-rover systems in IPC. There are two rovers in the system. Each can independently traverse the surface of Mars from one waypoint to another visible waypoint, take pictures in different modes such as colour, high and low resolution, collect soil and rock samples and transmit all the exploration data back to the lander. The domain is tailored to exhibit interactions between rovers that may cause perturbation. More specifically, the cameras and the samplers are not built-in rovers but tools that can be equipped, used and stored inside safeboxes at waypoints. The negative interactions occur when the first agent, `agentA`, generates a plan and publishes the end location of the tools, such as the camera. The second agent, `agentB`, generates its plan depending on that information. During the execution, `agentA` suffers a failure and fixes its plan. Our interest is whether or not the failure of the first agent will also be propagated to `agentB`.

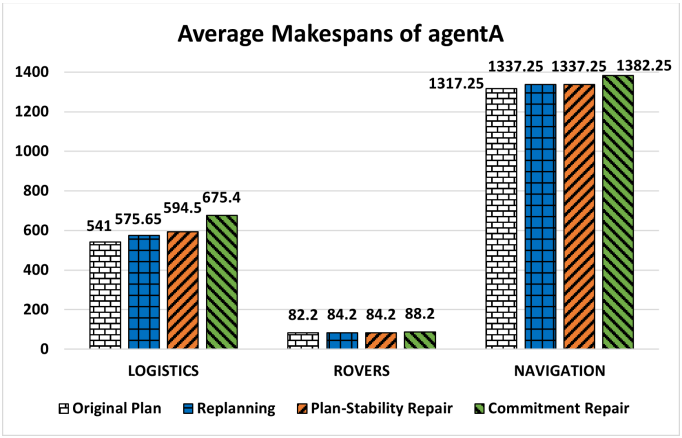
*A multi-robot navigation system with locked doors*

We designed a community of two agents (`agentA` and `agentB`) for the test problems from a multi-robot navigation system inspired by multi-robot planning with conflicts and synergies [29]. There are two robots in the system; each can independently move between locations (rooms and corridors) to navigate from their initial locations to their goal positions. Resource sharing collaborations among robots can be easily observed as the agents share the same environment and constrained resources, such as locked doors that need unlocking with the possibility of action synergy (e.g., multiple robots going through a door after a single expensive door-opening action). The domain is tailored to exhibit interactions between robots that may cause perturbation. More specifically, we added batteries and keys that the robots could equip to move and unlock doors, respectively. The negative interactions occur when the first agent, `agentA`, generates a plan and publishes the doors it will unlock, then other agents (in our case `agentB`) build their plans accordingly. During the execution, `agentA` suffers a failure and fixes its plan and our interest is whether or not `agentB` will suffer a failure.

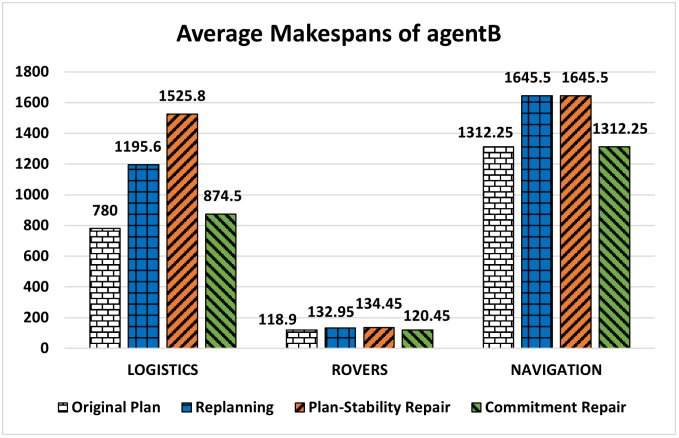
### *6.3. The Experiments*

Fig. 7 shows the average makespan of the new plans when the agents used replanning (LPG-TD), plan-stability repair (LPG-ADAPT), and commitment plan repair (C-TFLAP), for the selected test domains, compared to the original plans. In Fig. 7a, there are three clusters of columns. The first cluster of columns depicts the average makespan of plans for `agentA` in the logistics domain. In contrast, the second and the third clusters are for the rovers and the navigation domains, respectively. Within each cluster, there are four columns. The first bricks-style column represents `agentA` original plans' average makespan. The second, third and fourth columns represent the average makespan when `agentA` uses replanning, plan stability repair and commitment repair, respectively. On the other hand, Fig. 7b depicts the average makespan of `agentB`'s plans corresponding to how `agentA` fixed its plans in Fig. 7a. For example, the second column in Fig. 7b represents the average makespan of `agentB`'s plans in the logistics domain when it used replanning to fix its plan after `agentA` fixed its own plan using replanning.

Tables 2, 3 and 4 compare whether or not the new plans generated by `agentA` (when it performed replanning using LPG-TD, plan-stability repair using LPG-ADAPT and commitment repair using C-TFLAP) caused the second agent, `agentB`, to suffer failures. In addition to time lost by the agents due to failures and new plans for the logistics, rovers and navigation domains, respectively. We also compare the saved time when considering




(a) agentA.




(b) agentB.

Figure 7: The test domains' average makespan of the three approaches.

the plans of both agents. The totals and the averages for the number of failures, the agents' time-loss and the overall saved time for the experiments are calculated and shown in the last two rows of each table.

For brevity, to express time-loss and saved time, we define the following notation:

- $\pi_{ag}$ is the original plan of an agent $ag$.
- $\pi^m_{ag}$ is the new plan of agent $ag$ using method $m \in \{r, d, c\}$, where $r$ stands for replanning, $d$ for plan-stability repair, and $c$ for commitment plan repair. The agents use the same method within an experiment, e.g., when using replanning, if we have two agents, then both agents will use replanning to fix failures.
- $F_{ag_j}|\pi^m_{ag_i}$ is the number of failures suffered by an agent $ag_j$ due to $\pi^m_{ag_i}$.
- $LT^m_{ag} = makespan(\pi^m_{ag}) - makespan(\pi_{ag})$ is the time-loss of an agent $ag$ when using a particular method $m$. A negative time-loss implies saved time.
- $ST^{m2}_{m1} = \sum_{ag} makespan(\pi^{m2}_{ag}) - \sum_{ag} makespan(\pi^{m1}_{ag})$ is the saved time for all agents, defined as the makespan difference in the new plans of the agents when using method $m2$ compared to method $m1$. A negative saved time implies a time-loss.

| LOGISTICS | Replanning | | | Plan-Stability | | | | Commitment | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Nº | $F_b\|\pi^r_a$ | $LT^r_a$ | $LT^r_b$ | $F_b\|\pi^d_a$ | $LT^d_a$ | $LT^d_b$ | $ST^r_d$ | $F_b\|\pi^c_a$ | $LT^c_a$ | $LT^c_b$ | $ST^r_c$ | $ST^d_c$ |
| 0 | 1 | 0.00 | 450.00 | 1 | 0 | 530 | -80.00 | 0 | 170 | 0 | 280.00 | 360 |
| 1 | 1 | 70.00 | 780.00 | 1 | 70 | 1663 | -883.00 | 0 | 170 | 0 | 680.00 | 1563 |
| 2 | 1 | 0.00 | 470.00 | 1 | 0 | 530 | -60.00 | 1 | 0 | 460 | 10.00 | 70 |
| 3 | 1 | -137.00 | 0.00 | 0 | 240 | 0 | -377.00 | 0 | 240 | 0 | -377.00 | 0 |
| 4 | 1 | 0.00 | 780.00 | 1 | 0 | 780 | 0.00 | 0 | 300 | 0 | 480.00 | 480 |
| 5 | 1 | 240.00 | 630.00 | 1 | 240 | 1513 | -883.00 | 0 | 240 | 0 | 630.00 | 1513 |
| 6 | 0 | 0.00 | 0.00 | 0 | 0 | 0 | 0.00 | 0 | 0 | 0 | 0.00 | 0 |
| 7 | 1 | 240.00 | 780.00 | 1 | 240 | 1790 | -1010.00 | 1 | 290 | 790 | -60.00 | 950 |
| 8 | 1 | 0.00 | 780.00 | 1 | 0 | 780 | 0.00 | 0 | 300 | 0 | 480.00 | 480 |
| 9 | 0 | 0.00 | 0.00 | 0 | 0 | 0 | 0.00 | 0 | 0 | 0 | 0.00 | 0 |
| 10 | 0 | 0.00 | 0.00 | 0 | 0 | 0 | 0.00 | 0 | 0 | 0 | 0.00 | 0 |
| 11 | 1 | 70.00 | 780.00 | 1 | 70 | 1663 | -883.00 | 0 | 170 | 0 | 680.00 | 1563 |
| 12 | 1 | 0.00 | 577.00 | 1 | 0 | 577 | 0.00 | 0 | 170 | 0 | 407.00 | 407 |
| 13 | 1 | 240.00 | 630.00 | 1 | 240 | 1513 | -883.00 | 0 | 240 | 0 | 630.00 | 1513 |
| 14 | 1 | 0.00 | 0.00 | 1 | 0 | 0 | 0.00 | 1 | 0 | 0 | 0.00 | 0 |
| 15 | 1 | 120.00 | 630.00 | 1 | 120 | 1500 | -870.00 | 1 | 290 | 640 | -180.00 | 690 |
| 16 | 1 | 20.00 | -7.00 | 1 | 20 | 80 | -87.00 | 1 | 20 | 0 | -7.00 | 80 |
| 17 | 1 | 0.00 | 455.00 | 1 | 0 | 1150 | -695.00 | 1 | 88 | 0 | 367.00 | 1062 |
| 18 | 1 | 0.00 | 577.00 | 1 | 0 | 577 | 0.00 | 0 | 170 | 0 | 407.00 | 407 |
| 19 | 1 | -170.00 | 0.00 | 1 | -170 | 270 | -270.00 | 1 | -170 | 0 | 0.00 | 270 |
| **TOTAL:** | 17 | 693 | 8312 | 16 | 1070 | 14916 | -6981.00 | 7 | 2688 | 1890 | 4427 | 11408 |
| **AVERAGE:** | 0.85 | 34.65 | 415.6 | 0.8 | 53.5 | 745.8 | -349.05 | 0.35 | 134.4 | 94.5 | 221.35 | 570.4 |

Table 2: Comparison of the failures, time-loss and saved time in the logistics domain.

| ROVERS | Replanning | | | Plan-Stability | | | | Commitment | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Nº | $F_b\|\pi^r_a$ | $LT^r_a$ | $LT^r_b$ | $F_b\|\pi^d_a$ | $LT^d_a$ | $LT^d_b$ | $ST^r_d$ | $F_b\|\pi^c_a$ | $LT^c_a$ | $LT^c_b$ | $ST^r_c$ | $ST^d_c$ |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 10 | 0 | 1 | 10 | 0 | 0 | 1 | 10 | 0 | 0 | 0 |
| 2 | 1 | 0 | 50 | 1 | 0 | 50 | 0 | 1 | 15 | 0 | 35 | 35 |
| 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 20 | 30 | 0 | 20 | 30 | 0 | 0 | 20 | 0 | 30 | 30 |
| 5 | 0 | 0 | 30 | 0 | 0 | 30 | 0 | 0 | 0 | 30 | 0 | 0 |
| 6 | 1 | 0 | 50 | 1 | 0 | 50 | 0 | 1 | 0 | 0 | 50 | 50 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 20 | 0 | -20 | -20 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10 | 1 | 0 | 30 | 1 | 0 | 30 | 0 | 1 | 0 | 0 | 30 | 30 |
| 11 | 1 | 0 | 30 | 1 | 0 | 30 | 0 | 1 | 0 | 0 | 30 | 30 |
| 12 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 13 | 1 | 0 | 30 | 1 | 0 | 30 | 0 | 1 | 0 | 0 | 30 | 30 |
| 14 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 15 | 0 | -15 | -15 |
| 15 | 1 | 10 | 1 | 1 | 10 | 1 | 0 | 1 | 10 | 1 | 0 | 0 |
| 16 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 17 | 1 | 0 | 0 | 1 | 0 | 30 | -30 | 1 | 0 | 0 | 0 | 30 |
| 18 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 19 | 1 | 0 | 30 | 1 | 0 | 30 | 0 | 1 | 30 | 0 | 0 | 0 |
| **TOTAL:** | 12 | 40 | 281 | 12 | 40 | 311 | -30 | 12 | 120 | 31 | 170 | 200 |
| **AVERAGE:** | 0.6 | 2 | 14.05 | 0.6 | 2 | 15.55 | -1.5 | 0.6 | 6 | 1.55 | 8.5 | 10 |

Table 3: Comparison of the failures, time-loss and saved time in the rovers domain.

### 6.4. Discussion

First, it is worth mentioning that C-TFLAP average runtime for the experiments was 0.5 seconds. The running times are not compared since the three approaches can solve problems in a matter of seconds. The planning time is not a determining factor in choosing a concrete approach, and for our purposes, we will focus on the disruptions among agents.

We compared the stability and commitment of plans generated by the first agent, agentA, when using LPG-ADAPT for plan-stability repair and C-TFLAP for commitment repair, across different test domains in our experiments. As expected, mostly the use of one metric leads to plans that are better according to such metric. The commitment repair has achieved more committed plans than plan-stability repair, as the average commitment distances for commitment repair were 0.362,

| NAVIGATION | Replanning | | | Plan-Stability | | | | Commitment | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Nº | $F_b \vert \pi_a^r$ | $LT_a^r$ | $LT_b^r$ | $F_b \vert \pi_a^d$ | $LT_a^d$ | $LT_b^d$ | $ST_d^r$ | $F_b \vert \pi_a^c$ | $LT_a^c$ | $LT_b^c$ | $ST_c^r$ | $ST_c^d$ |
| 0 | 1 | -100 | 615 | 1 | -100 | 615 | 0 | 0 | 0 | 0 | 515 | 515 |
| 1 | 1 | 100 | 615 | 1 | 100 | 615 | 0 | 0 | 100 | 0 | 615 | 615 |
| 2 | 2 | 0 | -100 | 2 | 0 | -100 | 0 | 0 | 100 | 0 | -200 | -200 |
| 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 2 | 0 | -100 | 2 | 0 | -100 | 0 | 0 | 100 | 0 | -200 | -200 |
| 5 | 1 | 0 | 615 | 1 | 0 | 615 | 0 | 0 | 100 | 0 | 515 | 515 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 1 | 100 | 615 | 1 | 100 | 615 | 0 | 0 | 100 | 0 | 615 | 615 |
| 8 | 0 | -100 | 0 | 0 | -100 | 0 | 0 | 0 | -100 | 0 | 0 | 0 |
| 9 | 1 | 0 | 615 | 1 | 0 | 615 | 0 | 0 | 100 | 0 | 515 | 515 |
| 10 | 0 | 100 | 0 | 0 | 100 | 0 | 0 | 0 | 100 | 0 | 0 | 0 |
| 11 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 12 | 1 | 0 | 615 | 1 | 0 | 615 | 0 | 0 | 100 | 0 | 515 | 515 |
| 13 | 1 | 0 | 615 | 1 | 0 | 615 | 0 | 0 | 100 | 0 | 515 | 515 |
| 14 | 1 | 100 | 615 | 1 | 100 | 615 | 0 | 0 | 100 | 0 | 615 | 615 |
| 15 | 1 | 100 | 615 | 1 | 100 | 615 | 0 | 0 | 100 | 0 | 615 | 615 |
| 16 | 1 | 100 | 615 | 1 | 100 | 615 | 0 | 0 | 100 | 0 | 615 | 615 |
| 17 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 100 | 0 | -100 | -100 |
| 18 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 19 | 1 | 0 | 715 | 1 | 0 | 715 | 0 | 0 | 100 | 0 | 615 | 615 |
| **TOTAL:** | 15 | 400 | 6665 | 15 | 400 | 6665 | 0 | 0 | 1300 | 0 | 5765 | 5765 |
| **AVERAGE:** | 0.75 | 20 | 333.25 | 0.75 | 20 | 333.25 | 0 | 0 | 65 | 0 | 288.25 | 288.25 |

Table 4: Comparison of the failures, time-loss and saved time in the navigation domain.

0.108 and 0.275 compared to 0.450, 0.132 and 0.405, in the logistics, rovers, and navigation domains, respectively.

For the logistics test domain, commitment repair has avoided causing a failure to `agentB` in 65% of the experiments, compared to 15% and 20% in replanning and plan-stability repair, respectively. `agentB` average time-loss when `agentA` used commitment repair equals 23% of the average time-loss when it used replanning and 13% of the average time-loss when it used plan-stability repair.

For the rovers test domain, even though each of the three approaches has avoided causing failures in `agentB` in 40% of the experiments, this does not mean that they are equally good. Since `agentB` average time-loss when `agentA` used commitment repair equals 11% of the average time-loss when it used replanning and 10% of the average time-loss when it used plan-stability repair. In addition, using commitment repair did not result in time-loss for `agentB` in 11 out of 12 times when `agentB` failures were inevitable, in comparison with 5 out of 12 when it used replanning, and 4 out of 12 when it used plan-stability repair.

For the navigation test domain, commitment repair has avoided causing a failure to `agentB` in 100% (all) of the experiments compared to 35% in replanning and plan-stability repair. `agentB` average time-loss when `agentA` used commitment repair equals 0% of the average time-loss when it used replanning or plan-stability, as the average time-loss when `agentA` used commitment repair equals 0.00, compared to 333.25 when `agentA` used either replanning or plan-stability repair.

We also find it interesting to compare the saved time when considering implications for all agents. When we compared the saved time when the agents used commitment repair against the results of when they used plan-stability repair, we found that:

- For the logistics domain, commitment repair has saved time in 15 experiments and neither saved nor wasted time in 5 experiments, with 570.4 average saved time.
- For the rovers domain, commitment repair has saved time in 7 experiments, wasted time in 2 experiments and neither saved nor wasted in 11 experiments, with 10 average saved time.
- For the navigation domain, commitment repair has saved time in 11 experiments, wasted time in 3 experiments and neither saved nor wasted in 6 experiments, with 288.25 average saved time.

When we compared the saved time when the agents used plan-stability against the results of when they used replanning, we found that:

- For the logistics domain, plan-stability repair has wasted time in 12 experiments and neither saved nor wasted time in 8 experiments, with 349 average wasted time.
- For the rovers domain, plan-stability repair has wasted time in 1 experiment and neither saved nor wasted time in 19 experiments, with 1.5 average wasted time.
- For the navigation domain, plan-stability repair has neither saved nor wasted time in 20 experiments.

When we compared the saved time when the agents used commitment repair against the results of when they used replanning, we found that:

- For the logistics domain, commitment repair has saved time in 11 experiments, wasted time in 4 experiments and neither saved nor wasted time in 5 experiments, with 221 average saved time.
- For the rovers domain, commitment repair has saved time in 6 experiments, wasted time in 2 experiments and neither saved nor wasted time in 12 experiments, with 8.5 average saved time.
- For the navigation domain, commitment repair has saved time in 11 experiments, wasted time in 3 experiments and neither saved nor wasted time in 6 experiments, with 288 average saved time.

The results described above support our claim of the significance of plan commitment for other agents. It reduced the number of failures and the time-loss when failures were avoidable and decreased the time-loss when failures were inevitable. In addition, we found that commitment repair has achieved better saved time than plan-stability repair when considering implications for all agents and when contrasting each approach against replanning.

It is worth noting that there are situations when both commitment repair and plan-stability repair have wasted time compared to replanning, such as experiments 3 and 15 in the logistics domain. This is natural and was expected as the time waste occurs when there is no dependency between the agents; as the first agent suffers a failure, the second agent's plan is unaffected. When the first agent uses replanning to fix its failure, it generates the shortest makespan plan. In contrast, it generates longer plans when it uses commitment repair and plan-stability repair, leading to a loss of time.

According to Fig. 7, when comparing the average makespan of the agents' new plans when they used replanning, plan-stability and commitment repair in the three test domains, we found that plan commitment generated longer average makespan plans for `agentA` and that it generated shorter average makespan plans for `agentB`. In addition, based on our experiments, it may be observed that: 1) in a few cases when LPG-ADAPT seeks to generate a more stable plan, it adds many instances of the original plan's actions, which increases the stability but results in longer makespans and causes a time-loss for the agent compared to replanning which only seeks the shortest makespan plan. 2) Plan stability focuses on generating a higher stability plan (within the agent itself of its new plan compared to its original plan). However, plan stability does not correlate with makespan, i.e., higher stability plans may have longer or shorter makespans than lower stability plans. More importantly, a higher stability plan for an agent does not necessarily mean it will cause less disturbance to other agents.

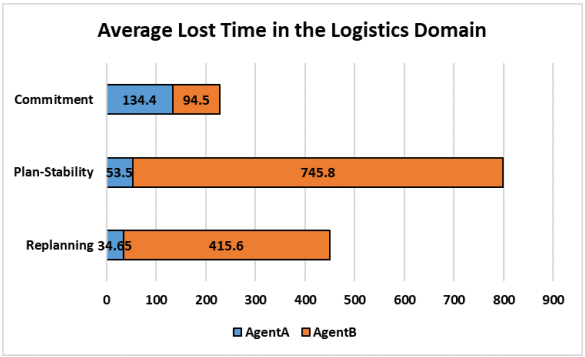
Fig. 8 compares the agents' average time-loss when using replanning, plan stability, and commitment repair in the three test domains to highlight further the additional costs (increased time-loss) incurred by the agent that suffered the failure (`agentA`) and the resulting decreased time-loss experienced by other agents (`agentB`). For brevity, there follows a detailed comparison of the values for the Logistics domain (Fig. 8a), whereas the values' comparison for the other domains in Fig. 8b and Fig. 8c are shown in Table 5.

In Fig. 8a, the comparison between the commitment column (the top column) and the plan-stability column (the middle column) demonstrates that for the Logistics domain, `agentA`'s average time-loss using commitment plan repair (134.4) exceeds its value when it used plan stability (53.5) by approximately 1.512 times. Correspondingly, the resulting `agentB`'s average time-loss using commitment plan repair (94.5) is approximately 6.89 times less than its value when it used plan stability (745.8). Similarly, the comparison between the commitment column (the top column) and the replanning column (the bottom column) demonstrates that for the Logistics domain, `agentA`'s average time-loss using commitment plan repair (134.4) exceeds its value when it used replanning (34.65) by approximately 2.875 times. Correspondingly, the resulting `agentB`'s average time-loss using commitment plan repair (94.5) is approximately 3.397 times less than its value when it used replanning (415.6).

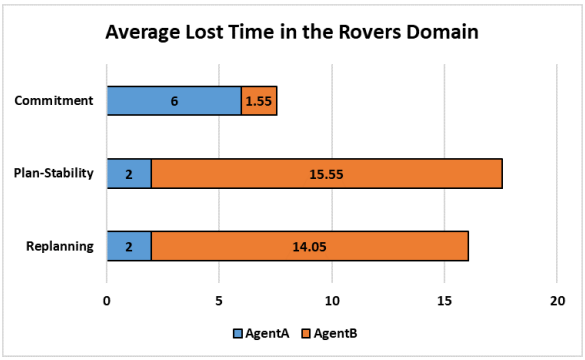
| | Change in Average Time-Loss | | | | | |
|---|---|---|---|---|---|---|
| | Logistics | | ROVERS | | NAVIGATION | |
| | $AVG.\ LT_c^r$ | $AVG.\ LT_c^d$ | $AVG.\ LT_c^r$ | $AVG.\ LT_c^d$ | $AVG.\ LT_c^r$ | $AVG.\ LT_c^d$ |
| **agentA** | 2.88 times more | 1.51 times more | 2.00 times more | 22.00 times more | 2.25 times more | 2.25 times more |
| **agentB** | 3.4 times less | 6.89 times less | 8.06 times less | 9.03 times less | -333.25 | -333.25 |

Table 5: The change in each agent's average time-loss in the three application domains. Where $AVG.\ LT_c^r$ is the average time-loss when an agent used commitment repair compared to replanning, and $AVG.\ LT_c^d$ is the average time-loss when an agent used commitment repair compared to plan stability.

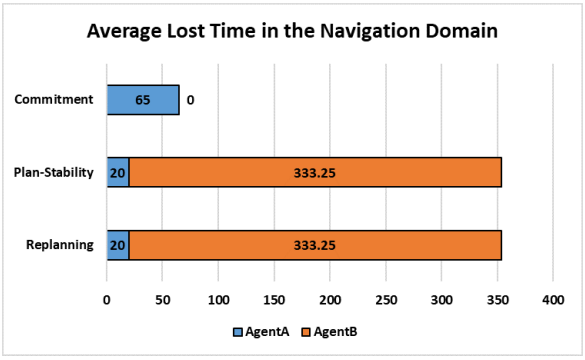
As can be observed in Table 5, the plan commitment metric has generated longer plans for the agent that suffered the failure (increased time-loss) and that it generated shorter plans for other agents (decreased time-loss) compared to replanning and plan stability repair. In addition, plan stability and replanning tend to optimise the agent's plan without considering the negative effects they will cause on other agents. Commitment repair, on the contrary, generates longer plans but significantly minimises the damage it causes to other agents.




(a) Logistics.




(b) Rovers.




(c) Navigation.

Figure 8: The agents' average time-loss using the three approaches in the test domains.

In light of this finding and the positive results obtained when comparing the overall saved time for all agents, the results suggest that commitment repair by an agent can have a significant positive impact on other agents (indicated by the reduced time-loss of the second agent, the reduced number of propagated failures and the total saved time) if that agent is willing to endure some loss of utility (indicated by the increased makespan for the agent that originally suffered a failure).

This study acknowledges that the plan commitment metric is a more sensitive metric than the plan stability metric for plan

repairing among multiple agents sharing the same execution environment in a conservative shared setting. Once one object changes inside an action, the plan stability metric (or other action-based metrics) cannot further distinguish the quality of the plans, no matter whether the change is small or large. Therefore, they cannot distinguish plans that disrupt other agents from those that do not, unlike the plan commitment metric, which accounts for and is sensitive to every change.

On the one hand, the plan commitment metric produces better results than other metrics whenever the agents use shared resources for generating their plans, one agent suffers a failure, multiple possible plans exist to fix the failure and reach the agent's goals, and some of these plans utilise the shared resources in a way that do not disrupt other agents while other plans do not. In agents that use the plan commitment metric, plans that utilise the same shared resources of the original plan are prioritised over plans that utilise alternative equivalents or resources of different types.

- Plans that utilise the same shared resources that were used in the original plan lead to meeting other agents' expectations, avoiding causing them failures and preventing time-loss.
- Plans that utilise alternative equivalent resources also meet other agents' expectations, despite causing them failures but preventing time-loss since the resource is equivalent and can be used as an alternative.
- Finally, plans that utilise resources of different types discard other agents' expectations; thus, they cause failures and time-loss to other agents.

Action-based metric distances do not distinguish the previously mentioned types of plans; thus, they are indifferent about selecting plans that disturb other agents.

On the other hand, if there is only one way for the agent to fix the failure and achieve its original goals, then an agent that uses plan commitment would have an equal performance as an agent that uses any other metric. Furthermore, in situations where there is no agent dependency (no shared resources) and multiple ways to fix a failure, the additional plan length resulting from generating a more stable or committed plan by the agent that suffered the failure does not lead to saving the other agents' time. In contrast, optimising other metrics of value to the agent, such as makespan, is more beneficial for the community of independent agents.

In the experiments, some failures were inevitable in both the logistics and rovers domains. In contrast, all the failures were avoidable in the navigation domain. The different nature of the test domains and the variety of the experiments made it interesting to examine the agents' behaviour when using the three approaches (replanning, plan-stability repair or commitment plan repair).

## 7. Extensibility and Future Work

*Extensibility*

C-TFLAP can be modified by manipulating the heuristic function; therefore, it can be easily extended to include new functionalities. There follows a brief description of how we extend its usage with a cognitive semantic layer to increase the agents' chances of a successful repair.

Automated planning knowledge extension is a cognitive semantic approach that involves adding new object types to the predefined list of object types for a planning task in a dynamic way [2]. This approach is used to make the system more context-aware and better able to reason with incomplete knowledge, which can improve its autonomy and chances of successful repairing.

Using the approach presented in [2], we utilised the context-aware knowledge acquisition that was designed initially to formulate new goals. The ontological details are abstracted out in this paper, and a detailed description can be found in [2]. For illustration, the output for the logistics application domain presented in this paper is shown in Fig. 9.

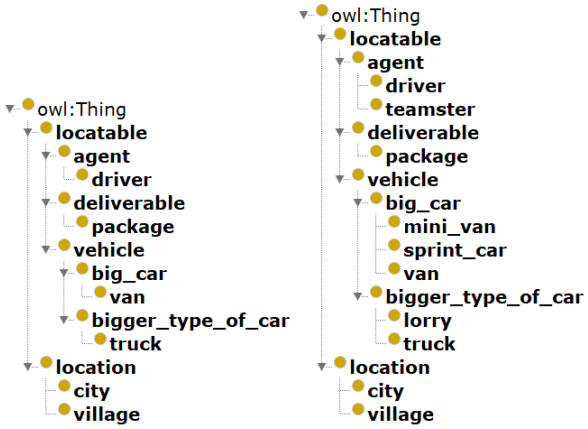



Figure 9: OWL Representation of the hierarchy of the predefined set of object types and the augmented hierarchy of types.

When using automated planning knowledge extension, the domain is augmented with new types, and therefore it must be passed to C-TFLAP. In addition, an important requirement for this extension to work is to have the domain types modelled in a reasonable hierarchy in which the types that must be used differently (such as `van` and `truck` in the logistics domain explained in Section 4.2) must not be modelled as siblings under the same parent. It is worth mentioning that typing in PDDL is domain-independent as PDDL offers the ability to express a type structure for the objects in a domain which allows typing of the parameters that appear in variables and operators. Furthermore, PDDL allows the types to be expressed as forming a particular hierarchy.

Fig. 9 demonstrates how `van` and `truck` are not sibling types, and they are modelled under different parent types, `big_car` and `bigger_type_of_car`, respectively. When this module is used, The condition of Line 11 in Algorithm 1 becomes:

**`else if`** $type(o) = type(o')$ **`or`** $parentType(o) = parentType(o')$ **`then`** $counter \leftarrow \max(counter, 0.5)$

Therefore, for the logistics problem shown in Section 4.2, the

algorithm would favour plans that use the same object used in the original plan (the same van, `vehicle0` as in Scenario 1), otherwise a different object from the same type (a different van, `vehicle3` as in Scenario 2) or a different object from a different type that is compatible with the original object type (`vehicle4` of a new type `sprinter` that is compatible with the type `van` as the extended scenario shown below), otherwise, a different object from a non-compatible type (`vehicle1` of a type `truck`) just to satisfy the goal.

Scenario (extended): `agentA` that was shown in Fig. 2a, suffers a failure due to the discrepancies `driver0` and `vehicle0` are no longer available, a new driver `driver3` is available at `city0` and `vehicle4` is at a close location `village0`, which is a new object of a new type `sprinter` unknown to the agent. Whether `agentA` performs replanning or plan-stability repair, it will discard `vehicle4`. Similarly to scenarios 1 and 2, it will generate the plan $\pi''_a$ which uses `vehicle1` of type `truck` (shown in Fig. 3b) and thus, `agentB` will suffer a failure. On the other hand, if `agentA` and `agentB` utilise automated planning knowledge extension `agentA` would have realised that the type `sprinter` is equivalent to `van`. Then, `agentA` would have performed commitment plan repair which considers objects types, generating the plan $\pi'_a$ shown in Fig. 10, which utilises `vehicle4`.

$a'_1$ : 0.002 : (*walk* **driver3** *city*0 *village*0)[50.000]
$a'_2$ : 50.010 : (*board* **driver3 vehicle4** *village*0)[10.000]
$a'_3$ : 60.020 : (*drive-van* **driver3 vehicle4** *village*0 *city*0)[18.000]
$a'_4$ : 78.030 : (*disembark* **driver3 vehicle4** *city*0)[10.000]
$a'_5$ : 88.040 : (*load package*0 **driver3 vehicle4** *city*0)[17.000]
$a'_6$ : 105.050 : (*board* **driver3 vehicle4** *city*0)[10.000]
$a'_7$ : 115.060 : (*drive-van* **driver3 vehicle4** *city*0*city*1)[300.000]
$a'_8$ : 415.070 : (*disembark* **driver3 vehicle4** *city*1)[10.000]
$a'_9$ : 425.080 : (*unload package*0 **driver3 vehicle4** *city*1)[17.000]

Figure 10: The commitment repair plan $\pi'_a$.

In that case, `agentB` would have suffered a failure and fixed it (with negligible processing time for invoking repairing or replanning) using `vehicle4`, thus avoiding time-loss of 368 time units. The plan commitments $c(\pi'_a, \pi_a) = 0.55$, $c(\pi''_a, \pi_a) = 0.57$, thus $\pi'_a$ is more committed. The authors did not use this extension in the evaluation against replanning and stability repair to avoid bias, as those methods cannot handle new objects of different but equivalent types.

*Future Work*

Multiple sound mathematical formulations may exist to define a plan metric distance; while all are mathematically correct, their applicability may vary depending on the specific context of the problem at hand. The plan commitment metric is designed to focus on the resources used within an action. In the vast majority of cases, two actions of the same operator that use the same resources are opposite actions. For example, drive from city A to city B and drive from city B to city A, or stack block A on block B and stack block B on block A. Although the plan commitment metric values both actions in the same way, the planner will select only those that lead to the objective. Additionally, in the future, if we want to compare plans of different agents, who probably use different models, our approximation will be more general and robust than an approximation that focuses on the ordering of objects within actions. This work outcome leaves many open lines of research and future development. There follow some of the implications and suggested future work.

It would be interesting to test with different types of agents (self-interested and collaborative) in the same community for future work. Currently, there is no limit on time/cost for the newly generated plan, and it may be interesting to specify or discover a threshold according to a library of plans. The current implementation is domain-independent, which may be considered an advantage. All objects are treated as equally important; however, for enhanced performance in specific applications, the plan commitment equation (shown in Section 4) can be easily modified to give some objects more weight. E.g., the objects that matter to other agents, instead of trying to commit to all the objects used in the original plan, possibly resulting in longer plans that discourage self-interested agents and may be considered a limitation in some cases. On the other hand, autonomously identifying the objects that matter more for each community of agents would require further investigation and research. C-TFLAP can continue to search for solutions with better commitment value after finding the first solution (as with LPG-ADAPT). Studying the utility of stability and commitment for an agent to autonomously decide when better stability or better commitment plan is worth the computation and the additional plan length after finding the first solution can be interesting. Furthermore, it may also be interesting to use the plan commitment in single-agent settings inversely, use the least committed plans for plan diversity purposes, and explore different solutions with states not explored in the original plan.

The notion of "responsibly repairing" a plan refers to the process of making changes or adjustments to a plan in a way that minimises the negative impact on other agents or systems. In other words, an agent that repairs its plan responsibly is one that takes into consideration the potential consequences of its actions on other agents and aims to minimise any disruption or adverse effects. This could involve, for example, consulting with other agents or systems to openly book commitments in advance such as intra-agent repair approach [47], or using a specific metric or algorithm to determine the repair to make as a single agent (no booking or coordination needed) such as the metric presented in our work. Coining the notion of "responsibly repairing" as a formal definition would require further investigation and would be an important area for future research. Additionally, a formal definition would benefit from establishing clear criteria of what is considered responsible in different scenarios and settings.

## 8. Conclusion

Plan repair is crucial for any intelligent agent that operates in a dynamic environment. In action-sensitive distances such as the plan stability and the action distances, when one object changes in an action, the whole action is considered different. Consequently, once one object changes, action-sensitive

distances cannot further distinguish the quality of the plans, no matter whether the change is small or large. This paper presents a property called plan commitment to ensure a responsible repairing policy among agents that aims to minimise the negative impact on others and presents the arguments to support the claim that plan commitment is a valuable property when an agent may have made bookings to others. An implementation is presented based on TFLAP resulting in C-TFLAP planner. The empirical evaluation results support our claim as commitment repair can reduce the number of failures and the time-loss among agents. The results demonstrate that the suggested approach outperforms typical replanning and plan-stability repair for the mentioned purposes. The approach is domain-independent and agile and can be easily modified and extended.

Determining an appropriate threshold for the delay caused by a repair method in a particular agent, beyond which it would negatively impact the overall system performance (other agents), is a complex task that requires further research and investigation. In traditional multi-agent systems where agents coordinate and communicate their plans and goals openly, the threshold is determined by simple trade-offs such as the extra time spent by an agent versus the resulting time loss for other agents or the overall system's performance (total saved time in the community of agents). However, determining the threshold is more challenging in a community of agents with reduced communication and privacy. As we mentioned previously, the advantages of our method are that no coordination is required among agents, no communication is required after an agent generates its plan, and goals and plans are private and not shared among agents. In future work, we will continue our investigation for single-agent planning and repairing in a community of agents with reduced communication and privacy settings by estimating the threshold through historical data (case-based) or simulating the existence of another agent in the execution environment. By doing so, we can determine the expected impact of a delay on the overall system's performance and estimate the threshold of a disruption occurring in the worst case or on average.

## Acknowledgements

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors. We thank Prof. Juan Carlos Cortes Lopez from the Applied Mathematics department at the Polytechnic University of Valencia for providing valuable mathematical feedback on the plan commitment distance. The authors would like to thank the following colleagues for their constructive criticism of the manuscript: Louise Cooke, PhD., Emeritus Professor of Information and Knowledge Management, School of Business and Economics, Loughborough University, UK and Majd Alwan, PhD., SVP of Technology & Business Strategy, LeadingAge, Executive Director, Center for Aging Services Technologies (CAST), Washington, DC, USA.

## CRediT authorship contribution statement

Conceptualisation, MB. Investigation and methodology, MB, OS. Formalisation MB, OS, EO. Supervision, OS, EO. Writing of the original draft, MB, OS. Writing and editing of final version MB, OS, EO. All authors read and approved the final manuscript.

## Declaration of Competing Interest

The authors declare they have no conflict of interest.

## Data availability

Data can be made available on request.

## References

[1] Babli, M., Ibáñez-Ruiz, J., Sebastia, L., Tejero, A.G., Onaindia, E., 2016. An intelligent system for smart tourism simulation in a dynamic environment, in: Spyropoulos, C.D., Pierris, G., Tzortzis, G. (Eds.), Proc. of the 2nd Workshop on Artificial Intelligence and Internet of Things co-located with the 22nd European Conference on Artificial Intelligence (ECAI 2016), The Hague, Netherlands, August 30, 2016, CEUR-WS.org. pp. 15–22. URL: http://ceur-ws.org/Vol-1724/paper3.pdf.

[2] Babli, M., Onaindia, E., 2019. A Context-Aware Knowledge Acquisition for Planning Applications Using Ontologies, in: Proc. of the 33rd International Business Information Management Association Conference, IBIMA 2019: Education Excellence and Innovation Management through Vision 2020. Granada, Spain, pp. 3602–3614.

[3] Babli, M., Rincon, J.A., Onaindia, E., Carrascosa, C., Julian, V., 2021. Deliberative Context-Aware Ambient Intelligence System for Assisted Living Homes. Hum.-centric Comput. Inf. Sci. (HCIS) 11. doi:10.22967/HCIS.2021.11.019.

[4] Bäckström, C., Nebel, B., 1995. Complexity results for SAS+ planning. Comput. Intell. 11, 625–656. doi:10.1111/j.1467-8640.1995.tb00052.x.

[5] Bai, H., Hsu, D., Lee, W.S., 2014. Integrated perception and planning in the continuous space: A POMDP approach. Int. J. Robotics Res. 33, 1288–1302. doi:10.1177/0278364914528255.

[6] Bajo, J., Julián, V., Corchado, J.M., Carrascosa, C., de Paz, Y., Botti, V.J., de Paz, J.F., 2008. An execution time planner for the ARTIS agent architecture. Eng. Appl. Artif. Intell. 21, 769–784. doi:10.1016/j.engappai.2007.07.006.

[7] Bechon, P., Lesire, C., Barbier, M., 2020. Hybrid planning and distributed iterative repair for multi-robot missions with communication losses. Auton. Robots 44, 505–531. doi:10.1007/s10514-019-09869-w.

[8] Bidot, J., Schattenberg, B., Biundo, S., 2008. Plan Repair in Hybrid Planning, in: Dengel, A., Berns, K., Breuel, T.M., Bomarius, F., Roth-Berghofer, T. (Eds.), KI 2008: Advances in Artificial Intelligence, 31st Annual German Conference on AI, Springer. pp. 169–176. doi:10.1007/978-3-540-85845-4_21.

[9] Carreno, Y., Willners, J.S., Petillot, Y.R., Petrick, R.P., 2021. Situation-Aware Task Planning for Robust AUV Exploration in Extreme Environments, in: Proc. of the IJCAI Workshop on Robust and Reliable Autonomy in the Wild.

[10] Castellini, A., Marchesini, E., Farinelli, A., 2021. Partially Observable Monte Carlo Planning with state variable constraints for mobile robot navigation. Eng. Appl. Artif. Intell. 104, 104382. doi:10.1016/j.engappai.2021.104382.

[11] Cenamor, I., de la Rosa, T., Vallati, M., Fernández, F., Chrpa, L., 2018. TemPoRal: Temporal Portfolio Algorithm, in: 2018 International Planning Competition, Temporal Tack.

[12] Chen, C., Xu, R., Zhu, S., Li, Z., Jiang, H., 2020. RPRS: a reactive plan repair strategy for rapid response to plan failures of deep space missions. Acta Astronaut. 175, 155 – 162. doi:https://doi.org/10.1016/j.actaastro.2020.05.011.

[13] Chien, S.A., Morris, R., 2014. Space Applications of Artificial Intelligence. AI Mag. 35, 3–6. doi:10.1609/aimag.v35i4.2551.

[14] Cushing, W., Kambhampati, S., 2005. Replanning: A new perspective. Proc. of the International Conference on Automated Planning and Scheduling Monterey, USA , 13–16.

[15] Edelkamp, S., Hoffmann, J., 2004. PDDL2.2: The language for the classical part of the 4th international planning competition. 4th International Planning Competition (IPC), at ICAPS .

[16] Fikes, R., Hart, P.E., Nilsson, N.J., 1972. Learning and executing generalized robot plans. Artif. Intell. 3, 251–288. doi:10.1016/0004-3702(72)90051-3.

[17] Fox, M., Gerevini, A., Long, D., Serina, I., 2006. Plan Stability: Replanning versus Plan Repair, in: Long, D., Smith, S.F., Borrajo, D., McCluskey, L. (Eds.), Proc. of the Sixteenth International Conference on Automated Planning and Scheduling, ICAPS 2006, Cumbria, UK, June 6-10, 2006, AAAI. pp. 212–221. URL: http://www.aaai.org/Library/ICAPS/2006/icaps06-022.php.

[18] Furelos-Blanco, D., Jonsson, A., 2018. CP4TP: A Classical Planning for Temporal Planning Portfolio, in: 2018 International Planning Competition, Temporal Tack.

[19] Garrido, A., Onaindia, E., Sapena, O., 2008. Planning and scheduling in an e-learning environment. A constraint-programming-based approach. Eng. Appl. Artif. Intell. 21, 733–743. doi:https://doi.org/10.1016/j.engappai.2008.03.009. constraint Satisfaction Techniques for Planning and Scheduling Problems.

[20] Gerevini, A., Serina, I., 2000. Fast Plan Adaptation through Planning Graphs: Local and Systematic Search Techniques, in: Chien, S.A., Kambhampati, S., Knoblock, C.A. (Eds.), Proc. of the Fifth International Conference on Artificial Intelligence Planning Systems, AAAI, Breckenridge, CO, USA. pp. 112–121. URL: http://www.aaai.org/Library/AIPS/2000/aips00-012.php.

[21] Ghallab, M., Nau, D., Traverso, P., 2014. The actor's view of automated planning and acting: A position paper. Artif. Intell. 208, 1–17.

[22] Ghallab, M., Nau, D.S., Traverso, P., 2004. Automated planning - theory and practice. Elsevier.

[23] Goldman, R., Kuter, U., Freedman, R., 2020. Stable Plan Repair for State-Space HTN Planning, in: The 3rd ICAPS Workshop on Hierarchical Planning (HPlan 2020), pp. 27–35.

[24] Hammond, K.J., 1990. Explaining and Repairing Plans That Fail. Artif. Intell. 45, 173–228. doi:10.1016/0004-3702(90)90040-7.

[25] Helmert, M., 2006. The fast downward planning system. JAIR 26, 191–246. URL: https://doi.org/10.1613/jair.1705, doi:10.1613/jair.1705.

[26] Hoffmann, J., 2003. The Metric-FF Planning System: Translating "Ignoring Delete Lists" to Numeric State Variables. J. Artif. Intell. Res. 20, 291–341. doi:10.1613/jair.1144.

[27] Hoffmann, J., Brafman, R.I., 2005. Contingent Planning via Heuristic Forward Search with Implicit Belief States, in: Biundo, S., Myers, K.L., Rajan, K. (Eds.), Proc. of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS 2005), AAAI, Monterey, California, USA. pp. 71–80. URL: http://www.aaai.org/Library/ICAPS/2005/icaps05-008.php.

[28] Ingrand, F., Ghallab, M., 2017. Deliberation for autonomous robots: A survey. Artif. Intell. 247, 10–44. doi:10.1016/j.artint.2014.11.003.

[29] Jiang, Y., Yedidsion, H., Zhang, S., Sharon, G., Stone, P., 2019. Multi-robot planning with conflicts and synergies. Auton. Robots 43, 2011–2032. doi:10.1007/s10514-019-09848-1.

[30] Joslin, D., Pollack, M.E., 1994. Least-cost flaw repair: A plan refinement strategy for partial-order planning, in: Hayes-Roth, B., Korf, R.E. (Eds.), Proceedings of the 12th National Conference on Artificial Intelligence, Seattle, WA, USA, July 31 - August 4, 1994, Volume 2, AAAI Press / The MIT Press. pp. 1004–1009.

[31] Karpas, E., Magazzeni, D., 2020. Automated Planning for Robotics. Annu. rev. control robot. auton. syst. 3, 417–439. doi:10.1146/annurev-control-082619-100135.

[32] Koenig, S., Likhachev, M., 2005. Fast replanning for navigation in unknown terrain. IEEE Trans. Robot. 21, 354–363. doi:10.1109/TRO.2004.838026.

[33] Komenda, A., Novák, P., Pěchouček, M., 2014. Domain-independent multi-agent plan repair. J. Netw. Comput. Appl. 37, 76–88. doi:10.1016/j.jnca.2012.12.011.

[34] van der Krogt, R., de Weerdt, M., 2005. Plan Repair as an Extension of Planning, in: Biundo, S., Myers, K.L., Rajan, K. (Eds.), Proc. of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS 2005), Monterey, California, USA, AAAI. pp. 161–170. URL: http://www.aaai.org/Library/ICAPS/2005/icaps05-017.php.

[35] Lima, O., Cashmore, M., Magazzeni, D., Micheli, A., Ventura, R., 2020. Robust Plan Execution with Unexpected Observations. CoRR abs/2003.09401. URL: https://arxiv.org/abs/2003.09401.

[36] Mohalik, S.K., Jayaraman, M.B., Badrinath, R., Feljan, A.V., 2018. HIPR: An Architecture for Iterative Plan Repair in Hierarchical Multi-agent Systems. J. Comput. 13, 351–359. doi:10.17706/jcp.13.3.351-359.

[37] Nebel, B., Koehler, J., 1995. Plan Reuse Versus Plan Generation: A Theoretical and Empirical Analysis. Artif. Intell. 76, 427–454. doi:10.1016/0004-3702(94)00082-C.

[38] Nguyen, T.A., Do, M.B., Gerevini, A., Serina, I., Srivastava, B., Kambhampati, S., 2012. Generating diverse plans to handle unknown and partially known user preferences. Artif. Intell. 190, 1–31. doi:10.1016/j.artint.2012.05.005.

[39] Ong, S.C.W., Png, S.W., Hsu, D., Lee, W.S., 2010. Planning under uncertainty for robotic tasks with mixed observability. Int. J. Robot. Res. 29, 1053–1068. doi:10.1177/0278364910369861.

[40] Palacios, H., Geffner, H., 2009. Compiling Uncertainty Away in Conformant Planning Problems with Bounded Width. J. Artif. Intell. Res. 35, 623–675. doi:10.1613/jair.2708.

[41] Richter, S., Westphal, M., 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. JAIR 39, 127–177. URL: https://doi.org/10.1613/jair.2972, doi:10.1613/jair.2972.

[42] Rodríguez, S., Julián, V., Bajo, J., Carrascosa, C., Botti, V.J., Corchado, J.M., 2011. Agent-based virtual organization architecture. Eng. Appl. Artif. Intell. 24, 895–910. doi:10.1016/j.engappai.2011.02.003.

[43] Sapena, O., Marzal, E., Onaindia, E., 2018. TFLAP: a temporal forward partial-order planner. Technical Report. 2018 International Planning Competition, Temporal Tack. URL: https://bitbucket.org/ipc2018-temporal/team2/src/master/.

[44] Sapena, O., Onaindia, E., 2008. Planning in highly dynamic environments: an anytime approach for planning under time constraints. Appl. Intell. 29, 90–109. URL: https://doi.org/10.1007/s10489-007-0083-x, doi:10.1007/s10489-007-0083-x.

[45] Spaan, M.T.J., Veiga, T.S., Lima, P.U., 2015. Decision-theoretic planning under uncertainty with information rewards for active cooperative perception. Auton. Agents Multi-Agent Syst. 29, 1157–1185. doi:10.1007/s10458-014-9279-8.

[46] Srivastava, B., Nguyen, T.A., Gerevini, A., Kambhampati, S., Do, M.B., Serina, I., 2007. Domain Independent Approaches for Finding Diverse Plans, in: IJCAI 2007, Proc. of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, pp. 2016–2022.

[47] Talamadupula, K., Smith, D.E., Cushing, W., Kambhampati, S., 2013. A theory of intra-agent replanning. Technical Report. Arizona State Univ Tempe Dept of Computer Science and Engineering.

[48] Vendrell, E., Mellado, M., Crespo, A., 2001. Robot planning and replanning using decomposition, abstraction, deduction, and prediction. Eng. Appl. Artif. Intell. 14, 505–518. doi:10.1016/S0952-1976(01)00027-6.

[49] Warfield, I., Hogg, C., Lee-Urban, S., Muñoz-Avila, H., 2007. Adaptation of Hierarchical Task Network Plans, in: Wilson, D., Sutcliffe, G. (Eds.), Proc. of the Twentieth International Florida Artificial Intelligence Research Society Conference, AAAI Press, Key West, Florida, USA. pp. 429–434. URL: http://www.aaai.org/Library/FLAIRS/2007/flairs07-084.php.

[50] Zweben, M., Davis, E., Daun, B., Deale, M.J., 1993. Scheduling and rescheduling with iterative repair. IEEE Trans. Syst. Man Cybern. 23, 1588–1596. doi:10.1109/21.257756.