Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling

Concurrent Plan Recognition and Execution for Human-Robot Teams

Steven J. Levine and **Brian C. Williams**
MIT Computer Science and Artificial Intelligence Laboratory
32 Vassar St., Cambridge, MA 02139
{sjlevine, williams}@mit.edu

Abstract

There is a strong demand for robots to work in environments, such as aircraft manufacturing, where they share tasks with humans and must quickly adapt to each other's needs. To do so, a robot must both infer the intent of humans, and must adapt accordingly. The literature to date has made great progress on these two tasks - recognition and adaptation - but largely as separate research activities. In this paper, we present a unified approach to these two problems, in which recognition and adaptation occur concurrently and holistically. Key to our approach is a task representation that uses choice to represent alternative plans for both the human and robot, allowing a single set of algorithms to simultaneously achieve recognition and adaptation. To achieve such fluidity, a labeled propagation mechanism is used where decisions made by the human and robot during execution are propagated to relevant future open choices, as determined by causal link analysis, narrowing the possible options that the human would reasonably take (hence achieving intent recognition) as well as the possible actions the robot could consistently take (adaptation). This paper introduces Pike, an 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). We evaluate Pike's performance and demonstrate it on a household task in a human-robot team testbed.

Introduction

There is a strong demand for robots to work in environments where they share tasks with humans. These tasks include aircraft manufacturing, household chores such as cooking and cleaning, medical care, and countless others. In all of these situations, the robots must be able to both infer the intent of humans, and must adapt accordingly.

The literature to date has made great progress on these two areas - recognition and adaptation - but largely as separate research activities. Numerous approaches to plan recognition have been proposed, as noted in (Carberry 2001). However, these approaches generally focus solely on recognition, not on how a robotic agent should adapt once the human's intent is inferred. There has similarly been a large body of work on adaptation. Most, however, do not incorporate plan recognition. This work presents a unified approach to these two problems, providing recognition and adaptation simultaneously through a single set of algorithms.

To achieve unified recognition and adaptation, we use a plan representation containing controllable choices for the robot and uncontrollable choices made by the human. These choices are not independent; rather, they are highly coupled through the plan's state and temporal constraints. Our approach exploits these interconnections to reason over consistent sets of decisions, allowing the executive to determine which courses of action the human would logically take (intent recognition) to maintain plan consistency, and additionally ascertaining which actions the robot should take (adaptation). During an offline compilation stage, labeled causal links are extracted from the contingent plan. These causal links are then transformed into constraints over consistent sets of choices, and subsequently encoded into an Assumption-based Truth Maintenance System (ATMS), along with extracted temporal conflicts. During execution, the executive quickly reasons over consistent sets of choices using this ATMS, committing to decisions as appropriate. As execution proceeds and the human and robot both make choices, the results of these decisions are propagated by the ATMS to other open choices, and hence their domains are pruned. Additionally, violated causal links (resulting from unmodeled disturbances) may further constrain future choices and hence influence plan recognition and adaptation. In this way, the executive reasons over choices made in the past, action preconditions, temporal constraints, and unanticipated disturbances.

Our innovations are threefold. First, we introduce a novel, unified approach to plan recognition and adaptation that uses a human-robot plan representation with choice. Second, we generalize the notion of causal links to contingent, temporally flexible plans and present a set of algorithms for automatically extracting them from said plans. Finally, we extend a state-of-the art dynamic execution system designed for temporal constraints to also make fast online choices based on state, thereby achieving concurrent recognition and adaptation.

Related Work

There is a rich literature on plan adaptation. Beginning with temporally-flexible plans such as Simple Temporal Networks (STNs), efficient dispatchers have been developed that perform fast, online, least-commitment scheduling (Dechter, Meiri, and Pearl 1991; Tsamardinos, Muscettola, and Morris 1998). Later approaches introduced uncertainty and uncontrollability into these models, resulting in executives that could adapt to many different types of temporal disturbances (Effinger et al. 2009; Vidal 1999). These systems focus on adapting to temporal rather than state constraints, and aren't designed for recognizing human plans.

A number of adaptation techniques do however recover based on violated state constraints. Many such systems have focused on integrated planning and execution, such as IxTeT eXeC (Lemai and Ingrand 2004), ROGUE (Haigh and Veloso 1998), IPEM (Ambros-Ingerson and Steel 1988), and HOTRiDE (Ayan et al. 2007). Some focus on planing at reactive time scales or continuously online (Finzi, Ingrand, and Muscettola 2004; Chien et al. 2000). The Human-Aware Task Planner (HATP), combined with SHARY, executes human-robot tasks and replans as needed (Alili et al. 2009; Clodic et al. 2009). The TPOPExec system generalizes plans and at every time point will try to execute the first action of a subplan consistent with the current state, thus minimizing the need for replanning (Muise, Beck, and McIlraith 2013). These systems generally focus on planning, execution with plan repair, and/or responding to disturbances that aren't explicitly modeled as human intent. We differ in that we reason explicitly over the possible choices a human is likely to make, and adapt accordingly.

The Chaski executive performs fast online task reallocation for human-robot teams with temporal constraints (Shah, Conrad, and Williams 2009). Chaski is capable of inferring if an agent - human or otherwise - is stuck, and will reallocate tasks for the other agents appropriately to maintain plan consistency. Chaski focuses on the dynamic task assignment problem, while we take a different approach and focus instead on plan recognition and adaptation by reasoning over explicit choices in the plan.

Drake is an executive capable of executing temporally flexible plans with choice (such as TPNs), scheduling and making choices online with low latency and using minimal space overhead (Conrad, Shah, and Williams 2009). It responds dynamically to temporal disturbances, both in terms of scheduling, and also in terms of making discrete choices that are guaranteed to be feasible given those temporal disturbances. This is accomplished by fusing ideas from temporal planning and from the ATMS. We borrow many techniques from Drake in this work, described shortly.

Many approaches to plan recognition have been proposed, just a few of which are (Kautz and Allen 1986; Bui 2003; Avrahami-Zilberbrand, Kaminka, and Zarosim 2005; Goldman, Geib, and Miller 1999). Most of these approaches, however, perform intent recognition non-interactively. That is, they do not attempt to interact with the human whose intent is being recognized. Our approach differs in that we perform execution while concurrently monitoring the human.

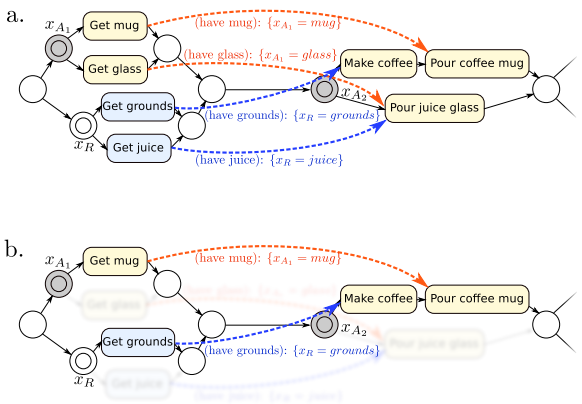

Figure 2: Kitchen example. Part A. shows the plan annotated with labeled causal links. Part B shows the resulting execution if Alice chooses $x_{A_1} = mug$.

Approach in a Nutshell

Consider the grounded example shown in Figure 1 on the next page, in which Alice is making breakfast for herself with the help of her trusty robot. The left half of the plan depicts the team either making coffee (for which Alice uses a mug) or getting some juice (for which Alice uses a glass), while the right half depicts either making a bagel with cream cheese or getting some cereal and milk. Alice is running late for work, so she imposes an overall temporal constraint that both her food and drink must be ready within 7 minutes.

Consider just the first half of the plan, in which the team prepares a beverage - either coffee or juice. There are three decisions - at first, Alice choose to either get a mug or a glass for her beverage. Shortly thereafter and in parallel, the robot chooses to either fetch the coffee grounds or juice from the refrigerator. Finally once all the necessary supplies have been retrieved, Alice will either make coffee and pour it into her mug, or pour the juice into her glass, depending on her preferred beverage this morning.

Key to our approach is noting that these three choices - two uncontrollable made by Alice and one controllable made by the robot - are not independent. Rather, they are tightly coupled through state constraints in this example, and more generally through temporal constraints as well. For example, if the robot chooses to get the juice out of the refrigerator, Alice will not be able to make coffee in her second choice since she won't have the coffee grounds. Such interrelationships are illustrated in Figure 2a., where we have defined three variables, x_{A_1}, x_R, and x_{A_2}, to represent the different choices. We've also annotated the plan with *labeled causal links*, denoted by dotted arcs and labeled with the environment under which they hold. These causal links capture precondition state requirements, and their environments imply constraints over feasible choice assignments.

Consider the example execution depicted in Figure 2b., in which Alice chooses to get the mug instead of the glass for her choice (i.e., $x_{A_1} = mug$). Alice will therefore have a mug (necessary for pouring coffee into the mug in her sec-

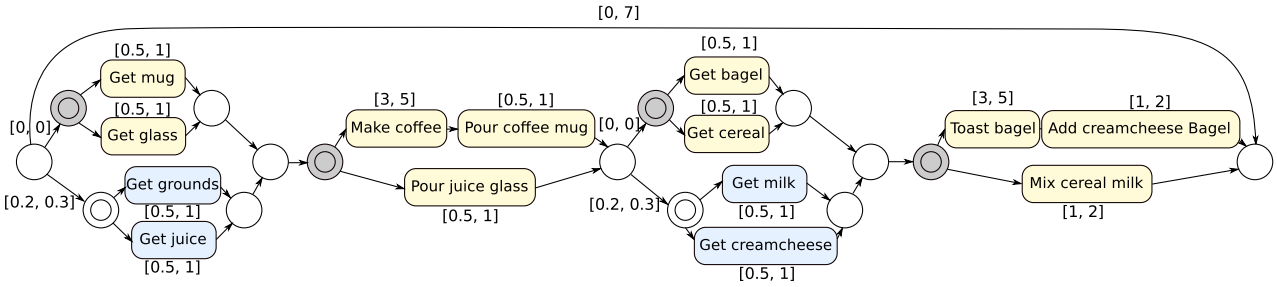

Figure 1: A plan network for making breakfast. Circles denote events, double circles denote controllable choices (made by the robot), and shaded double circles denote uncontrollable choices (made by Alice). Yellow boxes denote activities for Alice, while blue boxes are targeted at the robot. Note that each activity is represented with a start and end event, but illustrated here as a box for compactness. All unlabeled temporal constraints are $[0, \infty]$ ordering constraints.

ond choice), but will not have a glass (necessary for pouring juice into the glass). The top-most causal link, (have mug) contingent upon $x_{A_1} = mug$, will hold, yet the causal link (have glass) contingent upon $x_{A_1} = glass$ will not hold. Thus, Alice cannot choose $x_{A_2} = juice$, since the pour juice action would have a failed precondition. In this way, the robot infers that Alice's intent must be to make coffee (i.e., $x_{A_2} = coffee$) since she picked up a mug and not a glass.

Similar reasoning allows the robot to adapt to Alice's intent. Given that Alice is making coffee, she'll require the coffee grounds - illustrated by the causal link (have grounds) contingent upon $x_R = grounds$. Since this is the only causal link justifying said precondition, the robot infers that it must choose $x_R = grounds$ in order for the plan to be complete. Thus, the robot adapts by getting the grounds instead of the juice, resulting in the final execution depicted in Figure 2b.

If we step back and consider the larger plan shown in Figure 1, we note that Alice cannot both make coffee and make a bagel because she would run out of time. The minimum time required to make coffee and toast a bagel is more than 7 minutes. Thus, should Alice choose to grab a mug at the beginning, her intent must be not only to make coffee, but also to choose the less time-consuming cereal option so she'll arrive at work on time. By similar causal link analysis, the robot will adapt by getting milk for her cereal instead of cream cheese for a bagel.

Consider one final case, in which Alice at first chooses $x_{A_1} = glass$. She'll have enough time later for either a bagel or cereal. However, suppose while pouring her juice, an unexpected disturbance occurs: the toaster oven breaks. In this case, a causal link justifying her making a bagel has been dynamically violated at run time. Alice's only option, therefore would be to make cereal. Pike would detect this unanticipated change in world state, and instantaneously infer Alice's refined intent and adapt accordingly.

We have just illustrated how a single algorithm, based on labeled constraint propagation, concurrently achieves plan recognition and adaptation given state and temporal constraints as well as disturbances. We note that in this specific example, Alice's intent and the robot's adaptations were completely determined after she picked up the mug. Such is not generally the case however - often, there may still be multiple (though fewer) consistent options for future choices after constraint propagation. In these cases, further decisions, either by human or robot, are necessary to hone in on a single intent and adaptation scheme.

Problem Statement

As illustrated above, Pike takes in a contingent plan, as well as sensory inputs in the form of recognizing Alice's choices (ex., picking up mug vs. glass) and state estimates (for detecting unanticipated disturbances). It outputs, in real time, decisions for the robot, as well as a schedule for its actions such that the plan is expected to succeed. This section discusses these inputs and outputs in greater depth.

Pike's input contingent plan format is the Temporal Planning Network under Uncertainty (TPNU), which has roots in the Simple Temporal Network (STN) (Effinger et al. 2009; Dechter, Meiri, and Pearl 1991). STNs uses set-bounded, simple temporal constraints to relate events, which represent instantaneous time points (Dechter, Meiri, and Pearl 1991). A simple temporal constraint $[a, b]$ between events e_x and e_y implies that $t_{e_y} - t_{e_x} \in [a, b]$. An extension to the STN is the TPN, or Temporal Planning Network, which adds choice events between different threads of execution as well as executable activities (Kim, Williams, and Abramson 2001). The TPNU further builds off the TPN, introducing uncontrollable choice variables made by the environment (in our case, the human), not by the executive (Effinger et al. 2009). TPNUs and TPNs are thus contingent plan representations that compactly encode many possible candidate subplans. Figure 1 shows a TPNU, and Figure 2b shows a candidate subplan.

Each activity in a TPNU has an associated start and end event, as well as an action. For our purposes, these actions are grounded PDDL operators (Fox and Long 2003). Both the start and end events are treated as having preconditions and effects, as determined by the "at start" and "at end" conditions and effects defined by the PDDL operator.

Given all the assignments to discrete choices in the TPNU, as well as a schedule of when all event should be

executed, we have a subplan with no ambiguity. This subplan is said to be consistent if the schedule satisfies all of the simple temporal constraints, and complete if the preconditions of all events are satisfied when the event is executed.

We are now equipped to define Pike's problem statement. Pike takes as input: 1.) a TPNU, 2.) the world initial and goal states (sets of PDDL predicates), 3.) a continual stream of state estimates (sets of predicates), and 4.) a continual stream of time assignments and outcomes to the uncontrollable choices in the TPNU.

Given this input, Pike outputs 1.) a stream of choice assignments to the TPNU's controllable variables, and 2.) a dispatch of the TPNU's events, such that there is at least one complete and consistent candidate subplan possible after each of Pike's choices.

Approach

Our approach is as follows. First, we infer ordering constraints over events in the plan from its temporal constraints. This is accomplished by computing a labeled all-pairs shortest path (APSP) on the contingent plan. The resulting table allows us to efficiently query the shortest temporal distance between any two events, given a partial set of choices. We use a labeled value set (LVS), to be introduced shortly, to store this information compactly.

Next, using these ordering relations, we extract a minimal, dominant set of labeled causal links from the plan, again using an LVS. We also find threats in the plan, and infer constraints that resolve them.

Finally, once all the causal links and threat resolutions are extracted, they are encoded into an ATMS-based knowledge base. This involves constraint propagation, and results in a precomputed and compactly recorded database of candidate subplans that could possibly be complete and consistent during execution.

Online, the executive uses this knowledge base to schedule events and make choices in real time. As execution unfolds, large swaths of candidate subplans are pruned out based on choices made or unexpected disturbances.

Offline Compilation

We now present the offline compilation algorithm, which take as input a temporally-flexible plan with choice (i.e., a TPNU), the initial and goal world states, and outputs an ATMS-based knowledge base suitable for fast online execution. An overview is provided in Algorithm 1, and pertinent piece are discussed in the following sections.

Algorithm 1: Offline compilation

Input: TPNU plan, initial and goal states
Output: Knowledge base, dispatchable plan

```
Initialize 𝕃, 𝕋 as empty sets
Convert TPNU plan to labeled distance graph G
d_{e_i,e_j} ← compute labeled APSP of G
𝒟, ℂ ← compute dispatchable form, temporal conflicts
foreach precondition p of each event e_c in plan do
    ℒ, 𝒯 ← GetCausalLinksAndThreats(p, e_c)
    Add ℒ, 𝒯 to 𝕃, 𝕋
end
kb ← EncodeInATMS(𝕃, 𝕋, ℂ)
return (𝒟, kb)
```

Labeled APSP

To describe the labeled APSP, we first introduce relevant background borrowed from ATMSs and the Drake executive. We associate a discrete variable with each choice in the contingent plan. An *environment* is a set of assignments to these variables, denoted for example as $\{x_{A_1} = mug, x_R = grounds\}$. We say that an environment ϕ_a *subsumes* another environment ϕ_b if all assignments in ϕ_a are also contained in ϕ_b (De Kleer 1986). A *complete environment* assigns a value to all variables, and hence corresponds to a specific candidate subplan where all choice are made.

An environment ϕ that is not complete intuitively represents a set of candidate subplans - namely, all the candidate subplans that are subsumed by ϕ. For example, $\{x_{A_1} = mug\}$ can be thought of as representing $\{x_{A_1} = mug, x_R = grounds, ...\}$ and many more. In this way, environments are used to compactly represent large swaths of candidate subplans. They are used extensively in our offline and online algorithms for efficient reasoning.

We next briefly describe the *labeled value set* (LVS), which builds off the concept of subsumption and environments and was introduced by Drake (Conrad, Shah, and Williams 2009). An LVS intuitively represents the "tightest" values for some condition as a function of the environment (Conrad 2010). For example, suppose we wish to encode the constraint $x < a$, for some value of a that is environment-dependent. We could represent a by the LVS $\{(2, \{x = 1, y = 2\}), (3, \{x = 1\}), (6, \{\})\}$. We may then *query* it under different environments with the Q operator. For example, we may ask $Q(\{x = 1, y = 2\})$, meaning "what is the smallest value for a where $x < a$, over all environments represented by $x = 1, y = 2$?" We could say that $x < 3$, since $\{x = 1\}$ subsumes $\{x = 1, y = 2\}$, but the LVS also permits the tighter bound $x < 2$ under this environment. If, however, we query the LVS for the tightest bound on x over all environments represented by $x = 1$, the LVS can only guarantee $x < 3$.

A labeled value (a_i, ϕ_i) *dominates* another labeled value (a_j, ϕ_j) under relation R (ex., such as $<$) if $a_i \ R \ a_j$ and ϕ_i subsumes ϕ_j. For example, with R as $<$, we can say $(1, \{x = 1\})$ dominates $(2, \{x = 1, y = 2\})$. Intuitively, this means that we have derived a "tighter" bound on a value that subsumes all of the weaker bounds. However, $(1, \{x = 1, y = 2\})$ does not dominate $(2, \{x = 1\})$ since $\{x = 1, y = 2\}$ does not subsume $\{x = 1\}$. Such domination rules come into play when adding values to LVSs - only dominant pairs are added, thereby maintaining minimality (Conrad 2010).

The labeled APSP algorithm we use is similar to the one described in (Conrad 2010). Namely, it is a generalized version of the Floyd Warshall APSP algorithm that, instead of

performing operations on real numbers, performs them on labeled value sets. The output is a table d_{e_i,e_j}, where each entry is an LVS representing the shortest temporal distance between the two events (which may be queried under different environments).

Labeled Causal Links & Extraction

After computing the labeled APSP, we then proceed to extract labeled causal links.

Causal links have been used in a number of AI systems, ranging from partial order planners to execution monitoring systems (McAllester and Rosenblatt 1991; Veloso, Pollack, and Cox 1998; Lemai and Ingrand 2004; Levine 2012). Intuitively, a causal link from A to B encodes that activity A produces some effect that activity B requires to execute. Additionally, A must precede B and there may be no other event A' that also produces B's requirement (or negates it) between A and B. This leads to the concept of a threat: an activity C is said to *threaten* a causal link if it negates B's precondition and could occur between A and B.

We extend causal links to contingent, temporally-flexible plans, resulting in *labeled causal links*. Because they are crucial to reasoning about plan completeness, a key part of Pike's offline compilation stage is to extract these labeled causal links from the plan.

Suppose we have a plan in which event e_{p_1} produces a state predicate p consumed by a later-occurring event, e_c. Then, there is a causal link from e_p to e_c. Further suppose that e_{p_1}'s execution environment is $\{x = 1\}$ (that is, e_{p_1} will only be executed if $x = 1$), and that e_c's execution environment is $\{y = 1\}$. The causal link is contingent upon the producer event's environment. If the executive chooses $x = 2$, then e_{p_1} will not be executed, so the causal link is necessarily inactive. The executive may not additionally choose $y = 2$, because e_c would have a violated precondition. It must pick some $y \neq 1$ so that e_c is not executed (and its precondition p will not be required to hold). By similar logic, suppose that the executive chooses $\{y = 2\}$ initially. In this case, the consuming event e_c will not be executed, hence its requirement for p is nonexistent. The executive may therefore choose either $x = 1$ or $x = 2$ consistently. These two cases introduce the core intuition behind labeled causal link analysis: For a plan to be consistent, there must be at least one active causal link for each precondition of every active event. Otherwise, some precondition will not be met and the plan will fail.

Previous work generally considers a single causal link for each precondition of each activity in the plan. Contingent plans, however, require multiple labeled causal links, since different producers may execute in different environments and may both justify a precondition under different contexts. Continuing with the above example, suppose that we have another event e_{p_2} that produces p, but has execution environment $\{x = 2\}$. There are now two labeled causal links that justify e_c's precondition: one from e_{p_1} if $x = 1$, and another from e_{p_2} if $x = 2$. Similar reasoning to the above applies, but with a key difference: if the executive chooses $y = 1$ and e_c is hence activated, then the executive may now choose either $x = 1$ or $x = 2$ (but not, for example, $x = 3$). We

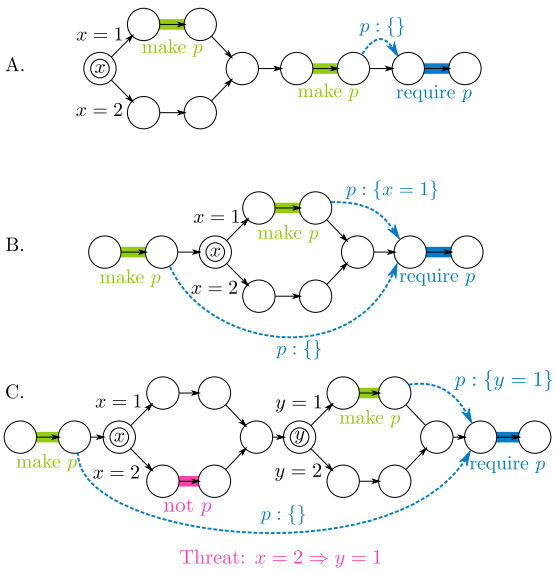

Figure 3: Labeled causal links and threats for different plans. There are three actions: `(make-p)` produces p as an effect, `(not-p)` negates p, and `(require-p)` requires p as a precondition. Labeled causal links are shown as dotted lines connecting producers to consumer events, and are labeled with the predicate and environment under which they apply.

call the set of all labeled causal links for a particular event's precondition an $\mathcal{L}$-*set*, denoted $\mathcal{L}$. Each $\mathcal{L}$-set is associated with an environment: namely, that of the consumer e_c. If the environment for e_c holds, the $\mathcal{L}$-set is activated and at least one causal link therein must be activated. This is equivalent to the implication $\phi_{\mathcal{L}} \Rightarrow \bigvee_i \phi_{l_i}$.

Figure 3 illustrates several examples of labeled causal links. It also demonstrates a key second property of labeled causal links: *dominance*. Note that in Figure 3A., there is a single causal link for the later-occurring producer, despite the existence of an earlier (make-p) action. This is because the later (make-p) action will always execute whenever the earlier one does, and is guaranteed to occur afterwards. We say that a causal link from e_{p_1} to e_c with label ϕ_1 *dominates* another causal link e_{p_2} to e_c with label ϕ_2 if: 1.) e_{p_1} occurs after e_{p_2}, and 2.) ϕ_1 subsumes ϕ_2. The later-occurring causal link in part Figure 3B. is not dominant, however, because it's environment does not subsume the earlier link's environment. In this case, the executive could choose either $x = 1$ or $x = 2$. We note however that the $x = 1$ choice is still useful as a contingency for responding to unanticipated disturbances; should p unexpectedly disappear and the first causal link be violated before the choice occurs, the executive can recover by forcing $x = 1$ to re-assert p.

While not required for correctness, Pike extracts only a minimal, dominant set of causal links from the plan. For large problems, this is important for performance. Eliminating unnecessary links significantly decreases the size of the generated ATMS knowledge base, leading to much faster

propagation and online reasoning.

Figure 3C. illustrates an example threat, now generalized to contingent temporal plans. In this example, if the executive chooses $x = 2$ then $\neg p$ will be asserted - thus threatening the earlier causal link. In this case, the only way to resolve this discrepancy is for the executive to also choose $y = 1$, which will provide a restorative action and re-assert p. Thus, threats are additionally contingent upon choices made in the plan. During compilation, Pike records the threat resolution $x = 2 \Rightarrow y = 1$.

We may now present the labeled causal link extraction process; pseudocode is shown in Algorithms 1 and 2.

Let $d_{e_b \to e_a}$ denote the shortest path (an LVS) from e_b to e_a as computed previously by the labeled APSP, and ϕ_a, ϕ_b denote the execution environments of events e_a and e_b, respectively. We say that e_a precedes e_b, denoted $e_a \prec e_b$, if the query $Q(\phi_a \cup \phi_b) < 0$ for $d_{e_b \to e_a}$. In other words, if the shortest temporal distance from e_b to e_a is negative in all environments where both e_a and e_b hold, $e_a \prec e_b$. This is a direct extension to the ordering constraint criterion presented for STNs, but extended to the labeled case (Conrad, Shah, and Williams 2009). We define the succession relation $\succ$ similarly; if $e_a \prec e_b$, then $e_b \succ e_a$. In general, it is possible that neither $e_a \prec e_b$ nor $e_b \prec e_a$. This happens if the plan's temporal constraints are loose enough to permit different orderings that are both consistent, so that a single order cannot be determined a priori.

Precedence constraints are used to ensure that producers occur before consumers. Offline compilation continues by computing the $\mathcal{L}$-sets and threat resolutions for every precondition of every event in the plan. Pseudocode for extracting these given a single precondition p of a consumer e_c is shown in Algorithm 2. We use an LVS with dominance relation $\succ$ to extract the minimal set of dominant labeled causal links. Lines 3-8 add every event e_p preceding e_c that produces either p or $\neg p$ to the this LVS (which will maintain only the latest-occurring, dominant ones). Should it be encountered that e_p produces $\neg p$ and neither $e_p \prec e_c$ nor $e_c \prec e_p$, the compilation process cannot determine beforehand if e_p is a threat. Pike will thus fail.

At this stage, the LVS contains all dominant events e_{p_i} producing p or $\neg p$. Those producing p will become labeled causal links in Lines 13-15, and those producing $\neg p$ are threats that are resolved in Lines 9-12. Each threat e_T is resolved by finding the set of all other events e_{R_i} in the LVS that succeed e_T and resolve the threat by reasserting p. If e_T is executed (i.e., if ϕ_T holds), then at least one of the resolvers must also hold - resulting in the threat resolution propositional constraint $\phi_T \Rightarrow \bigvee_i \phi_{R_i}$.

Once an $\mathcal{L}$-set with threat resolutions has been computed for all preconditions, the labeled causal link extraction process is complete.

Encoding causal links in an ATMS

The final stage of offline compilation is to encode the labeled causal links' corresponding constraints into an ATMS-based knowledge base. This ATMS will allow the online executive to quickly make fast queries regarding choice feasibility.

Algorithm 2: GETCAUSALLINKSANDTHREATS

Input: A precondition p for an event e_c
Output: An $\mathcal{L}$-set $\mathcal{L}$ and threat list $\mathcal{T}$

1 $\xi \leftarrow$ new labeled value set with relation $\succ$ over events
2 $\mathcal{L}, \mathcal{T} \leftarrow$ empty sets
3 **foreach** *event* e_p *producing* p *or* $\neg p$ *as effect* **do**
4 **if** e_p *produce* $\neg p$ *and neither* $e_p \prec e_c$ *nor* $e_c \prec e_p$ **then**
5 **return** ERROR
6 **end**
7 Add $(e_p, \text{ENV}(e_p))$ to ξ if $e_p \prec e_c$
8 **end**
9 **foreach** (e_T, ϕ_T) *producing* $\neg p$ *in* ξ **do**
10 Find all $(e_{R_i}, \phi_{R_i}) \in \xi$ where e_{R_i} produces p, $e_T \prec e_{R_i}$, and $\phi_{R_i} \cup \phi_T \neq \perp$
11 Add threat resolution $(\phi_T \Rightarrow \bigvee \phi_{R_i})$ to $\mathcal{T}$
12 **end**
13 **foreach** $(e_p, \phi_p) \in \xi$ *producing* p **do**
14 Add labeled causal link from e_p to e_c over p, with environment ϕ_p, to $\mathcal{L}$
15 **end**
16 **return** $\mathcal{L}$ and $\mathcal{T}$

The Assumption-based Truth Maintenance System, or ATMS, is a popular knowledge base that allows a problem solver to efficiently reason about facts without prematurely committing to them (De Kleer 1986). The ATMS introduces assumptions - facts whose certainty is not known and may be changed by the problem solver with little overhead.

The encoding process is illustrated in Figure 4. It begins by making a special ATMS node, $Consistent$, which holds if and only if the executed plan is complete and consistent. This $Consistent$ node will be constantly queried during online execution under different environments representing different possible choices that may be made.

The assumptions for the ATMS encoding consist of variable assignments of the form $x_i = v_i$, for all choice variables and their domains. Mutex constraints are added so that only one value for each variable is chosen (represented in Figure 4 by the left-most $\perp$ contradiction nodes).

The remainder of the nodes encode labeled causal links and threat resolutions. The key idea is that for each activated $\mathcal{L}$-set, at least one labeled causal link therein must be activated, and all threat resolution implications must hold true.

Taking the example in Figure 4A., we see that there are two labeled causal links: l_1 with environment $\phi_1 = \{\}$, and the l_2 with environment $\phi_2 = \{y = 1\}$. These are represented by ϕ_1 and ϕ_2 nodes in Figure 4B. We see that the l_1 and l_2 nodes are also connected to a $Holds$ assumption. This represents the case when causal links may be violated via unexpected disturbances. Should this occur, the executive would tie the corresponding $Holds$ assumption to a contradiction node (example shown with a dotted line in Figure 4B.), thereby ensuring that l_1 is necessarily FALSE. Note that each of the links is connected to a $Links$ node disjunctively, meaning that at least one of those links must

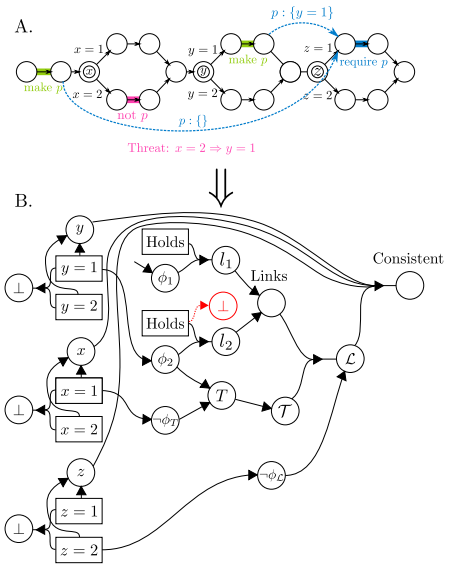

Figure 4: Example encoding of labeled causal links into an ATMS. Following the graphical notation in (Forbus 1993), assumptions are drawn as rectangles, and fact nodes as circles. Arrows represent justifications, where multiple tails are joined conjunctively. Multiple incoming arrows to a node represent a disjunction of those justifications. The $\perp$ nodes represent contradictions - if they are justified by some environment, that environment becomes a no-good.

be active for $Links$ to be TRUE. Finally, we encode the fact that this $\mathcal{L}$-set need not be active - i.e., the executive could pick $z = 2$, meaning that the (require-p) action is not active. This is represented by the $\neg\phi_\mathcal{L}$ node, where $\phi_\mathcal{L}$ represents the environment of the $\mathcal{L}$-set, or namely that of the consumer event (here $\{z = 1\}$). We wish for the implication $\phi_\mathcal{L} \Rightarrow Links \wedge Threats$ to hold - i.e., at least one causal link and all threat resolutions must be active, if the $\mathcal{L}$-set is active. To encode this, we convert the implication to $\neg\phi_\mathcal{L} \vee (Links \wedge Threats)$, and use the following to derive which assumptions to connect disjunctively to $\neg\phi_\mathcal{L}$:

$$\neg \underbrace{\bigwedge_i (x_i = v_{ik})}_{\phi_\mathcal{L}} \Leftrightarrow \bigvee_i (x_i \neq v_{ik}) \Leftrightarrow \bigvee_i \bigvee_{l \neq k} x_i = v_{il}$$

For example, if we had variables x and y each with domain $\{1, 2, 3\}$ and $\phi_\mathcal{L}$ were $\{x = 1, y = 2\}$, we could represent $\neg\phi_\mathcal{L}$ as $\neg(x = 1 \wedge y = 2)$ as $(x = 2 \vee x = 3 \vee y = 1 \vee y = 3)$. In Figure 4, this is simply $z = 2$. The threat resolutions, which also involve implications due to their form $\phi_T \Rightarrow \bigvee_i \phi_{R_i}$, use a similar encoding for $\neg\phi_T$.

Finally, temporal conflicts must be encoded into the ATMS. Pike extracts temporal conflicts using the same approach as Drake; namely, by finding environments in which negative cycles appear in the APSP (Conrad, Shah, and Williams 2009). For each conflict environment, we justify the contradiction node $\perp$ conjunctively with each of the environment's assignments to ensure that it becomes a no-good in the ATMS.

This completes the offline compilation process. Now that an ATMS-based knowledge base has been produced, it will be used for fast online execution.

Online Execution

The online execution algorithm is responsible for scheduling events and assigning values to controllable choices. It is similar in spirit to the version presented in the Drake executive (Conrad 2010). Simplified pseudocode is shown in Algorithm 3. The algorithm uses labeled execution windows for scheduling events. Additionally, controllable choices will be made only if the ATMS is capable of committing to them. For example, if the $Consistent$ node could hold when the $x = 2$ assumption holds, then the executive would not commit to $x = 2$. In this way, the executive will greedily make decisions, possibly pruning out future options, so long as at least one candidate subplan that is complete and consistent remains possible.

When the executive makes a choice (or when the human makes an uncontrollable decision), the ATMS commits to this decision $x_i = v_i$ by connecting all $x_i \neq v_i$ to $\perp$. This has the effect of forcing the choice. Similarly, should a causal link be violated, the corresponding $Holds$ assumption is tied to $\perp$.

This completes Pike's algorithmic description. We continue by providing some experimental results in simulation and on a hardware testbed.

Algorithm 3: ONLINEEXECUTION

Input: A dispatchable TPNU $\mathcal{D}$, knowledge base kb, streams of uncontrollable choice assignments $\mathcal{U}$ and state estimates $\mathcal{S}$
Output: Stream of controllable choice assignments, scheduled event times

```
Q ← all events in D
INITTIMEWINDOWS()
while Q ≠ ∅ do
    t ← CURRENTTIME()
    e ← some event in Q
    if COULDCOMMITTOEVENT(e, t, kb) then
        COMMITTOEVENT(e, kb)
        Remove now-inconsistent e_i from Q
        PROPAGATETIMEWINDOWS(e, t)
        ACTIVATECAUSALLINKS(e)
    end
    COMMITUNCONTROLLABLECHOICES(U, kb)
    l_vio ← GETVIOLATEDCAUSALLINKS(S)
    COMMITASVIOLATED(l_vio, kb)
    if NOCANDIDATESUBPLANSLEFT?(kb) then
        return FAILURE
    end
end
return SUCCESS
```

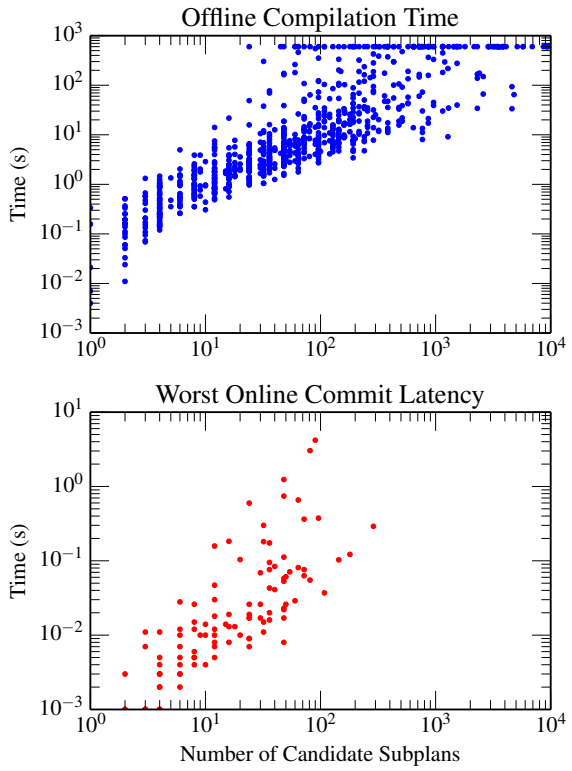

Figure 5: Top: compilation time for randomly-generated, structured TPNU's and corresponding PDDL domains. Bottom: slowest time for the ATMS to commit to a decision during online execution.

Results

We have tested Pike both in simulation and on hardware. Our results show that Pike is able to dispatch plans and usually make decisions at reactive timescales.

Figure 5 shows experimental data for Pike, as run on structured, randomly-generated TPNU's with associated PDDL domains. Each point represents an individual problem instance. To generate these problems, TPNU's with random choice, parallel, and sequential structures were generated. Causal links and threats were then randomly generated for randomly-generated action preconditions, and used to define corresponding PDDL domains.

The top plot shows offline compilation time for these problems as a function of the number of candidate subplans (which grows roughly, though not always, exponentially with the number of TPNU choice nodes). Theoretically, this offline compilation may take exponential time, as the labeled APSP is not polynomially bounded (unlike its unlabeled counterpart). The sharp line at the top is a result of our 10 minute per test time limit - these instances would have run for longer.

An additional computational complexity lies in our use of the $Holds$ assumptions for causal links - each of which adds an assumption to the ATMS. A plan with N choices (each with a domain of size d) and L causal links will result in a total of $Nd+L$ ATMS assumptions. Since the ATMS may be forced to enumerate all environments over all assumptions, the worst case is a state space explosion often dominated by L. This is why it is crucial from a performance standpoint to extract only a minimal, dominant set of causal links - it drastically reduces L, and the size of the ATMS problem.

Figure 6: Integration on a household kitchen scenario.

The bottom plot shows the highest latency for the ATMS knowledge base to commit to a decision, for those randomly-generated problems that were feasible. This is a worst-case estimate, as the mean commit time was often much faster. As execution unfolds and the ATMS prunes more candidate subplans, committing becomes much easier.

In additional to these simulated tests, we have also implemented[1] the the first half of the "breakfast" domain in Figure 1, or namely the "beverage" domain shown in Figure 2. A picture of our testbed can be seen in Figure 6.

Our integration uses a computer vision system to track the locations of the various objects (mug, glass, coffee grounds, and orange juice), which the Barrett WAM arm may then manipulate. We have also implemented a state estimator that outputs the current world state and a simple activity recognizer. If the activity recognizer sees the mug moving without the robot's intervention, it infers that the human must be executing the (get mug) task and signals to Pike that the corresponding uncontrollable choice has been made. The demo proceeds as described earlier in this paper. Namely, if the human picks up the mug, the robot will offer coffee grounds. If the human picks up the glass, the robot will offer juice.

Conclusions & Future Work

We have introduced Pike, an executive for human-robot teams that achieves plan recognition and adaptation concurrently through a single set of algorithms. We believe Pike will be widely applicable to a number of domains.

Pike's task representation has proven useful for simultaneous recognition and adaptation. The TPNU can encode many different types of contingencies: recovery ac-

[1] A video showcasing this integration can be seen at http://people.csail.mit.edu/sjlevine/ICAPS14/Kitchen.mov

tions (ex., if mug is dropped, pick it up), choice of agent, faster/slower action alternatives to achieve temporal constraints, and more. Pike's only requirement is that failures must be addressable by future contingencies. However, the TPNU does have one major limitation: flexibility outside of these contingencies. Pike will be unable to adapt should the robot or human stray beyond the TPNU's available contingencies. Should this happen, Pike would immediately detect failure. In the future, we plan to experiment with integrating Pike with a generative planner for added robustness. Additionally, we plan to develop probabilistic and continuous state-space variants of Pike.

Acknowledgments

We thank Pedro Santana, Peng Yu, David Wang, Andreas Hofmann, Enrique Fernandez, Ameya Shroff, Frank Yaul, and Scott Smith for many insightful discussions. We are indebted to Pedro Santana for implementing the computer vision system, as well as to Patrick Conrad and David Wang for their work implementing Drake. This work has been graciously funded under Boeing grant MIT-BA-GTA-1.

References

Alili, S.; Warnier, M.; Ali, M.; and Alami, R. 2009. Planning and plan-execution for human-robot cooperative task achievement. In *19th International Conference on Automated Planning and Scheduling*.

Ambros-Ingerson, J. A., and Steel, S. 1988. Integrating planning, execution and monitoring. In *AAAI*, volume 88, 21–26.

Avrahami-Zilberbrand, D.; Kaminka, G.; and Zarosim, H. 2005. Fast and complete symbolic plan recognition: Allowing for duration, interleaved execution, and lossy observations. In *Proc. of the AAAI Workshop on Modeling Others from Observations, MOO*.

Ayan, N. F.; Kuter, U.; Yaman, F.; and Goldman, R. P. 2007. Hotride: Hierarchical ordered task replanning in dynamic environments. In *Proceedings of the 3rd Workshop on Planning and Plan Execution for Real-World Systems (held in conjunction with ICAPS 2007)*, volume 2.

Bui, H. H. 2003. A general model for online probabilistic plan recognition. In *IJCAI*, volume 3, 1309–1315. Citeseer.

Carberry, S. 2001. Techniques for plan recognition. *User Modeling and User-Adapted Interaction* 11(1-2):31–48.

Chien, S. A.; Knight, R.; Stechert, A.; Sherwood, R.; and Rabideau, G. 2000. Using iterative repair to improve the responsiveness of planning and scheduling. In *AIPS*, 300–307.

Clodic, A.; Cao, H.; Alili, S.; Montreuil, V.; Alami, R.; and Chatila, R. 2009. Shary: a supervision system adapted to human-robot interaction. In *Experimental Robotics*, 229–238. Springer.

Conrad, P. R.; Shah, J. A.; and Williams, B. C. 2009. Flexible execution of plans with choice.

Conrad, P. R. 2010. Flexible execution of plans with choice and uncertainty. Master's thesis, Massachusetts Institute of Technology.

De Kleer, J. 1986. An assumption-based tms. *Artificial intelligence* 28(2):127–162.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial intelligence* 49(1):61–95.

Effinger, R. T.; Williams, B. C.; Kelly, G.; and Sheehy, M. 2009. Dynamic controllability of temporally-flexible reactive programs. In *ICAPS*.

Finzi, A.; Ingrand, F.; and Muscettola, N. 2004. Model-based executive control through reactive planning for autonomous rovers. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 1, 879–884. IEEE.

Forbus, K. D. 1993. *Building Problems Solvers*, volume 1. MIT press.

Fox, M., and Long, D. 2003. Pddl2. 1: An extension to pddl for expressing temporal planning domains. *J. Artif. Intell. Res.(JAIR)* 20:61–124.

Goldman, R. P.; Geib, C. W.; and Miller, C. A. 1999. A new model of plan recognition. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, 245–254. Morgan Kaufmann Publishers Inc.

Haigh, K. Z., and Veloso, M. M. 1998. Interleaving planning and robot execution for asynchronous user requests. In *Autonomous agents*, 79–95. Springer.

Kautz, H. A., and Allen, J. F. 1986. Generalized plan recognition. In *AAAI*, volume 86, 32–37.

Kim, P.; Williams, B. C.; and Abramson, M. 2001. Executing reactive, model-based programs through graph-based temporal planning. In *IJCAI*, 487–493.

Lemai, S., and Ingrand, F. 2004. Interleaving temporal planning and execution in robotics domains. In *AAAI*, volume 4, 617–622.

Levine, S. J. 2012. Monitoring the execution of temporal plans for robotic systems. Master's thesis, Massachusetts Institute of Technology.

McAllester, D., and Rosenblatt, D. 1991. Systematic nonlinear planning.

Muise, C.; Beck, J. C.; and McIlraith, S. A. 2013. Flexible execution of partial order plans with temporal constraints. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, 2328–2335. AAAI Press.

Shah, J. A.; Conrad, P. R.; and Williams, B. C. 2009. Fast distributed multi-agent plan execution with dynamic task assignment and scheduling. In *ICAPS*.

Tsamardinos, I.; Muscettola, N.; and Morris, P. 1998. Fast transformation of temporal plans for efficient execution.

Veloso, M. M.; Pollack, M. E.; and Cox, M. T. 1998. Rationale-based monitoring for planning in dynamic environments. In *AIPS*, 171–180.

Vidal, T. 1999. Handling contingency in temporal constraint networks: from consistency to controllabilities. *Journal of Experimental & Theoretical Artificial Intelligence* 11(1):23–45.