Contents

Chapter 1

Introduction

1.1 Motivation

As technology progresses, robots are becoming more and more prevalent in everyday life. Be it physical robots such as an robotic arms and Roombas, or be it robotic agents such as Siri and Alexa. One of their great strengths is in being very efficient and able to execute precise operations and calculations in short time in a repeated and monotonous fashion. As their capabilities grow, so do we allow them to seep into more and more of complex fields such as medicine, economy and transportation. Areas such as these require not only precise and quick calculations but also the ability to handle uncertainty in real time. This uncertainty may stem from a dynamic environment or even from the need to work alongside a human counterpart. Unlike a sterilized factory floor, the world outside is dynamic and constantly on the move, be it through weather change, time of day lighting, moving objects etc.. The number of unique situations such a robot can experience are practically limitless. How then can one configure a robot to deal with such a dynamic environment? How do we allow our agent to deal with uncertainty?

Several approaches exist that tackle this very question. For the sake of clarity we will describe a scenario that will serve us as a running example throughout this thesis.

Running Example: A Hard Day's Night. *Rob the robot has had a very long day at the plant and is looking forward to some quality time at home to charge his battery. Once he punches a hole in his worksheet and steps out the door he stops to think what route should he take home?*

One common approach is to "determinize and replan" [22], that is, come up with a single plan which might solve the problem, start executing it, and if anything goes wrong — replan.

Rob decides to take the bus. He approaches the bus station and discovers that no buses are scheduled to arrive soon. Not a big fan of waiting, Rob decides to think of another way he can get home. After some thought Rob decides his next best option is to walk home - saving some money and breathing in some fresh air.

Although this approach often works, it preforms poorly on problems which are "probabilistically interesting" [17], such as problems with avoidable dead-ends. In addition, replanning takes place in real-time i.e. during online execution and results in the agent being *offline* for the duration. 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., [11]), 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.

While pondering how to commute home, Rob finds himself lost in thought about all the things that could go wrong with his plan. "What if the bus has a flat tire? What if the road is blocked? What if I fall and break my leg? what if..."

We advocate taking a middle-ground approach between the two aforementioned methods by using flexible plans with choices. This scheme bestows our agent access to multiple pre-calculated solutions (plans) while allowing it to navigate between them freely by making *choices* along the way, as it proceeds towards its goal. This way we are able to address an extent of the uncertainty as determined by the number of plans we include.

Rob knows that there are many possible ways he can get home. He chooses several of them and notes to himself how he can transition between them in case something goes wrong. "I can go to the bus stop, if there aren't any buses soon I'll hop on a taxi. If the road is blocked I'll just start walking from there..."

Specifically, we advocate using the Temporal Planning Networks formalism [15], referred to as TPNs. 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 [4], and in controlling micro-UAVs [21]. The Pike executive [16], 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 using intent recognition and plan adaptation techniques.

While the effectiveness of Pike and TPNs has been shown, both suffer from lack of presence in the industry. This can be associated with the fact that that generating TPNs has so far only been done by manually encoding them, a tedious job requiring a domain expert. Although it is possible to compile a control program written in the Reactive Model Planning Language (RMPL) [12] into a TPN, the control program must still be manually written.

1.2 Problem Statement

In this work, 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 [8]. Furthermore, we will attempt that the generated TPN be *good*.

In this work we shall define a good TPN as being compact but flexible, we further elaborate on this subject in section 3.1.

1.3 Problem Approach

Our solution to the task at hand relies on obtaining multiple dissimilar solutions that solve the same Temporal Planning problem. This scheme is known as Diverse Planning. Although the problem of attaining multiple diverse solutions is well studied, all approaches refer to Classical Planning problems, while none address the temporal domain. Therefore, unfortunately, none of the current available off-the-shelf diverse planners [3, 19, 20, 13] fit our requirements. Thus, Our first step is to describe a diverse temporal planner, based on adapting the top-k based diverse planner [13] to the temporal setting.

After the dissimilar plans are obtained, they are passed to our process and merged into a single representation. The merging algorithm is based on identifying sets of points along the different solutions which *can* be merged together based on criteria that we define.

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 that encompass all original solutions and, if possible, finds additional ones.

1.4 Thesis Roadmap

The structure of this Thesis is as follows.

Chapter 2 covers the relevant background required for this work.

Next, chapter 3 provides an in depth description of our working process.

Chapter 4 provides an empirical evaluation of our process on a set of IPC benchmarks and shows that our approach can quickly generate TPNs, even for large problems.

Lastly, chapter 5 summarizes the accomplishments of this Thesis and raises open questions for future work down this path.

Chapter 2

Background

We start this section with a brief introduction to automated planning, one of the earliest areas of Artificial Intelligence, which addresses the problem of synthesizing autonomous behaviors in automated way from a model.

2.1 Automated Planning

Automated Planning is the model-based branch of Artificial Intelligence (AI) focused on obtaining a solution, described through a sequence of actions, to a certain task using the knowledge of the world at hand.
More specifically, given a model describing the world and the interactions available to agents in it, including their corresponding effects on it, we can describe an initial state and a goal state of the world and find a solution that transitions between them, if such exists. This solution constitutes a sequence of actions, denoted from now on as a "Plan", that if executed accordingly by the domain agents, promises the arrival at the goal state. Finding such plans is not a trivial issue and is a task reserved for a family of complex solvers called Planners.
Planners are currently seen as automated solvers for precise classes of mathematical models represented in compact form. A typical planner takes three inputs: a description of the initial state of the world (initial state), a description of the desired goals (goal state), and a set of actions that the executor (agent) is able to perform, all encoded in a formal language such as PDDL [10]. Planners are in general domain-independent in the sense that they do not know what the variables, actions, and domain stand for, and for any such description they must decide effectively which actions to initiate in order to achieve the goals for the given planning task. The planner produces an ordered set of actions that leads from the initial state to a state satisfying all the goals.

In the circumstances of our work, as is in life, it is not enough to have a great plan, there is also a need for timing.

2.1.1 Temporal Planning

Temporal Planning is the extension of the classical planning regimes to the temporal setting. The classical formalism does not take into consideration the aspect of time and assumes actions taken in the world are instantaneous and take place in sequential order. In order to model more realistic problems there was need to control when each action should happen and allow actions to occur in parallel to one another. For Instance, the logistics domain would highly benefit from parallel actions if a robot equipped with two arms would be able to use them simultaneously and not one after the other. Adding time to the equation would also allow a factory robot to time his actions according to a shipment scheduled to arrive that same day. When involving time in a solution, the output is no longer a "plan" (sequence of actions) but a "schedule", mapping actions to their designated occurrence and execution stages. The third International Planning Competition (IPC), held in 2002, presented the planning community with the challenge of handling time. This necessitated the development of a modelling formalism capable of expressing temporal properties of planning domains which lead to the emergence of PDDL2.1

A Temporal Planning problem modeled in the propositional subset of PDDL 2.1 [8] is given by a tuple $\Pi = \langle F, A, I, G \rangle$:

By using the above formalism to describe a temporal planning task, we can commit Π as input to a temporal Planner and obtain a schedule τ.

We now know how to obtain a single solution to our input temporal planning task. If we require a method for obtaining multiple different solutions.

2.1.2 Diverse Planning

Diverse planning is a method for obtaining multiple dissimilar solutions to the same planning task. It is commonly used in decision support scenarios where a human analyst needs to understand the space of possible plans or has difficulty specifying the planning domain model that exactly matches their application. The diversity between plans in a plan set Λ is measured by the average distance between them. Previous literature such as [3] discussed how to best take advantage of plan characteristics to compare and measure this distance using domain-independent criteria.

Each such criteria has its own distance metric $D(\pi, \pi')$ to calculate the dissimilarity between two given plans.

The Maximal Plan Diversity Problem *Given a plan distance metric D, a number k and a planning problem instance P, obtain a collection of k solution plans $\pi_1, \pi_2 ... \pi_k \in \Lambda$ solving P, such that no other set of k solution plans Λ' solving P upholds $Div(\Lambda) < Div(\Lambda')$*

Finding the maximum set can be impossible depending on how D is defined[5]. Therefore, a relaxation of this problem is solved instead, where a distance metric

Var	Definition
F:	The domain facts. A 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 $\text{dur}_{max}(a)$ uphold $0 \leq \text{dur}_{min}(a) \leq \text{dur}_{max}(a)$. • 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). • Invariant condition $\text{inv}(a) \subseteq F$ which must hold during the whole execution of a.
I:	The initial state, specifying exactly which propositions in F are true at time zero. $I \subseteq F$.
G:	The goal, which propositions we wish to be true at the end of plan execution. $G \subseteq F$.

Table 2.1: Temporal Planning PDDL2.1 formalism

D is taken into account during plan set generation, but maximal plan diversity is not guaranteed.

Top-k Planning Top-k planning is the task of obtaining a set of k solutions s.t. there exists no better quality solution outside that set. This approach values the quality of the solutions over their diversity from one another. The iterative plan Forbid Reformulation [14] approach to top-k planning exploits cost-optimal planners by iteratively reformulating the original planning task to forbid exactly the solution they acquire at each iteration. The solution at each iteration is added to the solution set until extracting exactly k plans.

In this work we made use of the diverse planner [13], which uses a top-k planner to generate the top-k best dissimilar solutions to the planning task at hand. As part of this work, we have extended this planner to handle the Temporal Planning setting.

We are now ready to take a closer look at TPNs and their previous usages in the literature.

2.2 TPNs & Pike

A Temporal Planning Network (TPN) is a formalism for representing flexible plans with choices. A TPN is an extension to the Simple Temporal Network [7], 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 [16] for a Temporal Planning Network under Uncertainty (TPNU), 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, i.e. which choices does Pike initiate and which does the environment. Formally a TPNU is a tuple $\langle V, \mathcal{E}, C, A \rangle$, 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. $\varphi_c \implies (l \leq e_f - e_s \leq u)$.

- $\mathbb{A}$: 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. there is no assignment of choices to the uncontrollable set. The task of determining which choices are uncontrollable is a subject for future work.

Pike

Pike [16] 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 achieves plan recognition and adaptation concurrently

through a single set of algorithms. Pike takes as input a flexible plan with choices defined using the TPNU formalism. During execution, Pike makes use of a state estimator and activity recognition module for additional inputs in order to access and respond to state disturbances and recognize the choices made by the human counterpart. Using these inputs Pike infers his team-mate's intent by reasoning about the uncontrollable choices, and makes the controllable choices such that the end-goal will be achieved. 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.
In order to dispatch activities at their proper times, which may also depend on the choices made, Pike leverages ideas originally presented in the Drake executive [6]. A labeled all-pairs shortest path (APSP) is computed on the TPNU, to determine which events occur before which other events. This labeled APSP is used as a dispatchable form for execution.

To make sure our process provides Pike with high quality TPN, we will want some optimization formalism to use during the generation phase.

2.3 Constraint Optimization

A Constraint Optimization Problem (COP) is a mathematical optimization question defined by an objective function with respect to some variables in the presence of constraints on those variables. These constraints can either be strictly required (hard constraints) or have variable range values for which to be penalized to an extent based on the conditions that are not satisfied (soft constraints).

A solution to a COP is a mapping from each variable to a value, selected from a finite domain, s.t. the objective functions achieves it's desired value, be it minimal for such problems as reducing cost or energy usage, or maximal for accumulating reward or raising utility. The constraints of the problem limit the variable values that can be assigned simultaneously.

Formally a COP is a tuple $\langle X, D, C, O \rangle$:

Var	Definition
X:	The set of variables.
D:	The set of their respective domains of values.
C:	The set of constraints.
O:	The objective function.

Table 2.2: Constrain Optimization Problem

Further on, in chapter 3, when constructing a TPN out of the diverse solutions we obtain, we will model the question of which time-points to merge between plans, such as to maximize our TPN's utility, as a COP.

Chapter 3

Automatically Generating Temporal Planning Networks

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.

3.1 Formal Problem Definition

> *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.*

Informally we don't just want any TPN. We wish to generate a "good" TPN. As no current methods for comparing TPNs exist, we chose in this work to focus on generating the most compact TPN representation possible. To achieve this we will maximize the number of points our process merges along the obtained diverse plans.

While we are aware that other TPN attributes are also comparison-worthy, we advocate that 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. In addition, this objective steers towards a solution that requires more merges between the diverse plans, meaning more connections between them and as a result more flexibility during execution.

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 often leads to a new decision variables being generated. 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.

Additionally, Pike (the TPN executive) 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 the TPN constraints C.

We are now ready to start getting our hands dirty and describe how we tackle this problem.

3.2 Methods

First off, we describe our devised approach to Diverse Temporal Planning.

3.2.1 Diverse Temporal Planning

Our approach to diverse temporal planning builds upon the top-k approach in [13] using a *plan forbidding reformulation* as in [14] .

The top-k approach defines metrics in order to measure how close plans are to the best subset of all known plans, while the forbidding reformulation constitutes a diverse planning algorithm that instead of modifying the planner, in order to find a dissimilar solution, modifies the planning task itself. The reformulation occurs after each iteration (after which a plan is found) and alters the planning task in such a way as to forbid the possibility of finding the same solution that was obtained in the previous iteration.

The main challenge we address here is that temporal plans are not a sequence of instantaneous happenings (IHs), 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).*

In other words, by observing only the sequence of occurrences in a temporal planning schedule, i.e. *action-start* and *action-end* events, we can refer to this TPS as a sequential plan for our diverse planning purposes.
We now define the *TPS suffix* which will come in handy in section 3.2.2 when choosing time-points for merging.

Definition 2 (TPS Suffix) *Given a TPS* π *and an IH* $a \in \pi$*. The TPS suffix of* π *from* a*, denoted* Σ_a^π*, is the ordered 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

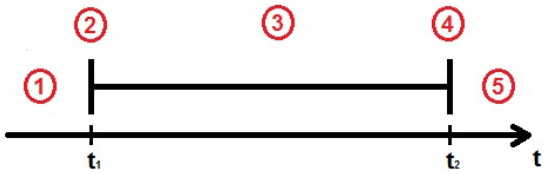

Figure 3.1: All the possible points to deviate from TPS π, from the point of view of the i^{th} action in π, a_i.

same TPS, and which 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 [13] 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* [14] 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 3.1, and correspond to the five cases described below.

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

Case 2: We 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 [14], 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:

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

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

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 variable 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 *a solution with TPS* π*. The task* Π' *is a plan elimination reformulation of* Π *and* π*.*

Running the plan elimination reformulation on our input temporal planning task iteratively will yield the diverse solutions we desire.

With a set of diverse TPSs $\{\pi_1 \dots \pi_k\}$ at hand for our input planning task Π, we can dive in to tackling how to merge these into a single TPN representation, that compactly encodes all generated solutions and possibly more.

3.2.2 Merging Diverse Plans into a TPN

A 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 initial state, which is shared, 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 to is shared. We denote this structure from now on as the *naive TPN*. Although this approach defeats the purpose of having a flexible plan with choices, we would like to emphasize the fact that the naive TPN is attainable for any set of k solutions. This means that no matter how big k is or the length of each solution, we can always generate an functioning TPN to pass onward to Pike. This by itself is already good news and provides our process with a degree of safety as a first step.

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 3.2.

create example like in introduction

In this example, after a long day at work, our agent needs to get home and eat dinner. 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.

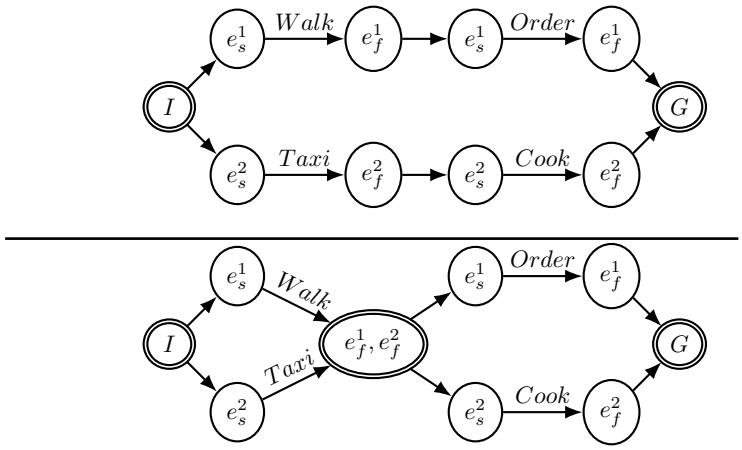

Figure 3.2: Additional paths in the graph are created as a result of a merge. Choice time points are depicted with double circles.
Top: Two separate plans, number of possible paths: 2.
Bottom: Two merged plans, number of possible paths: 4.

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.

Some intuitive approaches include *i)* comparing the underlying PDDL states of time points and *ii)* comparing ordered sequences of activities originating from time points. Unfortunately, both techniques fall short of accounting for important merges possibilities we wish to obtain.

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. We define two notions of compatibility:

Definition 3 (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: Formally, a merge between time points e_1, e_2 is applicable iff:

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

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 [16].

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.

3.2.3 Merge Selection

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: *i)* $m_{i,j} \implies p_i \neq p_j$ *ii)* $m_{i,j} \wedge m_{i,k} \implies p_j \neq p_k$ *iii)* $s_i = s_j \implies p_i \neq p_j$

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

This concludes our TPN generating process.
we now showcase it's empirical results.

Chapter 4

Results

4.1 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 [1] 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 [18] and use Gecode [9] 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.

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: *i)* OPTIC is able to obtain k diverse solutions to the problem and *ii)* 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 4.1 we report the number of problems for which we

were able to obtain k solutions as #N, and the number of successful runs as #S. 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 to the size of the naive TPN: $compactness(TPN) = 1 - \frac{|\mathcal{E}_{new}|}{|\mathcal{E}_{naive}|}$. In the right hand side of Table 4.1 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.

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.

4.2 Results Discussion

As the results in Table 4.1 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.

	#S												Compactness											
#Solutions	k=2					k=4				k=8			k=2					k=4				k=8		
Transitivity	St		L		#N	St		L		St		L		St		L		#P	St		L		#P
Compatibility	F	S	F	S	-	F	S	F	S	#N	F	S	F	S	#P	F	S	F	S	#P	F	S	F	S	-
crewplanning(30)	5	5	5	5	25	5	5	5	5	25	2	0	3	1	22	0.017	0.116	0.017	0.113	5	0.185	0.175	0.185	0.178	5	-	-	-	-	0
parking(10)	8	8	8	8	9	9	9	9	9	9	6	6	6	6	6	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)	5	5	5	5	8	5	5	5	5	6	5	5	5	5	5	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)	8	8	8	8	10	10	9	10	10	10	8	8	9	9	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)	2	2	2	2	2	1	1	1	1	1	0	0	1	0	1	0.03	0.057	0.03	0.065	2	0.16	0.043	0.014	0.02	1	-	-	-	-	0
Total | Avg	28	28	28	28	54	30	29	30	30	51	21	19	24	21	44	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 4.1: Summary of Empirical Results

Chapter 5

Conclusion and open questions

5.1 Conclusion and open questions

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 [16], much more broadly applicable, as there is no need to manually generate a TPN or RMPL program [15]. We have also adapted the plan reformulation elimination [14] 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 [2], 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.

Bibliography

[1] J. Benton, Amanda Jane Coles, and Andrew Coles. "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*. 2012.

[2] Ronen I. Brafman and Carmel Domshlak. "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*. 2008, pp. 28–35.

[3] Daniel Bryce. "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*. 2014.

[4] Sean Burke et al. "Intent Recognition and Temporal Relaxation in Human Robot Assembly". In: *ICAPS Demo Track*. 2014.

[5] Alexandra Coman and Hector Muñoz-Avila. "Generating Diverse Plans Using Quantitative and Qualitative Plan Distance Metrics". In: *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011*. Ed. by Wolfram Burgard and Dan Roth. AAAI Press, 2011.

[6] Patrick Raymond Conrad. "Flexible execution of plans with choice and uncertainty". PhD thesis. Massachusetts Institute of Technology, 2010.

[7] Rina Dechter, Itay Meiri, and Judea Pearl. "Temporal Constraint Networks". In: *Artif. Intell.* 49.1-3 (1991), pp. 61–95.

[8] Maria Fox and Derek Long. "PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains". In: *J. Artif. Intell. Res.* 20 (2003), pp. 61–124.

[9] Gecode Team. *Gecode: Generic Constraint Development Environment*. Available from `http://www.gecode.org`. 2020.

[10] Malik Ghallab et al. "PDDL - The Planning Domain Definition Language". In: (Aug. 1998).

[11] Jörg Hoffmann and Ronen Brafman. "Contingent planning via heuristic forward search with implicit belief states". In: *Proc. ICAPS*. Vol. 2005. 2005.

[12] Michel Ingham, Robert Ragno, and Brian C Williams. "A reactive model-based programming language for robotic space explorers". In: *Proceedings of ISAIRAS-01* (2001).

[13] Michael Katz and Shirin Sohrabi. "Reshaping Diverse Planning: Let There Be Light!" In: *HSDIP 2019* (2019), p. 1.

[14] Michael Katz et al. "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*. 2018, pp. 132–140.

[15] Phil Kim, Brian C. Williams, and Mark Abramson. "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*. 2001, pp. 487–493.

[16] Steven James Levine and Brian Charles Williams. "Watching and Acting Together: Concurrent Plan Recognition and Adaptation for Human-Robot Teams". In: *J. Artif. Intell. Res.* 63 (2018), pp. 281–359.

[17] Iain Little and Sylvie Thiebaux. "Probabilistic planning vs. replanning". In: *ICAPS Workshop on IPC: Past, Present and Future*. 2007.

[18] Nicholas Nethercote et al. "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*. 2007, pp. 529–543.

[19] Tuan Anh Nguyen et al. "Generating diverse plans to handle unknown and partially known user preferences". In: *Artif. Intell.* 190 (2012), pp. 1–31.

[20] Biplav Srivastava et al. "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*. 2007, pp. 2016–2022.

[21] Eric Timmons et al. "Reactive Model-based Programming of Micro-UAVs". In: *ICAPS Demo Track*. 2015.

[22] Sung Wook Yoon, Alan Fern, and Robert Givan. "FF-Replan: A Baseline for Probabilistic Planning". In: *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling, ICAPS 2007, Providence, Rhode Island, USA, September 22-26, 2007*. 2007, p. 352.