Automatic Generation of Flexible Plans via Diverse Temporal Planning

Yotam Amitai, Ayal Taitler, Erez Karpas
Technion Israel Institute of Technology
yotama@campus.technion.ac.il, {ataitler, karpase}@technion.ac.il

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 is dangerous, while accounting for all possible outcomes is often computationally infeasible. One approach, which lies between ignoring the uncertainty completely and addressing it completely is using flexible plans with choice, formulated as Temporal Planning Networks (TPNs). This method has been successfully demonstrated to work in human-robot teamwork 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 present a novel method for adapting an existing diverse planning method, based on top-k planning, to the temporal setting. We then describe how merging diverse plans into a single TPN is performed 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

Operating in the real world necessitates dealing with uncertainty at some degree, be that it stems from a dynamic environment, or even simply from the need to work alongside a human counterpart. Several approaches to dealing with this uncertainty exist in the literature. 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 deadends. 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. Alas, 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 *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 little use of them in the field, mostly due to the complexity of their construction. So far, generating a TPN was only possible by manually encoding them with the help of a domain expert. 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 itself must still be manually written, leaving us right where we started.

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). This work focuses on generating TPNs and leaves advocating their efficiency (compared to contingent planning or replanning methods for instance) to other papers. In this work we assume deterministic dynamics and propose a method for dealing with uncertainty in decision-space.

We describe an approach based on merging diverse plans. Unfortunately, none of the existing diverse planners (Bryce 2014; Nguyen et al. 2012; Srivastava et al. 2007; Katz and Sohrabi 2019) are capable of handling temporal planning, so we first establish and describe the first diverse temporal planner, based on adapting the top-k based diverse planner (Katz and Sohrabi 2019) to the temporal setting. Next, we

identify sets of points along different plans (denoted as time points) which *can* be merged together, creating passages between the diverse plans. Lastly, we choose which of these time 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.

In this work we attempt to create the smallest possible TPN. This is achieved by maximizing the number of points our process merges along the different plans. Smaller TPNs encode all the original plans more compactly, reduce complexity by lowering the number of elements in the TPN, and intuitively, are easier to communicate to a human counterpart. In addition, maximizing merges also results in prioritizing encompassing more possible transitions between plans, allowing the overall TPN more freedom of choice between them.

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 is the set of Boolean propositions, s.t S, the set of all possible states is then all the subsets of F. A is 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 $\text{dur}_{max}(a)$, follow $0 \leq \text{dur}_{min}(a) \leq \text{dur}_{max}(a)$. 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), and invariant condition $\text{inv}(a) \subseteq F$ which must hold during the whole execution of a. $I \subseteq F$ is the initial state, specifying exactly which propositions in F are true at time zero. $G \subseteq F$ is 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 (IHs) 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 $T(s, \Sigma)$. From a TPS point of view (i.e. ignoring temporal constraints on action durations), a 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 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; Katz and Sohrabi 2020). 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 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 decision 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* decisions, determined by the executive at run time. V_U are the *uncontrollable* decision variables, whose decisions are determined by the human or environment, rather than the executive.
- $\mathcal{E}$: The set of notable time points (Events). Each $e \in \mathcal{E}$ is associated with a conjunction of decision 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 (Episodes). 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 decision 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 relate to the durative actions of the underlying PDDL domain.

We note that in this work we only generate TPNs, i.e. no uncontrollable decisions are assigned. Determining which de-

cisions are uncontrollable is a subject for future work.

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 (events) $\mathcal{E}$ as these are the elements we wish to minimize in order to achieve the smallest TPN. Merging time points creates 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 (i.e. invalid plans), so we do not address how these merges may affect the TPN's temporal constraints 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), ordered by time.*

Given a TPS π and an IH $a \in \pi$, the TPS suffix of π from a, denoted Σ^{π}_{a}, 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 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).

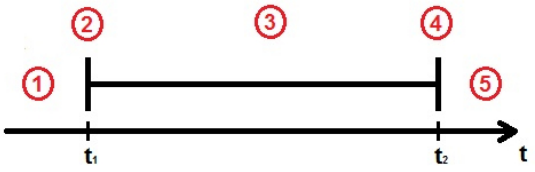

Figure 1: All the possible points to deviate from TPS π, from the point of view of the i^{th} action in π, a_i.

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 instead of dealing with the durative actions, we deal with the IHs $a_{\vdash}$ and $a_{\dashv}$. 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 for $i-1$ IHs, 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 durative actions, 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:

$$a^0 = \big\langle pre_\vdash(a), eff_\vdash(a) \cup \{p\}, inv(a), pre_\dashv(a), eff_\dashv(a)\big\rangle$$

$$a^1 = \big\langle pre_\vdash(a) \cup \{p\}, eff_\vdash(a), inv(a), pre_\dashv(a), eff_\dashv(a)\big\rangle$$

$$a^2 = \big\langle pre_\vdash(a) \cup \{\neg p, \neg p_{i-1}\}, eff_\vdash(a) \wedge p, inv(a), pre_\dashv(a), eff_\dashv(a)\big\rangle$$

$$a^3 = \big\langle pre_\vdash(a) \cup \{\neg p, p_{i-1}\}, eff_\vdash(a) \wedge \neg p_{i-1} \wedge p_i, inv(a), pre_\dashv(a) \cup \{p\}, eff_\dashv(a)\big\rangle$$

$$a^4 = \big\langle pre_\vdash(a) \cup \{\neg p, p_{i-1}\}, eff_\vdash(a) \wedge \neg p_{i-1} \wedge p_i, inv(a), pre_\dashv(a) \cup \{\neg p, \neg p_{i'-1}\}, eff_\dashv(a) \wedge p\big\rangle$$

$$a^5 = \big\langle pre_\vdash(a) \cup \{\neg p, p_{i-1}\}, eff_\vdash(a) \wedge \neg p_{i-1} \wedge p_i, inv(a), pre_\dashv(a) \cup \{\neg p, p_{i'-1}\}, eff_\dashv(a) \wedge \neg p_{i'-1} \wedge p_{i'}\big\rangle$$

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

We now explain 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.*[1]

Proof. 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. We define a backward mapping between Π' and Π and show the inclusion.

[1]Full proofs supplied in supplementary file.

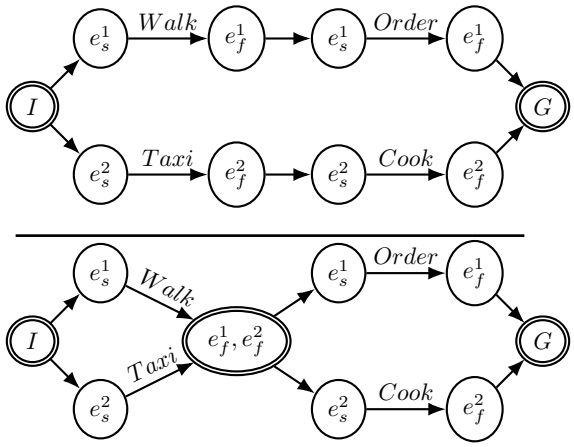

Figure 2: Additional paths created as a result of a merge. Choice time points are depicted with double circles. Top: Two separate plans, number of possible paths: 2. Bottom: Two merged plans, number of possible paths: 4.

For the second direction, we show that every solution of Π is included in Π', excluding π. We make use of a forward mapping and show the inclusion in the forward direction. □

Merging Diverse Plans into a TPN

Now that we have obtained a set of diverse TPSs $\{\pi_1 \dots \pi_k\}$ 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 approach 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 diverging into the k constituent plans in parallel to one another, up until they converge again at the end, 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. We note here that although the naive TPN is seemingly uninteresting, it is always possible to construct given that we obtained k solutions.

It is convenient to think of a TPN as a graph where time points are nodes (events) connected by constraints (episodes). By merging time points, additional solutions can be created, as demonstrated by the example in Figure 2.

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.

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. We portrayed this in the following example:
Consider the merge demonstrated in Figure 2, it is possible that the time points e_f^1, e_f^2 share the same state predicates, as arriving home may be a landmark, and the rest of the solution does not have to be dependent on the method of arrival. Whereas there could be a predicate denoting if money has been spent, whereas the time points would not have an equal state representation. This highlights how sensitive state-space comparison can be to the way each domain is constructed, hence, state-space comparison was abandoned.

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 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 compatible:*

- ***Fully** 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** 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 the goal is achieved by at least one of the TPS suffixes from the other's current state s_a.

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, merging 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 (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? This is an instance of the minimum clique cover problem. However, in future work we plan to introduce more constraints and change the cost function to account for additional preferences such as human decision models and thus we require a more flexible encoding.

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 Constraint Optimization Problem (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 by merging *groups* of IHs. 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 index**: $s_i \neq i \implies m_{i,s_i}$ 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 the description of our process, we now provide claims regarding it's soundness and completeness.

Proposition 1 (Soundness)**.** *Any TPN returned by our technique is valid and contains at least the k original plans.*

Proof. Our technique starts off by generating the naive TPN and then advances by initiating merges. These, can change the number of plans the TPN represents, but never hinder the original k solutions. Therefore, any TPN returned contains at least the original k plans. □

Proposition 2 (Completeness)**.** *Given a complete planner, our technique is also complete.*

Proof. *Given that at least a single plan has been found by the planner, the naive TPN can always be generated by merging the initial and terminal time points of all found plans. Thus, our technique is complete.* □

Granting these, We note that our technique is not optimal.

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 previously described, 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 (Schulte, Tack, and Lagerkvist 2019) to solve it. As output, we receive a mapping from time point pairs to merge operations resulting in a 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 $\{2, 4, 8\}$, both compatibility methods (Full & Semi denoted as F,S) and merging transitivity (Strict & Loose denoted as St,L). Thus, for each planning 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 30min for the COP task.

We evaluated our approach on 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 TPNs do not support such instances. From the remaining, we only display domains where OPTIC was able to retrieve multiple diverse solutions for more than a single problem[2]. We mention here that although the number of viable domains is limited by OPTIC, as the solutions of OPTIC are our input, its ability to solve temporal planning problems is out of the scope of this paper.

To broaden our benchmark domains and bolster our results, we further expanded our algorithm to enable the generation of TPNs from classical planning instances. In order for the process to take a non-temporal planning problem and output a Temporal Planning Network we require that the user supply a specification for the action durations of the domain at hand. If none are supplied, our algorithm treats all actions as actions with duration 1. The significance of this extension is in its ability to generate a TPN complete with valid solutions and schedules out of a relatively easy-to-construct classical PDDL model. For this classical implementation we make use of the Fast Downward planning system (Helmert 2006) extended with the support for structural symmetries and the orbit space search algorithm (Domshlak, Katz, and Shleyfman 2015) with LM-cut heuristic (Helmert and Domshlak 2009) as was used in (Katz et al. 2018). The classical domains we present are supplied without action durations and were taken from the results of (Katz et al. 2018) where the emphasis was on domains which our parser supports (no PDDL constants, no grounded actions).

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 as a success. 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 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.

Table 1 also displays a cost analysis (CA) between planning and optimizing phases of our process. CA is the ratio between the planning duration and the average optimization duration per configuration averaged over all commonly solved problems. For instance $CA = 2$ means that planning took twice as long as solving the COP.

Table 2 reports 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: $compactness(TPN) = 1 - \frac{|\mathcal{E}_{new}|}{|\mathcal{E}_{naive}|}$. We display 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. We note here that the attainable compactness varies greatly between domains and

[2]Full domain names are provided in the supplementary file.

	Successes															
#Sol	k=2						k=4						k=8			
Trans	St		L		#N	CA	St		L		#N	CA	St		L		#N	CA
Compat	F	S	F	S	-	-	F	S	F	S	-	-	F	S	F	S	-	-
Ba(20)	8	8	8	8	8	26.86	8	8	8	8	8	0.21	0	0	**8**	**8**	8	0.21
De(22)	9	9	9	9	9	234.73	7	7	7	7	12	0.07	8	8	8	8	9	0.08
DL(13)	12	12	12	12	12	5.17	12	12	12	12	12	0.09	12	12	12	12	12	$6e^{-3}$
El(30)	22	22	22	22	22	55.37	22	22	22	22	22	0.55	14	14	**22**	**22**	22	0.14
Fl(20)	16	16	16	16	16	12.68	16	16	16	16	17	0.12	4	4	**16**	**16**	16	0.13
Gr(20)	20	20	20	20	20	0.09	8	8	**13**	**13**	20	$9e^{-4}$	3	3	**7**	6	20	$3e^{-4}$
Mi(150)	**141**	93.	108	84.	141	0.36	77	76	**112**	106	142	0.14	35	37	71	**72**	141	$8e^{-4}$
Mo(30)	29	29	29	29	30	0.76	30	30	30	30	30	0.46	**30**	**30**	21	29	30	$2e^{-3}$
My(30)	15	15	15	15	21	426.31	15	15	15	15	20	100.14	14	14	14	14	22	7.93
NM(15)	12	12	12	12	12	9.13	12	12	12	12	12	0.04	12	12	12	12	12	0.01
Pe(30)	28	28	28	28	28	4.06	28	28	28	28	28	0.1	27	27	**28**	**28**	28	0.02
Sc(30)	12	**15**	12	**15**	16	37.32	16	16	16	16	16	9.34	16	16	16	16	16	0.05
So(30)	13	13	**14**	**14**	20	0.03	5	4	12	**13**	20	$9e^{-4}$	1	1	**2**	0	15	-
St(30)	17	17	17	17	17	30.38	17	17	17	17	17	0.79	16	16	**17**	**17**	17	0.1
Tp(30)	7	7	7	7	7	4.09	7	7	7	7	7	0.28	5	5	**7**	6	7	0.02
Ts(30)	9	9	9	9	11	21.54	11	11	11	11	11	1.43	5	5	**9**	**9**	9	0.05
Vis(20)	7	**9**	7	**9**	14	61.4	11	11	11	11	14	0.76	9	9	9	9	16	0.8
Ze(20)	12	12	12	12	13	28.9	13	13	13	13	13	1.06	**12**	11	**12**	**12**	13	0.05
Total C	**389**	346	357	338	417	-	315	313	**362**	357	421	-	223	224	291	**296**	413	-
CP(30)	5	5	5	5	25	$2e^{-4}$	5	5	5	5	25	$6e^{-4}$	2	0	**4**	2	22	$3e^{-3}$
Pa(10)	8	8	8	8	9	972	9	9	9	9	9	1216	6	6	6	6	6	558
QC(10)	5	5	5	5	8	0.84	5	5	5	5	6	5.25	5	5	5	5	5	$1e^{-3}$
Tr(10)	8	8	8	8	10	37.68	**10**	09	**10**	**10**	10	16.99	8	8	**9**	**9**	10	0.07
TO(20)	2	2	2	2	2	1116	1	1	1	1	1	0.19	0	0	**1**	0	1	-
Total T	28	28	28	28	54	-	**30**	29	**30**	**30**	51	-	21	19	**25**	22	44	-

Table 1: Summary of Empirical Success Results

is a function of their inherent structure, this in turn is why the average at the bottom of the table is of little relevance for any specific domain. the table supplies both the average for successful runs Avg_S and for overall runs Avg_O for both Temporal (T) and Classical (C) domains

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. On the other hand, an increase in compatible pairs also inflates our optimization's search space thus raising complexity and memory consumption, meaning finding a good solution in the time limit becomes more challenging.

As intuition suggests, for larger k the number of successes decreases. However, on the other extreme, we notice specific domains where low k also results in such a decrease. With smaller ks there is greater probability that the 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 can be seen in the parking (Pa) and trucks (Tr) domains, where using $k = 4$ resulted in more success than either $k = 2$ or $k = 8$.

In addition, Table 1 can also highlight for us major structural differences between domains. Such an example can be seen in the crewplanning domain (C-P) when comparing the number of problems the the planner was able to obtain diverse solutions for vs. the number of solutions for the configurations. The cause of this delta is due to the fact that no possible merges were available for most of the problems in the domain. This can suggest that either the diverse solutions in this domain vary greatly or that the order of the actions is crucial such that no merges result in a valid solution.

We now turn out attention to examining the differences between the four configurations of our approach. First, we 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, as it uses a higher proportion of the original solutions' time points.

Comparing strict transitivity to loose transitivity, the former applies 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 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.

	Compactness														
#Sol	k=2					k=4					k=8				
Trans	St		L		#P	St		L		#P	St		L		#P
Compat	F	S	F	S	-	F	S	F	S	-	F	S	F	S	-
Ba(20)	.48	.48	.48	.48	8	.5	.5	**.62**	.58	8	-	-	-	-	0
De(22)	.45	.45	.45	.45	9	**.69**	**.69**	.61	.61	7	**.52**	**.52**	.31	.31	8
DL(13)	.44	.44	.44	.44	12	**.67**	**.67**	.65	.65	12	.63	**.64**	.50	.50	12
El(30)	.45	.45	.45	.45	22	**.69**	**.69**	.64	.64	22	**.59**	.58	.36	.37	14
Fl(20)	.47	.47	.47	.47	16	.45	.45	**.7**	**.7**	16	**.43**	**.43**	.25	.26	4
Gr(20)	.48	.48	.48	.48	20	.50	**.51**	.50	.50	8	**.66**	**.66**	.22	.42	3
Mi(150)	**.46**	.43	**.46**	.44	84	.52	.54	**.60**	.41	76	.61	**.63**	.37	.29	35
Mo(30)	.36	**.43**	.36	**.43**	29	.58	**.69**	.58	**.69**	30	.72	**.84**	.69	.80	21
My(30)	.33	.33	.33	.33	15	.59	.59	.59	.59	15	**.77**	**.77**	.75	.75	14
NM(15)	.44	.44	.44	.44	12	.68	.68	.68	.68	12	**.64**	**.64**	.41	.40	12
Pe(30)	.38	.38	.38	.38	28	**.57**	**.57**	.56	.56	28	.52	**.53**	.38	.38	27
Sc(30)	**.42**	.41	**.42**	.41	12	.58	.61	.56	**.66**	16	.68	**.75**	.45	.62	16
So(30)	.41	.41	**.42**	**.42**	13	**.55**	**.55**	.51	.41	4	-	-	-	-	0
St(30)	.31	.31	.31	.31	17	**.54**	**.54**	.52	.52	17	**.58**	.57	.38	.38	16
Tp(30)	.42	.42	.42	.42	7	.62	.61	**.67**	**.67**	7	**.71**	**.71**	.34	.35	5
Ts(30)	.38	.38	.38	.38	9	**.59**	**.59**	.57	.57	11	**.70**	**.70**	.47	.47	5
Vis(20)	**.34**	.31	**.34**	.31	7	.39	**.49**	.38	.48	11	.42	**.62**	.41	.47	9
Ze(20)	.41	.41	.41	.41	12	.61	.61	.61	.61	13	.63	**.67**	.47	.47	11
Avg_S C	.42	.42	.42	.42	332	.56	.58	**.59**	.56	313	.62	**.65**	.45	.46	212
Avg_O C	.33	.33	.33	.33	417	.42	**.44**	**.44**	.42	421	.32	**.33**	.23	.24	413
CP(30)	.25	**.29**	.25	**.29**	5	**.18**	.17	**.18**	**.18**	5	-	-	-	-	0
Pa(10)	.04	.04	.04	.04	8	.09	**.10**	.09	**.10**	9	**.12**	**.12**	.10	.11	6
QC(10)	.15	.15	.15	.15	5	.09	.09	.09	.09	5	**.32**	**.32**	.31	**.32**	5
Tr(10)	.05	**.08**	.05	**.08**	8	**.07**	**.07**	.06	**.07**	9	**.07**	.06	.03	.03	8
TO(20)	.03	.06	.03	**.07**	2	**.16**	.05	.02	.02	1	-	-	-	-	0
Avg_S T	.10	**.12**	.10	**.12**	28	**.10**	**.10**	.09	**.10**	29	**.15**	**.15**	.13	.13	19
Avg_O T	.05	**.06**	.05	**.06**	54	**.06**	**.06**	.05	**.06**	51	**.07**	.06	.05	.06	44

Table 2: Summary of Empirical Compactness Results

Summary and Future Work

We have presented the first approach for automatically generating a TPN 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. We 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 plan to 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.

Finally, the objective we optimized here, minimizing the size of the resulting TPN, is only one possible objective. In the human-robot teamwork context, one may want to optimize for human-focused metrics, such as the ease of explaining the TPN to the human, 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.

Acknowledgements

This work was supported by funding from the Israeli ministry of Science and Technology.

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.

Domshlak, C.; Katz, M.; and Shleyfman, A. 2015. Symmetry breaking in deterministic planning as forward search: Orbit space search algorithm. Technical report, Technical Report IS/IE-2015-03, Technion, Haifa.

Fox, M.; and Long, D. 2003. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *J. Artif. Intell. Res.* 20: 61–124.

Helmert, M. 2006. The Fast Downward Planning System. *J. Artif. Intell. Res.* 26: 191–246.

Helmert, M.; and Domshlak, C. 2009. Landmarks, Critical Paths and Abstractions: What's the Difference Anyway? In Gerevini, A.; Howe, A. E.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the 19th International Conference on Automated Planning and Scheduling, ICAPS 2009, Thessaloniki, Greece, September 19-23, 2009*. AAAI.

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.; and Sohrabi, S. 2020. Reshaping Diverse Planning. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, 9892–9899. AAAI Press.

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.

Schulte, C.; Tack, G.; and Lagerkvist, M. Z. 2019. Modeling. In Schulte, C.; Tack, G.; and Lagerkvist, M. Z., eds., *Modeling and Programming with Gecode*. Corresponds to Gecode 6.2.0.

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.