Automatic Generation of Flexible Plans via Diverse Temporal Planning

Yotam Amitai[⋆] and **Ayal Taitler**[⋆] and **Erez Karpas**[†] and **Per-Olof Gutman**[‡]
[⋆]Technion Autonomous Systems Program
[†]Faculty of Industrial Engineering and Management
[‡]Faculty of Civil and Environmental Engineering
Technion Israel Institute of Technology
Haifa, Israel 32000

Abstract

Robots operating in the real world must deal with uncertainty, be it due to working with humans who are unpredictable, or simply because they must operate in a dynamic environment. Ignoring the uncertainty could be dangerous, while accounting for all possible outcomes, as in contingent planning, is often computationally infeasible. One possibility, which lies between ignoring the uncertainty completely and addressing it completely is to use flexible plans with choice, formulated as Temporal Planning Networks (TPNs). This approach has been successfully demonstrated to work in human-robot teaming using the Pike executive, an online executive that unifies intent recognition and plan adaptation. However, one of the main challenges to using Pike is the need to manually specify the TPN. In this paper, we address this challenge by describing a technique for automatically synthesizing a TPN which covers multiple possible executions for a given temporal planning problem specified in PDDL 2.1. Our approach starts by using a diverse planner to generate multiple plans, and then merges them into a single TPN. As there were no available diverse planners for temporal planning, we first show how to adapt existing diverse planning based on top-k planning to the temporal setting. We then show how to merge the diverse plans into a single TPN using constraint optimization. Finally, an empirical evaluation on a set of IPC benchmarks shows that our approach scales well, and generates TPNs which can generalize the set of plans they are generated from.

Introduction

Robots operating in the real world must deal with uncertainty. The uncertainty could stem from operating in a dynamic environment, or from the need to work together with a human. There are several approaches to dealing with this uncertainty. One common approach is to "determinize and replan" (Yoon, Fern, and Givan 2007), that is, come up with a single plan which might solve the problem, start executing it, and if anything goes wrong — replan. Although this approach often works, it preforms poorly on problems which are "probabilistically interesting" (Little and Thiebaux 2007), such as problems with avoidable dead-ends. This is even more problematic in the case of human-robot teamwork, as the new plan will need to be communicated to the human every time replanning occurs. Another approach, on the other extreme, is to account for all possible uncertainty and come up with a contingent plan (e.g., (Hoffmann and Brafman 2005)), which dictates what must occur in response to any possible uncertain outcome or disturbance. Unfortunately, offline contingent planning is a computationally challenging problem, and this approach does not scale well. We advocate taking a middle-ground approach between the two aforementioned options by using flexible plans with choices. Specifically, we advocate using Temporal Planning Networks (Kim, Williams, and Abramson 2001), referred to as TPNs, to address only *some* of the uncertainty.

TPNs were originally designed to control robotic space explorers. More recently, they have been demonstrated in the context of human-robot teamwork in an airplane manufacturing scenario (Burke et al. 2014), and in controlling micro-UAVs (Timmons et al. 2015). The Pike executive (Levine and Williams 2018), which was used in both demonstrations mentioned above, executes TPNs by making choices for the robots, while monitoring execution and dispatching actions at the appropriate times. While the effectiveness of Pike and TPNs has been shown, there is still a barrier to using TPNs due to the fact that generating TPNs has so far only been done by manually encoding them. Although it is possible to compile a control program written in the Reactive Model Planning Language (RMPL) (Ingham, Ragno, and Williams 2001) into a TPN, the control program must still be manually written.

In this paper, we describe the first method for the automatic generation of a TPN, given a specification of a temporal planning problem in standard PDDL 2.1 (Fox and Long 2003).

We describe an approach based on merging diverse plans. None of the existing diverse planners (Bryce 2014; Nguyen et al. 2012; Srivastava et al. 2007; Katz and Sohrabi 2019) handle temporal planning so we first describe a diverse temporal planner, based on adapting the top-k based diverse planner (Katz and Sohrabi 2019) to the temporal setting. Then, we identify sets of points along different plans which can be merged together. Lastly, we choose which of these points should be merged by modeling and solving a Constraint Optimization Problem (COP). The result is a single TPN encompassing all the generated plans. An empirical evaluation on a set of IPC benchmarks shows that our approach can quickly generate TPNs, even for large problems.

We attempt to create the smallest possible TPN by maximizing the number of points our process merges along the different plans. Smaller TPNs encode all the original plans compactly, reduce complexity by lowering the number of elements in the TPN and intuitively, are easier to communicate to a human counterpart.

Background

We start by reviewing the necessary background on temporal planning, TPNs and diverse planning.

We assume we have a planning problem modeled in the propositional subset of PDDL 2.1 (Fox and Long 2003), that is, given by a tuple $\Pi = \langle F, A, I, G \rangle$, where:

- F: The set of Boolean propositions, s.t S, the set of all possible states is then all the subsets of F, meaning $\mid S \mid = 2^F$ states.
- A: The set of durative actions.
  Each durative action $a \in A$ has a duration $\text{dur}(a) \in [\text{dur}_{min}(a), \text{dur}_{max}(a)]$ and is described by $a = \langle \text{pre}_{\vdash}(a), \text{eff}_{\vdash}(a), \text{inv}(a), \text{pre}_{\dashv}(a), \text{eff}_{\dashv}(a) \rangle$, where:
  - Minimum duration $\text{dur}_{min}(a)$ and maximum duration $\text{dur}_{max}(a)$, where $0 \le \text{dur}_{min}(a) \le \text{dur}_{max}(a)$,
  - Start condition $\text{pre}_{\vdash}(a) \subseteq F$ (respectively, end condition $\text{pre}_{\dashv}(a) \subseteq F$), must hold when durative action a starts (respectively, ends),
  - Start effect $\text{eff}_{\vdash}(a)$ (respectively, end effect $\text{eff}_{\dashv}(a)$), occurs when durative action a starts (respectively, ends). The effects specify which propositions in F become true (add effects), and which become false (delete effects), and
  - Invariant condition $\text{inv}(a) \subseteq F$ which must hold during the whole execution of a.
- $I \subseteq F$: The initial state, specifying exactly which propositions in F are true at time zero.
- $G \subseteq F$: The goal, which propositions we wish to be true at the end of plan execution.

A solution to a temporal planning task is a schedule τ which is a sequence of triples $\langle a, t, d \rangle$, where $a \in A$ is a durative action, $t \in \mathbb{R}^{0+}$ is the time when the durative action a is started, and $d \in \left[\text{dur}_{min}(a), \text{dur}_{max}(a)\right]$ is the duration chosen for a. Similarly to plan validity, we call a solution's schedule τ valid, if the assignment for the time duration for each durative action respects all the temporal constraints. A schedule can be seen as a sequence of Instantaneous Happenings (IH) occurring at least ϵ time units apart (Fox and Long 2003), which occur when a durative action starts and when a durative action ends. Specifically, for each triple $\langle a, t, d \rangle$ in the schedule, we have an IH (also called snap action), $a_{\vdash}$ occurring at time t (requiring $\text{pre}_{\vdash}(a)$ to hold ϵ time before t, and applying the effects $\text{eff}_{\vdash}(a)$ right at t), and an ending IH $a_{\dashv}$ at time $t+d$ (requiring $\text{pre}_{\dashv}(a)$ to hold ϵ before $t + d$, and applying the effects $\text{eff}_{\dashv}(a)$ at time $t + d$). Thus, a solution for a temporal planning problem can be viewed as a sequence of such happenings, each with its time stamp, e.g. $\langle (a^1_{\vdash}, t_1), ..., (a^i_{\vdash}, t_i), ..., (a^1_{\dashv}, t_j), ..., (a^i_{\dashv}, t_k), ... \rangle$. In order for a schedule to be *feasible*, we also require the invariant condition $\text{inv}(a)$ to hold over the open interval between t and $t + d$. Finally, given a state s and a sequence of IHs Σ (without timestamps), we will denote the state resulting from applying Σ from s by $T(s, \Sigma)$. The schedule τ is valid if $G \subseteq T(I, \Sigma)$, where Σ is the sequence of IHs in τ, ordered by the time they occur – that is, we require the goal to hold after all actions have completed. In this work we make us of OPTIC (Benton, Coles, and Coles 2012) as the underlying temporal solver. In this work we make use of diverse planners, which generate dissimilar solutions for the same planning task. Various approaches to diverse planning have been proposed (Bryce 2014; Nguyen et al. 2012; Srivastava et al. 2007). Reviewing the extensive literature on diverse planning is out of scope of this paper, see (Roberts, Howe, and Ray 2014) for such a review. In this paper, we will extend the diverse planner (Katz and Sohrabi 2019), which uses a top-k planner (Katz et al. 2018) to generate the top-k best solutions to the planning task.

We are now ready to define Temporal Planning Networks (TPNs), a formalism for representing flexible plans with choices. A TPN is an extension to the Simple Temporal Network (Dechter, Meiri, and Pearl 1991), which adds decision nodes and labels on constraints conditioned on these decisions (also referred to as choices). We shall build upon (but simplify) the definition supplied in (Levine and Williams 2018) for a Temporal Planning Network under Uncertainty, as this is the formalism Pike expects as input. The main difference between a TPN and a TPNU being the mapping of the choice variables to groups of controllable and uncontrollable choices. Formally a TPNU is a tuple $\langle V, \mathcal{E}, C, \mathbb{A} \rangle$, where:

- V: The set of decision variables. Decision variables are partitioned into two groups $V = V_C \cup V_U$. Each $v \in V$ is a discrete variable with a finite domain $Domain(v)$. V_C are *controllable* choices, decidable by the executive at run time. V_U are the *uncontrollable* choice variables, whose decisions are determined by the human or nature, rather than the executive.
- $\mathcal{E}$: The set of notable time points (Events). Each $e \in \mathcal{E}$ is associated with a conjunction of choice variable assignments φ_e. Events can be seen as correlated to the underlying PDDL states of the original task.
- C: The set of temporal constraints. Each $c \in C$ is a tuple $\langle e_s, e_f, l, u, \varphi_c \rangle$ where e_s is the *start* event, e_f is the *finish* event, φ_c is a conjunction of choice variable assignments and $l, u \in \mathbb{R}$ represent temporal upper and lower bounds s.t. $\varphi_c \implies (l \le e_f - e_s \le u)$.
- $\mathbb{A}$: The set of activities. An activity $a \in \mathbb{A}$ is a tuple $\langle c, \alpha \rangle$ where $c \in C$ is a temporal constraint, and α is an action that will be executed online. With $c = \langle e_s, e_f, l, u, \varphi_c \rangle$, action α starts when e_s is scheduled, and terminates when e_f is scheduled. We require that $l > 0$ Activities are related to the durative actions of the underlying PDDL domain.

We note that in this work we only generate TPNs, i.e. no uncontrollable choices are assigned. The task of determining which choices are uncontrollable is a subject for future work.

Pike is an online plan executive for human-robot teamwork that quickly adapts and infers intent based on the preconditions of actions in the plan, temporal constraints, unanticipated disturbances, and choices made previously (by either robot or human). It takes as input a flexible plan with choices defined using the TPNU formalism. Using Pike to control robots results in a mixed initiative execution in which humans and robots simultaneously work and adapt to each other to accomplish a task.

Automatic Generation of TPNs

Now that we have supplied the relevant motivation and background for the task at hand, we can further define the problem which we wish to solve. Given a Temporal Planning task $\Pi = \langle F, A, I, G \rangle$ as input, we wish to generate a Temporal Planning Network (TPN) which represents multiple dissimilar plans that solve the task at hand.

We note here that our work focuses primarily on the TPN time points $\mathcal{E}$ as these are the elements we wish to minimize in order to achieve the smallest TPN. Merging time points can create decision variables. Different combinations of such decisions might lead to valid or invalid plans, thus, other possible optimization objectives could include generating TPNs with a high number of decision variable combinations, or which lead to a high number of valid plans. We leave optimizing these measures for future work.

The TPN executive (Pike) incorporates the ability to avoid making combinations of choices that lead to infeasible paths and so we do not trouble ourselves with addressing how these merges may affect C.

We first describe our devised approach to diverse temporal planning, then go on to tackle the question of how to merge multiple solutions into a single representation (TPN), and lastly we showcase the results of our technique on the latest IPC domains.

Diverse Temporal Planning

Our approach to diverse temporal planning builds upon the top-k approach (Katz and Sohrabi 2019). The main challenge we address here is that temporal plans are not a sequence of instantaneous happenings, but rather a schedule, and thus the top-k approach does not apply directly. Therefore, we first define the *temporal plan skeleton* of a solution to a temporal planning task.

Definition 1 (Temporal Plan Skeleton). *Given a planning task $\Pi = \langle F, A, I, G \rangle$ and a solution τ. The temporal plan skeleton (TPS) π is the sequence of the IHs in τ (without their time stamps).*

Given a TPS π and an IH $a \in \pi$. The TPS suffix of π from a, denoted Σ_a^π, is the sequence of IHs in π from right after a occurs (excluding a) until the end of the TPS.

The objective of diverse temporal planning is to find dissimilar solutions to the temporal planning task Π. We argue that two different plans, with the same TPS, and which

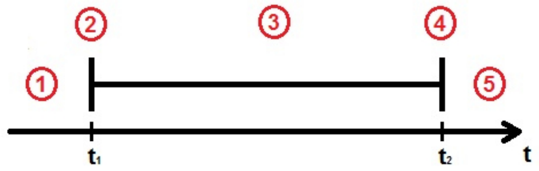


Figure 1: All the possible points to deviate from TPS π, from the point of view of the i^{th} action in π, a_i.

vary only in their time stamps, are not very different. Specifically, for the purposes of merging these plans into a TPN, they are not different at all, as the TPN executive will make the scheduling decisions. Thus, we define two plans to be different if and only if they have a different TPSs.

The diverse top-k planning approach (Katz and Sohrabi 2019) works iteratively by calling a planner to obtain a solution τ, then creating a modified planning task which eliminates the solution τ, calling the planner again, and so forth. Thus, to apply this approach to temporal planning, we create a *temporal plan elimination formulation*, which takes as input a temporal planning task Π and a solution τ, and creates a modified temporal planning task Π' which eliminates all solutions which share the same TPS as τ (while all other solutions remain valid).

The main technical challenge here is that temporal planning is performed with durative actions, while the *plan forbidding reformulation* (Katz et al. 2018) works on IHs (as in classical planning). Furthermore, the plan forbidding reformulation is based on detecting when the current candidate plan deviates from the plan to forbid, which is simpler for classical planning.

To overcome this challenge, we think of each durative action a as two IHs; $a_\vdash$ at the start and $a_\dashv$ at the end. Note that a TPS is determined by the order of the IHs, similarly to a classical plan. Thus, if we could somehow plan with IHs (while still respecting action durations, invariant conditions, and temporal constraints) we could use the classical planning approach directly. While this is not possible, we can think of a durative action as a pair of two IHs, and look at five different points where a durative action might deviate from the given TPS π. These are illustrated in Figure 1, and correspond to the five cases described below.

Case 1: We have already deviated from π before a started.

Case 2: We have followed π until now, but action a is different than the i^{th} action in π.

Case 3: Action a starts according to π, but between $a_\vdash$ and $a_\dashv$, another instantaneous event occurs, which deviates from π.

Case 4: Action a starts according to π, but the end event $a_\dashv$ is not according to π, i.e., the end event is not according to the sequence. This occurs when some other event should have occurred before $a_\dashv$.

Case 5: Action a starts and ends according to π. This case is when π is being followed, and a future action will deviate.

Having described these 5 cases, we can now describe our *temporal plan elimination formulation*. This formulation has 6 different versions of each durative action that takes part in the plan: one for each of the above five cases, and one for actions which do not appear in π. Also, similarly to the top-k approach (Katz et al. 2018), we introduce new proposition to encode deviation from π. Specifically, for a given TPS with n IHs, we use $2n + 2$ propositions: $2n + 1$ to encode the sequence π, and another for representing whether we have already deviated. Note, that only durative action participating in the plan to forbid will be multiplied, and not the entire space of durative actions.

We now formally describe our formulation. Let $\Pi = \langle F, A, I, G\rangle$ be a planning task, and $\tau = \langle a_1, t_1, d_1\rangle, ..., \langle a_n, t_n, d_n\rangle$ be some temporal plan with a corresponding TPS π, where i and i' are the time indexes of $a_\vdash$ and $a_\dashv$ appropriately in π. The planning task $\Pi' = \langle F', A', I', G'\rangle$ is defined as follows:

- $F' = F \cup \{p, p_0, ..., p_{2n}\}$,
- $A' = \{a^0 \mid a \in A, a \notin \tau\} \cup \{a^1, a^2, a^3, a^4, a^5 \mid a \in \tau\}$ where:

$$\begin{aligned}
a^0 =& \langle \text{pre}_\vdash(a), \text{eff}_\vdash(a) \cup \{p\}, \text{inv}(a), \text{pre}_\dashv(a), \text{eff}_\dashv(a)\rangle \\
a^1 =& \langle \text{pre}_\vdash(a) \cup \{p\}, \text{eff}_\vdash(a), \text{inv}(a), \text{pre}_\dashv(a), \text{eff}_\dashv(a)\rangle \\
a^2 =& \langle \text{pre}_\vdash(a) \cup \{\neg p, \neg p_{i-1}\}, \text{eff}_\vdash(a) \land p, \text{inv}(a), \\
& \text{pre}_\dashv(a), \text{eff}_\dashv(a)\rangle \\
a^3 =& \langle \text{pre}_\vdash(a) \cup \{\neg p, p_{i-1}\}, \text{eff}_\vdash(a) \land \neg p_{i-1} \land p_i, \\
& \text{inv}(a), \text{pre}_\dashv(a) \cup \{p\}, \text{eff}_\dashv(a)\rangle \\
a^4 =& \langle \text{pre}_\vdash(a) \cup \{\neg p, p_{i-1}\}, \text{eff}_\vdash(a) \land \neg p_{i-1} \land p_i, \\
& \text{inv}(a), \text{pre}_\dashv(a) \cup \{\neg p, \neg p_{i'-1}\}, \text{eff}_\dashv(a) \land p\rangle \\
a^5 =& \langle \text{pre}_\vdash(a) \cup \{\neg p, p_{i-1}\}, \text{eff}_\vdash(a) \land \neg p_{i-1} \land p_i, \\
& \text{inv}(a), \text{pre}_\dashv(a) \cup \{\neg p, p_{i'-1}\}, \\
& \text{eff}_\dashv(a) \land \neg p_{i'-1} \land p_{i'}\rangle
\end{aligned}$$

- $I' = I \cup \{p_0\}$
- $G' = G \cup \{p\}$

Explaining the reformulation. For ease of presentation, we abuse notation and say that a temporal action a is along π, when $a_\vdash, a_\dashv \in \pi$ with indexes i, i'. The proposition p represents a deviation from π, so it starts as false, and becomes true when the sequence of actions applied is not a prefix of π. Once the value p is achieved, it remains true. p is also part of the new goal, G', as the objective here is to find a deviation from π.

Propositions $p_0, ..., p_{2n}$ encode the progress along the TPS π, before deviating from it. Actions a^0 are the activities that do not appear in π, thus automatically indicate deviation from π and achieve p. The actions $a^1, ..., a^4$ are copies of actions in π, corresponding to cases 1...4 above. a^5 are copies of actions along π, these actions are responsible for following the sequence π and are applicable only while the sequence is still followed, i.e. p is false. Note that in all five cases when an action along π has more than one instance, each instance is treated as a different action with a different corresponding p_i variable indicating its position in the sequence. The convention in the reformulation is that the preconditions are sets which requirements are added to, and effects are sets comprised of delete and add effects, thus the conjunction between delete and add effects of the auxiliary variables.

Theorem 1. *Let Π be a temporal planning task and τ it's temporal solution plan with temporal skeleton π. The task Π' is a plan elimination reformulation of Π and π.*

Proof. First we define the set of events $O = \{A_\vdash, A_\dashv\}$ as the set containing all the start and end events of durative actions in A. We also define 4 additional sets of events: o^0 events that are not in π, o^1 events that are in π but the skeleton has already been broken, o^2 events that are in π and are the first occurrence of event not in the right sequence in π, and finally o^3 events that are in π and in the right order, i.e for each of the action in the elimination formulation it holds that for $a \in a^0$, $a = \{a_\vdash, a_\dashv\} \to a_\vdash \in o^0, a_\dashv \in o^0$. For $a \in a^1$, $a = \{a_\vdash, a_\dashv\} \to a_\vdash \in o^1, a_\dashv \in o^1$. For $a \in a^2$, $a = \{a_\vdash, a_\dashv\} \to a_\vdash \in o^2, a_\dashv \in o^1$. For $a \in a^3$, $a = \{a_\vdash, a_\dashv\} \to a_\vdash \in o^3, a_\dashv \in o^1$. For $a \in a^4$, $a = \{a_\vdash, a_\dashv\} \to a_\vdash \in o^3, a_\dashv \in o^2$. And finally $a \in a^5$, $a = \{a_\vdash, a_\dashv\} \to a_\vdash \in o^3, a_\dashv \in o^3$.

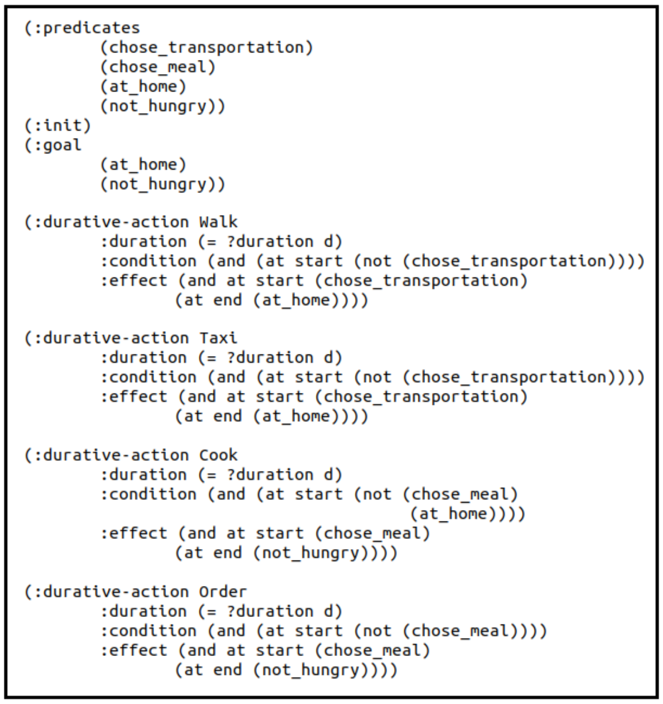
For the first direction, we show that the set of solutions of the reformulation, e.g., the set of solutions of Π' is included in the set of solutions of Π. Let $\mathcal{R} : O' \to O$ be the mapping defined by $\mathcal{R}(o^0) = a$ and $\mathcal{R}(a^1) = \mathcal{R}(a^2) = \mathcal{R}(a^3) = o$. Note that Π' is restricted to the propositions V equal to those in task Π, modulo the three equal five equal instances of events in π and the five equal instances of the actions in τ. Thus, for each skeleton π' for Π', $\mathcal{R}(\pi)$ is a skeleton for Π.

For the second direction, we show that every solution of Π is included in Π', excluding π. Since for each event $o \in \pi$ at most one event of the events $o^j,\ j = 1, 2, 3$ corresponding to actions $a^i,\ i = 1, ..., 5$ is applicable in each state s of Π', given a sequence of events α applicable in the initial state of Π it can be mapped to an applicable sequence α' in the initial state of Π', such that $\mathcal{R}(\alpha') = \alpha$, by choosing in each state the relevant representative out of $o^j,\ j = 0, ..., 3$ corresponding to $a^i,\ a = 0, ..., 5$. I.e., $\mathcal{R}$ is restricted to applicable events sequences in the initial state, and $\mathcal{R}$ is invertible, we denote its inverse mapping by $\mathcal{R}^{-1}$. Now, for a given π, let $\pi' = \mathcal{R}^{-1}(\pi)$ be the inverse mapping of the plan $\pi = o_1, ..., o_{2N}$ for Π. Then $\pi' = o_1^3, ..., o_{2N}^3$ induced by $\tau' = a_1^5, ..., a_n^5$ by the definition of the set a^5. At each step p is False, thus τ' is not a plan for Π', consequently π' is not a skeleton for Π'. Now, let ρ be a plan for Π with a corresponding skeleton α, such that $\alpha \neq \pi$. Let o be the first event in α that differs from corresponding event in π, with index i. So the prefix of events $\alpha' = o_1, ..., o_{i-1}$ is shared both by pi and α. Thus we have $\mathcal{R}^{-1}(\alpha') = o_1^3, ..., o_{i-1}^3$, and since $o \neq o_i^3$ the next event on $\mathcal{R}(\alpha)$ will not be o_i^3. If $o \in \pi$, it will be of o^2, and either of a^2 or a^4, if its $a_\vdash$ or $a_\dashv$, otherwise it will be of o^0. In either case it will set the value of p to True. Consequently, all the following events o' are mapped to either o^1 or o^0 with corresponds to actions a^0, a^2 and a^3, and the preconditions of these actions are restricted to $p = True$, and thus the skeleton $\mathcal{R}^{-1}(\alpha)$ achieves the goal values on S, V and thus is a plan.

Last thing to show is the applicability of o^2 in the state $s_i' := [o_1^3, ..., o_{i-1}^3]$ for $o \in \pi$ such that $o \neq o_i$. By default the preconditions of o holds in s_i'. Further, since at the initial state $\neg p,\ p_0, \neg p_j \quad j = 1, ..., 2N$, after applying $o_1^3, ..., o_{i-1}^3$ we have $s_i'[p] = True,\ s_i'[p_j] = False\ 0 \leq j \leq 2N\ j = i$ and $s_i'[p_i] = True$. Since $o \neq o_i$, for each $1 \leq j \leq$ such that $o = o_j$, we have $s_i'[p_{i-1} = False$, and thus, event from o^2 is applicable in s_i'. $\square$

Example 1 (Get Home & Eat). *As an example for the use of the theorem we will look at a Get Home & Eat problem where after a long day at work, our agent needs to get home and eat dinner. Our domain has four durative actions:* Walk, Taxi, Cook *and* Order. *The problem is such that some actions can be concurrent, the initial state is empty, and at the goal our agent is at home and ,having eaten his meal, not hungry.*

The domain and problem content for this example is as follows:



```
(:predicates
        (chose_transportation)
        (chose_meal)
        (at_home)
        (not_hungry))
(:init)
(:goal
        (at_home)
        (not_hungry))

(:durative-action Walk
        :duration (= ?duration d)
        :condition (and (at start (not (chose_transportation))))
        :effect (and at start (chose_transportation)
                (at end (at_home))))

(:durative-action Taxi
        :duration (= ?duration d)
        :condition (and (at start (not (chose_transportation))))
        :effect (and at start (chose_transportation)
                (at end (at_home))))

(:durative-action Cook
        :duration (= ?duration d)
        :condition (and (at start (not (chose_meal)
                                        (at_home))))
        :effect (and at start (chose_meal)
                (at end (not_hungry))))

(:durative-action Order
        :duration (= ?duration d)
        :condition (and (at start (not (chose_meal))))
        :effect (and at start (chose_meal)
                (at end (not_hungry))))
```

Figure 2: PDDL example domain

A feasible solution for this planning task is the following tuple of triplets: $(\langle Walk,\ 0,\ d\rangle, \langle Order,\ \epsilon,\ d\rangle)$ where the appropriate order of IHs or skeleton is: $\langle Walk_{\vdash},\ Order_{\vdash}, Walk_{\dashv}, Order_{\dashv}\rangle$. Note that the durative actions $Taxi$ and $Cook$ do not participate in this solution plan.

Thus, in the solution plan skeleton there are four IHs. the compilation will add six new predicates, five denoting the order of the plan and one for deviating from the plan, i.e. the predicates set is now: $\{chose\_transportation,\ chose\_meal,\ at\_home,\ not\_hungry,\ p,\ p_0,\ p_1,\ p_2,\ p_3,\ p_4\}$. The initial state now includes p_0 to denote that nothing has happened yet and it is still possible to find the original plan. The goal includes in addition p to denote deviation from the plan. The durative action $make\_tea$ does not participate is the plan, and thus becomes the a_0 set of actions, i.e., p is added to the *at start* effects to denote it is a new component in the plan. Now, each of the action participating in the plan is compiled into five different actions, denoting the different possibilities of orders of the IHs. For example action $Walk$ is comprised of the $Walk_{\vdash}$ which is the first IH in the skeleton thus if it is executed as the first action then p_0 is true and a start effect is to switch it to false and switch to true p_1. Similarly $Walk_{\dashv}$ is the third IH and if p_2 is true then the original plan is still followed and it switched to false, and p_3 is switched to true. This is an action of set a^1, if p_2 is false then the order of the original plan was broken and there is no need to turn p_3 on, and p can be switched to true to indicate the change. Like so, all the five actions are built and in similar manner it is done also to the other actions participating in the original plan, in this case the $Order$ action.

We note that in principle, if there were any useless actions (i.e. which don't contribute to the solution), then it would be possible that the new solution would not be very different from the original one. However, for such scenarios we take advantage of OPTIC's search method which prunes such useless actions in advance.

Theorem 2. *Let* Π *be a temporal planning task and* τ *a solution with TPS* π*. The task* Π' *is a plan elimination reformulation of* Π *and* π*.*

Merging Diverse Plans into a TPN

Now that we have obtained a set of diverse TPSs $\{\pi_1 \ldots \pi_k\}$ for our input planning task Π, we would like to merge these into a single TPN, that compactly encodes all generated solutions and possibly more.

The naive merging approach for generating a TPN would be to create a single decision variable $v \in V$ corresponding to a choice between the k obtained solutions. The structure of the TPN would then be a single decision at the start, and then the k constituent plans running in parallel up until they merge at the end of their path when reaching the terminal state which is shared. Of course, this approach defeats the purpose of having a flexible plan with choices.

The technique we describe here starts with the aforementioned naive TPN, but then looks for opportunities to merge additional time points. It is convenient to think of a TPN as a graph where time points are nodes connected by constraints (edges). By merging time points, additional solutions can be created, as demonstrated by the example in Figure 3.

The two constituent plans, shown at the top, are to walk home and then order in, or to take a taxi home and then cook. However, by merging the middle time point of these two solution paths, we obtain the TPN shown below, which yields 2 new solutions: walking home and then cooking, and taking a taxi home and then ordering in.

Next, we describe how we choose which time points from the naive TPN to merge.

Merging Time Points in the TPN

While it is theoretically possible to merge any two time points in the TPN, it is likely a bad idea to merge two random time points. Thus, we identify criteria for when it "makes sense" to consider merging two time points.

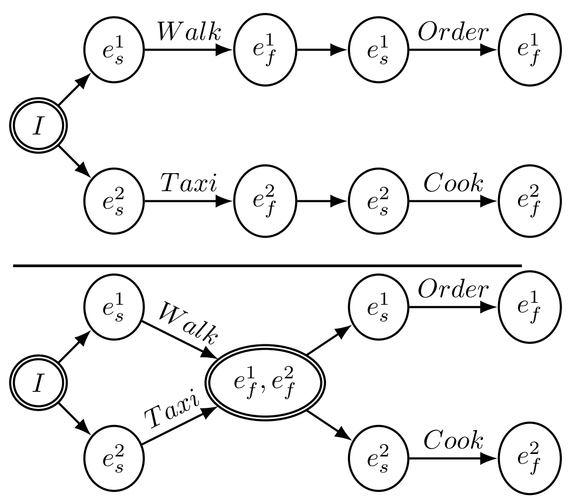


Figure 3: Additional paths created as a result of a merge. Choice time points are depicted with double circles. **Top:** Two separate plans, number of possible paths: 2. **Bottom:** Two merged plans, number of possible paths: 4.

An intuitive approach is to compare and require the equivalence of the underlying PDDL state at each time point along the original plan it belongs to (i.e. the subset of Boolean propositions $s \subseteq F$). Unfortunately, this does not cover all possible cases and can be very sensitive to domain representation. For instance if we return to the example in Figure 3 and introduce a new predicate $Walked$, which activates once our agent executes the *Walk* action, e_f^1, e_f^2 will not equivalent by state space even though this proposition might have no future affect on the rest of the plan.

Another approach that comes to mind is to compare action sequences, meaning instead of searching through the state space, we will instead direct our attention at the plan space and look for matching action sequences in different plans. This approach falters when the sequences can be arbitrarily positioned in a plan. It might recognize a matching sequence of actions between two plans and wish to merge them without taking into account that one might be at the end of plan i while the other at the beginning of plan j. Consider for example a classic logistics domain with trucks and locations. Imagine a scenario where *truck1* must travel from location A to location B both at the beginning of the plan (to pick up a package) and at its end (to park there). An action sequence oriented approach might suggest a merge based on these identical actions thus creating a merge between one plan's beginning to another's end.

We would want to avoid these scenarios by creating a constraint on the merges to be somehow in accordance with a plan's execution state.

Compatibility

Thus we suggest a method to determine whether two time points should be merged based on the validity of the solutions originating from their merge. We introduce the *compatibility* attribute of two IHs. We define compatibility based on IHs as opposed to time points as the latter can change when merging time points in the TPN while the former remains a static property of the TPSs of the original diverse solutions. For instance, the outcome of any merge between two TPN time points which do not share the *exact* underlying PDDL state is necessarily a single TPN time point e who's underlying PDDL state is unknown in advance and depends on choices made during execution, before e was reached. We define two notions of compatibility:

Definition 2 (**Full and Semi Compatibility between IHs**). *Given a pair of IHs a_1, a_2 from TPSs π_i, π_j respectively, let s_{a_1}, s_{a_2} be the underlying PDDL state after these IHs occurred in π_i, π_j, respectively, and let their corresponding TPS suffixes be $\Sigma_{a_1}^{\pi_i}, \Sigma_{a_2}^{\pi_j}$. a_1, a_2 are:*

- ***Fully compatible*** *iff* $G \subseteq T(s_{a_1}, \Sigma_{a_2}^{\pi_j})$ *and* $G \subseteq T(s_{a_2}, \Sigma_{a_1}^{\pi_i})$
- ***Semi compatible*** *iff* $G \subseteq T(s_{a_1}, \Sigma_{a_2}^{\pi_j})$ *or* $G \subseteq T(s_{a_2}, \Sigma_{a_1}^{\pi_i})$

In other words two IHs a_1, a_2 are Fully Compatible if we can execute the TPS suffix of each π from the other's current state s_a and achieve the goal, while two IHs a_1, a_2 are merely Semi Compatible if we can execute at least one of the TPS suffixes from the other's current state s_a and achieve the goal.

We denote the set containing all such compatible pairs as $\mathbb{M}$. Since $\mathbb{M}$ is static, we can efficiently compute it once at the beginning of the process. Each pair of compatible IHs $\{a_1, a_2\} \in \mathbb{M}$ is an operation m corresponding to a merge we can perform on the TPN time points.

From now on, we will restrict our attention to applying only merges between pairs contained in $\mathbb{M}$ to the TPN. While this limits the space of possible TPNs, it also reduces the complexity of the problem to a manageable size.

Merging two time points in the TPN outputs a single new time point. This new time point must account for all IHs involved in the merge (recall that a time point may consist of multiple IHs). Therefore the merging of two time points is an operation between sets of IHs. Consider the 2-step merging sequence to the naive TPN $m(a_1, a_2), m(a_2, a_3)$. The first merge operation creates a new time point e_{new}. The second merge operation is now $m(e_{new}, a_3) = m(\{a_1, a_2\}, a_3)$.

This scenario raises questions about the compatibility attribute as it applies to sets of IHs. We can define different transitivity notions when merging in order to experiment with this concept and widen or narrow our solution space. We define two notions of transitivity in when we allow merges:

Definition 3 (**Strict and Loose transitivity between time points**). *Given two time points e_1, e_2 where each time point is a set of IHs, we will define a merge as applicable iff:*

- ***Strict:*** $\forall a_i \in e_1 \land \forall a_j \in e_2 : \{a_i, a_j\} \in \mathbb{M}$
- ***Loose:*** $\exists a_i \in e_1 \land \exists a_j \in e_2 : \{a_i, a_j\} \in \mathbb{M}$

We have not limited ourselves to generating TPNs with only valid solutions as the TPN executive (Pike) incorporates the ability to avoid making combinations of choices that lead to infeasible paths (Levine and Williams 2018).

Merge Selection

Once $\mathbb{M}$ has been acquired, we require a method to determine which merge operations to execute. The reason for this necessity is that the merge operations in $\mathbb{M}$ are only compatible by definition in the naive TPN. Consider the following configuration — the naive TPN contains three TPSs π_1, π_2, π_3, and each path contains time points e_1, e_2, e_3 in π_1, π_2, π_3, respectively, such that e_1 is fully compatible with e_2 *or* e_3 but not with both. This scenario demonstrates the dilemma we now face — which pairs of time points to merge?

Constraint Programming To solve this optimization problem we call upon CP to maximize the number of merges we can apply to the naive TPN and formulate the problem as a COP. We define the set of all IHs participating in the optimization as N, in other words, these are all the *different* IHs appearing in $\mathbb{M}$. To mark a pair of IHs $a_i, a_j \in N$ as chosen to be merged together, we define the Boolean decision variable $m_{i,j}$. The set of all such variables is then $M \equiv \{m_{i,j} \mid \forall i, j \in N\}$.

Our objective is to create the smallest possible TPN, and this is done by merging *groups* of IHs. Thus, to formulate our optimization objective using the $m_{i,j}$ decision variables, we must create auxiliary decision variables for keeping track of groups. For example, merging a_1, a_2 and a_3 sets six decision variables to TRUE ($m_{1,2}, m_{2,1}, m_{1,3}, m_{3,1}, m_{2,3}$ and $m_{3,2}$), but only counts as one group. To do so we assign each IH the shared minimal index i of its group. For example, given the group containing IHs a_1, a_2, a_3, each IH is assigned the shared minimal index 1. This assignment is described via an additional decision variable s_i of type integer which is simply derived from the $m_{i,j}$ decision variables. The set of all such variables is then $S \equiv \{s_i \mid \forall i \in N\}$ The initial value for each shared minimal index variable is its own index: $s_i = i$. The COP therefore contains $N^2 + N$ decision variables.

Let us now formulate the constraints applied in our COP. These will also define the merging transitivity, as discussed previously, which we wish to apply to the TPN. The following constraints hold for both Strict and Loose configurations:

- **Compatible Merges**: $m_{i,j} \implies \{a_i, a_j\} \in \mathbb{M}$.
- **Symmetric Merges**: $m_{i,j} \iff m_{j,i}$.
- **Shared Minimum Node**: $s_i \neq i \implies m_{i,s_i}$ and $m_{i,j} \implies s_i = min(s_i, s_j)$

The Strict configuration contains a single additional constraint dictating that any merging between three time points must uphold that they are all compatible with one another:

$$m_{i,j} \wedge m_{i,k} \implies \{a_i, a_j\}, \{a_i, a_k\}, \{a_j, a_k\} \in \mathbb{M}$$

The Loose configuration allows more freedom for applying merges but must make sure no two IHs originating from the same original solution are merged. Therefore, for this configuration, we pass P, a mapping from time points to original solutions, as input to the COP. Therefore, $P \equiv \{p_i \mid i = 1, ..., k\}$ where k is the number of TPSs. The additional constraints are then:

- $m_{i,j} \implies p_i \neq p_j$
- $m_{i,j} \wedge m_{i,k} \implies p_j \neq p_k$
- $s_i = s_j \implies p_i \neq p_j$

Lastly, the objective function of the COP is to maximize the number of elements in S who's value differs from their index, in other words we want to count the number of IHs which were merged. Formally:

$$f = Max \sum_{i=0}^{N} \mathbb{1} \mid s_i \neq i$$

Empirical Evaluation

In order to empirically evaluate our algorithm, our compilation takes a temporal planning task Π and generates a set of k diverse solutions using the plan elimination compilation described in Section 4, with OPTIC (Benton, Coles, and Coles 2012) as the underlying solver. The TPS of each solution is obtained and $\mathbb{M}$ is computed, containing all the compatible time point pairs found in the naive TPN. We then construct a COP based on $\mathbb{M}$ in Minizinc (Nethercote et al. 2007) and use Gecode (Gecode Team 2020) to solve it. As output, we receive a mapping from time point pairs to merge operations resulting in a new TPN. If no compatible pairs are found (i.e. $\mathbb{M} = \emptyset$) the naive TPN is returned as output.

IPC Benchmarks

We evaluated all combinations of k chosen from $\{2, 4, 8\}$, and both compatibility methods (Full & Semi denoted as F,S) and merging transitivity (Strict & Loose denoted as St,L). Thus, for each problem, we ran the diverse planner 3 times (for the different values of k), and then ran 4 different versions of our COP (for the different compatibility and transitivity). The experiments were performed on Intel i7-7700K 32GB RAM, with a time limit of 30min for generating the diverse solutions and 5min for the COP task.

We evaluated our approach on all domains from the temporal track IPC in 2011, 2014, and 2018. Of these, we eliminated domains with actions whose durations depend on the current state, as this is not supported in a TPN. Out of these, we kept the domains where OPTIC was able to retrieve multiple diverse solutions for more than a single problem.

We define a run on a specific problem to be successful if:

1. OPTIC is able to obtain k diverse solutions.
2. The generated TPN is more compact than the naive TPN.

Otherwise, although the algorithm might have terminated successfully, we do not count it. Such a scenario occurs when the generated solutions in Π are very different from one another and no compatible pairs are found between them, leading to no possible merges to the naive TPN. In the left hand side of Table 1 we report the number of problems for which we were able to obtain k solutions as #N, and the number of successful runs is shown in the columns per configuration.

We also report by how much our approach was able to reduce the size of the naive TPN. As different values of k for the same problem lead to different sizes of the naive TPN, we evaluate the compactness of the generated TPN relative

	Successes														
#Solutions	k=2					k=4					k=8				
Transitivity	St		L		#N	St		L		#N	St		L		#N
Compatibility	F	S	F	S	-	F	S	F	S	-	F	S	F	S	-
crewplanning(30)	5	5	5	5	25	5	5	5	5	25	2	0	3	1	22
parking(10)	8	8	8	8	9	9	9	9	9	9	6	6	6	6	6
quantom circuit(10)	5	5	5	5	8	5	5	5	5	6	5	5	5	5	5
trucks(10)	8	8	8	8	10	10	9	10	10	10	8	8	9	9	10
turn and open(20)	2	2	2	2	2	1	1	1	1	1	0	0	1	0	1
Total | Avg	28	28	28	28	54	30	29	30	30	51	21	19	24	21	44

	Compactness														
#Solutions	k=2					k=4					k=8				
Transitivity	St		L		#P	St		L		#P	St		L		#P
Compatibility	F	S	F	S	-	F	S	F	S	-	F	S	F	S	-
crewplanning(30)	0.017	0.116	0.017	0.113	5	0.185	0.175	0.185	0.178	5	-	-	-	-	0
parking(10)	0.042	0.042	0.042	0.042	8	0.09	0.098	0.09	0.098	9	0.117	0.122	0.102	0.111	6
quantom circuit(10)	0.035	0.035	0.035	0.035	5	0.072	0.072	0.072	0.079	5	0.131	0.126	0.066	0.075	5
trucks(10)	0.05	0.081	0.05	0.077	8	0.059	0.063	0.056	0.058	9	0.066	0.048	0.032	0.023	8
turn and open(20)	0.03	0.057	0.03	0.065	2	0.16	0.043	0.014	0.02	1	-	-	-	-	0
Total | Avg	0.035	0.066	0.035	0.066	28	0.113	0.09	0.083	0.087	29	0.105	0.099	0.067	0.07	19

Table 1: Summary of Empirical Results

to the size of the naive TPN:

$$compactness(TPN) = 1 - \frac{|\mathcal{E}_{new}|}{|\mathcal{E}_{naive}|}$$

In the right hand side of Table 1 we show the average compactness over the commonly solved problems for each k by the four different configurations of our approach. #P denotes the number of commonly solved problems for each k.

Results

Before we analyze the results, we note that the larger k is, the more IHs we have to compare between diverse solutions. Therefore we expect that an increase in k will result in more compatible pairs and thus in more merges and better compactness, but at the cost of more computational effort.

As the results in Table 1 show, the above intuition is partially correct. First, as the intuition suggests, for larger k the number of successes decreases. This can be explained by the rise in complexity of the COP due to many compatible pairs and the associated high memory consumption. However, on the other extreme, when k is low there is a greater probability that the few diverse solutions generated will differ significantly from one another. Such instances may lead to either a low number of compatible pairs – less merges and a worse compactness *or* finding no compatible pairs all together and resulting in an unsuccessful run. Indeed, this is what happened in the parking and trucks domains, where using $k = 4$ resulted in more success than either $k = 2$ or $k = 8$.

We now turn out attention to examining the differences between the four configurations of our approach. First, note that using semi-compatibility always results in at least as many compatible pairs as using full-compatibility. This increase in the number of possibilities leads to an increase in the difficulty of solving the COP, which explains the higher success count of the full-compatibility, as can be seen in crewplanning for $k = 8$ and truck for $k = 4$.

On the other hand, the extra possibilities allow for more merges, and thus for better compactness. This is especially evident for $k = 2$, where a single merge contributes more to the compactness than for higher values of k, since it involves time points from a higher proportion of the original solutions.

Comparing using strict transitivity to loose transitivity, the former is a stricter constraints, thus pruning the space of possible solutions. With lower values of k, this pruning makes little difference, implying that the best solutions are not typically not pruned by this. With higher values of k this pruning reduces the size of the search space, allowing the solver to find better solutions in the alloted time, at the cost of a slight reduction in the success count.

Summary and Future Work

We have presented the first approach for automatically generating Temporal Planning Networks from a description of a planning task. This makes the useful tools based on the TPN formalism, such as the Pike executive (Levine and Williams 2018), much more broadly applicable, as there is no need to manually generate a TPN or RMPL program (Kim, Williams, and Abramson 2001). We have also adapted the plan reformulation elimination (Katz et al. 2018) to the temporal setting, thus creating the first diverse temporal planner. In this paper, we focused on fully controllable TPNs. In future work, we will address the uncertainty inherent in some domains, such as human-robot teamwork – one of the original motivations for the TPN formalism. In order to do this, we intend to use a multi-agent formalism, such as MA-STRIPS (Brafman and Domshlak 2008), and define which agents are under our control and which are not. The objective here will be to generate a TPNU.

Additionally, the objective we optimized here, minimizing the size of the resulting TPN, is only one possible objective. In the context of human-robot teamwork, one may want to optimize for some human-focused metrics, such as the ease of explaining the TPN to the human, the mental effort needed to keep track of the current state of execution, and the flexibility given to the human at any given moment during execution.

Finally, as previously mentioned, TPNUs can serve as a middle ground between contingent planning and conformant planning or naive replanning. We intend to explore using such automatically generated TPNUs in planning under uncertainty, where a TPNU is generated, and executed until something outside its specification occurs, when a new TPNU is synthesized.

References

Benton, J.; Coles, A. J.; and Coles, A. 2012. Temporal planning with preferences and time-dependent continuous costs. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012, Atibaia, São Paulo, Brazil, June 25-19, 2012*.

Brafman, R. I., and Domshlak, C. 2008. From one to many: Planning for loosely coupled multi-agent systems. In *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling, ICAPS 2008, Sydney, Australia, September 14-18, 2008*, 28–35.

Bryce, D. 2014. Landmark-based plan distance measures for diverse planning. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2014, Portsmouth, New Hampshire, USA, June 21-26, 2014*.

Burke, S.; Fernandez, E.; Figueredo, L.; Hofmann, A.; Hofmann, C.; Karpas, E.; Levine, S.; Santana, P.; Yu, P.; and Williams, B. 2014. Intent Recognition and Temporal Relaxation in Human Robot Assembly. In *ICAPS Demo Track*.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artif. Intell.* 49(1-3):61–95.

Fox, M., and Long, D. 2003. PDDL2.1: an extension to PDDL for expressing temporal planning domains. *J. Artif. Intell. Res.* 20:61–124.

Gecode Team. 2020. Gecode: Generic constraint development environment. Available from `http://www.gecode.org`.

Hoffmann, J., and Brafman, R. 2005. Contingent planning via heuristic forward search with implicit belief states. In *Proc. ICAPS*, volume 2005.

Ingham, M.; Ragno, R.; and Williams, B. C. 2001. A reactive model-based programming language for robotic space explorers. *Proceedings of ISAIRAS-01*.

Katz, M., and Sohrabi, S. 2019. Reshaping diverse planning: Let there be light! *HSDIP 2019* 1.

Katz, M.; Sohrabi, S.; Udrea, O.; and Winterer, D. 2018. A novel iterative approach to top-k planning. In *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling, ICAPS 2018, Delft, The Netherlands, June 24-29, 2018*, 132–140.

Kim, P.; Williams, B. C.; and Abramson, M. 2001. Executing reactive, model-based programs through graph-based temporal planning. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001, Seattle, Washington, USA, August 4-10, 2001*, 487–493.

Levine, S. J., and Williams, B. C. 2018. Watching and acting together: Concurrent plan recognition and adaptation for human-robot teams. *J. Artif. Intell. Res.* 63:281–359.

Little, I., and Thiebaux, S. 2007. Probabilistic planning vs. replanning. In *ICAPS Workshop on IPC: Past, Present and Future*.

Nethercote, N.; Stuckey, P. J.; Becket, R.; Brand, S.; Duck, G. J.; and Tack, G. 2007. Minizinc: Towards a standard CP modelling language. In *Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007, Proceedings*, 529–543.

Nguyen, T. A.; Do, M. B.; Gerevini, A.; Serina, I.; Srivastava, B.; and Kambhampati, S. 2012. Generating diverse plans to handle unknown and partially known user preferences. *Artif. Intell.* 190:1–31.

Roberts, M.; Howe, A. E.; and Ray, I. 2014. Evaluating diversity in classical planning. In Chien, S. A.; Do, M. B.; Fern, A.; and Ruml, W., eds., *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2014, Portsmouth, New Hampshire, USA, June 21-26, 2014*. AAAI.

Srivastava, B.; Nguyen, T. A.; Gerevini, A.; Kambhampati, S.; Do, M. B.; and Serina, I. 2007. Domain independent approaches for finding diverse plans. In *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, 2016–2022.

Timmons, E.; Fang, C.; Fernandez-Gonzalez, E.; Karpas, E.; Levine, S. J.; Santana, P.; Wang, A. J.; Wang, D.; Yu, P.; and Williams, B. C. 2015. Reactive model-based programming of micro-uavs. In *ICAPS Demo Track*.

Yoon, S. W.; Fern, A.; and Givan, R. 2007. Ff-replan: A baseline for probabilistic planning. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling, ICAPS 2007, Providence, Rhode Island, USA, September 22-26, 2007*, 352.