Planning with Temporal Uncertainty, Resources and Non-Linear Control Parameters

Mikael Nilsson, Jonas Kvarnström, Patrick Doherty
Department of Computer and Information Science
Linköping University, SE-58183 Linköping, Sweden
{mikael.a.nilsson,jonas.kvarnstrom,patrick.doherty}@liu.se

Abstract

We consider a general and industrially motivated class of planning problems involving a combination of requirements that can be essential to autonomous robotic systems planning to act in the real world: Support for temporal uncertainty where nature determines the eventual duration of an action, resource consumption with a non-linear relationship to durations, and the need to select appropriate values for control parameters that affect time requirements and resource usage.

To this end, an existing planner is extended with support for Simple Temporal Networks with Uncertainty, Timed Initial Literals, and temporal coverage goals. Control parameters are lifted from the main combinatorial planning problem into a constraint satisfaction problem that connects them to resource usage. Constraint processing is then integrated and interleaved with verification of temporal feasibility, using projections for partial temporal awareness in the constraint solver.

1 Introduction

In recent years there has been great interest in extending and adapting automated planning to better deal with requirements arising when autonomous robotic systems must plan to act in the real world. While each extension allows a planner to cover interesting new problem classes, there is often significant *interference* between different potential extensions: Planning algorithms are not "additive" in the sense that once a feature has been explored in one setting, it can easily be introduced in another. This often leads to new features being explored in isolation, frequently also requiring reduced expressivity in the underlying planning framework.

In this paper we consider a wide class of scenarios requiring multiple distinct features on top of the "baseline" for temporal concurrent planning, all of which are directly motivated by the requirements of an airplane surveillance scenario (Section 2) resulting from collaboration with industry. Efficiently supporting the *combination* of these features requires taking their *interactions* into careful consideration.

First, there are areas to be surveilled during specific intervals of time, which requires support for *non-classical temporal coverage goals*. Also, actions such as in-air refueling can only be executed within certain *time windows*.

Second, actions such as flying have "ordinary parameters" affecting discrete state transitions and *control parameters* that (in this paper) are defined by the fact that they exclusively affect time and resource usage. Selecting a good control parameterization is essential for time and resource constraints to be satisfied, but what is "good" depends on complex interdependencies throughout the entire plan: In one case, preserving fuel may be the best choice. In another, an airplane may need to *spend* additional fuel to be able to take over surveillance earlier, leaving another airplane with more fuel than strictly necessary, because it could then reach a refueling location within an earlier time window and therefore take over another surveillance task at an earlier time. Such trees of consequences can be complex to completely analyze in a heuristic function. Therefore much can be gained by *delaying commitment* for the control parameters, lifting their constraints into an independent problem. (Note that these parameters, when chosen, will not vary during action execution. This differs from how control parameters are used in other areas, e.g. control theory and dynamics.)

Third, the relation between time and resource requirements is highly *non-linear*, whereas existing research into separating control parameters has mainly focused on linear dependencies (see related work below).

Fourth, action durations are *uncertain* and cannot be perfectly predicted even if control parameters are fixed, and overestimating durations would be unsound, as it could lead to surveillance ending earlier than predicted. The temporal aspects of such problems can be modeled as STNUs, Simple Temporal Networks with Uncertainty (Vidal and Ghallab 1996). If an STNU is *dynamically controllable* (DC), the corresponding plan is guaranteed to be executable given that certain information about execution progress is dynamically provided to the execution algorithm. However, current efficient STNU algorithms require predetermined lower and upper bounds on durations, whereas in our case, bounds *depend* on control parameters. As an example setting a control parameter to *FAST* may yield STNU time bounds of $[60, 75]$ seconds and cause a resource consumption of $[2, 3]$ kg whereas the value *SLOW* would result in longer durations in the range of $[80, 100]$ seconds but instead consumes less resources, $[1.5, 2.2]$ kg.

This examplifies a class of planning problems where control parameter selection strongly interacts with uncertain

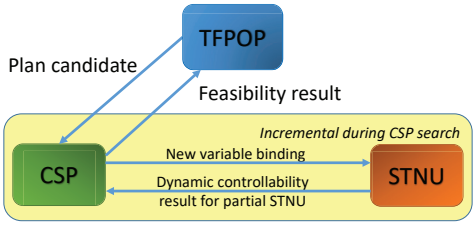

Figure 1: Final System Configuration

temporal bounds and non-linear resource consumption, and where existing approaches to individual aspects cannot trivially be combined while retaining efficiency. For example, setting control parameters relative to resource constraints and testing whether the result is DC leads to an inefficient brute force generate-and-test method.

An integrated treatment is therefore required. At the same time, forcing all aspects of the problem into a single solver is not the most efficient solution. We can benefit greatly from carefully *integrating* the use of existing efficient algorithms for verifying dynamic controllability.

After discussing the motivating example and related work, we present such an integrated system. The presentation follows a step by step approach where section 4 describes our starting point: an existing planner generating partially ordered plans (TFPOP, (Kvarnström 2011)). After gaining background knowledge of STNUs in section 5 we proceed stepwise in sections 6 to 9 to integrate and extend TFPOP with in turn: a DC-checking algorithm, temporal constraint derivation, temporal coverage goals and timed initial literals. Section 10 concludes the extensions by integrating a general constraint solver for efficient handling of control parameters and non-linear resource usage. Figure 1 shows the final configuration of system.

We then identify an opportunity to provide additional feedback between the two algorithms: Though modeling an entire STNU in a general constraint solver would be inefficient, specific *projections* from the STNU can be efficiently modeled as constraints that are necessary but not sufficient for dynamic controllability. This allows the constraint solver to very efficiently take into account many interactions between time and resources and to avoid a large majority of the control parameter assignments that would have violated DC, while still using a highly efficient DC verification algorithm to detect violations that satisfy the projections.

Finally, we empirically test and analyze the performance of the resulting integrated planner.

2 Motivating Example

The techniques presented in this paper are applicable to a wide variety of planning problems involving uncertain durations and non-linear dependencies between alternative time and resource requirements. To demonstrate these techniques, we will now describe an industrially motivated example where a set of manned or unmanned *airplanes* surveil a set of *bridges* to detect cars matching given descriptions.

As an airplane cannot hover, it must instead perform a sequence of *sweeps*, each of which consists of crossing its assigned bridge, turning around, crossing in the other direction, and turning around again. To guarantee that all cars passing the bridge will be seen by on-board sensors, each sweep must be finished within the bridge's maximum *sweep duration*, which depends on the length of the bridge and the maximum speed of a car. The range of *possible* sweep durations for a particular airplane depends on its flight envelope (including minimum/maximum airspeed and turn acceleration) and the length of the bridge.

The objective is to surveil each bridge throughout its own *surveillance interval*, which requires both arriving on time and sufficient *endurance*: "Finishing early" is unacceptable. When an airplane flies from its initial *base* location to a bridge, it may (depending on the required airspeed) consume a significant proportion of its *fuel*. The airplane can then perform a number of sweeps, after which it may have to hand over surveillance to another in order to return to base or go to a designated *in-air refueling* location. Due to multiple ongoing missions, in-air refueling is only allowed during specific *time slots*, 20 minutes each hour.

Estimating Time and Resource Usage. Each flight action has a single control parameter: The *target airspeed*. For any given value we can estimate time and fuel requirements using a motion planner that generates 3D trajectories relative to known no-fly-zones using a simulated airplane model, taking into account limits on speed, acceleration, turn acceleration, and rate of rise and descent. The effect of altitude on fuel consumption is also considered. As this cannot be described using analytic expressions, the control parameter is discretized and the motion planner is called for each value. The empirical evaluation tests different discretizations.

Some uncertainty arises from imperfectly capturing the dynamics and fuel consumption characteristics of an airplane. More importantly, unpredictable wind can significantly affect the distance an airplane must travel relative to the air, especially at lower speeds. With a fixed *airspeed*, the duration of a flight can vary. If one attempts to achieve a fixed action duration, the required airspeed will instead vary significantly and more fuel must be reserved.

The Importance of Full DC Verification. When crossing a bridge, lower temporal bounds must be considered as they determine the amount of guaranteed surveillance coverage. Upper bounds must also be considered as they must be below the maximum permitted sweep duration. Both bounds also affect handover scheduling and the ability to reach refueling locations within a window. For example, if an aircraft B that will take over surveillance from A arrives early, it must loiter more, and if it arrives late (relative to the earliest time A may finish its sweeps), coverage may be interrupted. DC checking verifies all bounds within the entire plan. This can in turn rule out control parameter choices, affect the number of possible sweeps for a particular aircraft, etc.

3 Related Work

There is a wide variety of planners that in certain ways overlap with the functionality required by our motivating exam-

ple, but still lack support for critical aspects.

The focus of Kongming (Li and Williams 2008) and Scotty (Fernández-González, Karpas, and Williams 2017) is on continuous control parameters and continuous dynamics modeled using Flow Tubes. These planners assume controllable action durations and are therefore not suited for handling the temporal uncertainty aspect that we address.

Partial-order heuristic planners supporting linear programs include COLIN (Coles et al. 2009), POPF (Coles et al. 2010), OPTIC (Benton, Coles, and Coles 2012) (focusing on optimization) and POPCORN (Savas et al. 2016). Of these, POPCORN is most closely related to our work, extending POPF to allow control parameters that are separated into an LP and can affect resources. However, it assumes that the execution mechanism can choose both times and resource usages of actions within given bounds. Thus it cannot handle temporal or resource uncertainty. Being LP-based, it also cannot handle the type of non-linear effects that we require.

Among Hierarchical Task Network (HTN) planners, FAPE (Dvorák et al. 2014) supports temporal uncertainty but does not allow lifting control parameters that affect temporal bounds. CHIMP (Stock et al. 2015) and GSCCB-SHOP2 (Qi et al. 2017) allow resources but not temporal uncertainty. CTPU-HTN (Liu et al. 2016) supports CSTNUs, where temporal bounds can differ depending on observations during execution as opposed to being chosen by a planner. Resources and control parameters are not supported. A recent unnamed HTN planner (Zhao et al. 2017) handles temporal uncertainty but not control parameters.

Timeline-based planners, and specifically flexible timeline-based planners (Mayer, Orlandini, and Umbrico 2016), are related to our work since they also use intervals to bound event times. However, they either support resources but not uncertain durations (Umbrico, Orlandini, and Mayer 2015) or the opposite (Cimatti et al. 2018; Umbrico et al. 2017). A further difference when comparing with our planner is that current flexible timeline based planners do not support the type of control parameters that we deal with: Control parameters that affect both duration bounds and resource consumption/production, both being uncertain.

The InCell library (Pralet et al. 2014) is more focused on scheduling than planning. It can be used for planning with resources and control parameters but lacks support for uncertain action durations.

Recently a new temporal formalism was proposed. The CCTPU formalism (Cui and Haslum 2017) can model STNUs with duration / edge alternatives, where subnetworks are chosen dynamically during execution. But this high expressivity comes at a price: Its verification algorithms are shown by benchmarks to scale like generate-and-test methods. Therefore, combining this formalism with the other aspects that we target in this paper would be infeasible.

4 Background: TFPOP

Our planner is based on TFPOP, a temporal partial-order planner where each action is associated with an *agent* and each agent has one or more sequential *threads* of execution, with partial ordering *across* threads (Figure 2). This allows

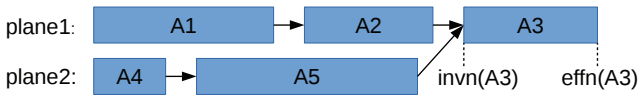

Figure 2: Plan Structure with Precedence Constraints

strong state information to be generated, allowing the use of precondition control formulas (Bacchus and Ady 1999) for guidance. We briefly present a simplified version of TFPOP, removing features that will be replaced by new functionality. See (Kvarnström 2011) for details and (Weld 1994) for an overview of *standard* partial order (POCL) planning.

Planning Problems. TFPOP uses a typed finite-domain *fluent* (state variable) representation, where $\mathsf{loc}(\mathit{object})$ may be a location-valued fluent taking an object as a parameter.

The first parameter of an *operator* is always the executing agent: $\mathsf{fly}(\mathit{plane}, \mathit{from}, \mathit{to})$. For agents with multiple threads (independent flying and camera control), the second parameter identifies the thread. The *precondition* $\mathit{pre}(o)$ may be disjunctive and quantified and can use $\mathsf{goal}(\phi)$ to test whether a formula ϕ is entailed by the goal. *Effects* are conjunctive and unconditional. An *action* is a fully grounded operator.

The *initial state* is assumed to be completely defined, while the *goal formula* can be disjunctive.

Expressivity. Some features from PDDL (Edelkamp and Hoffmann 2004) are not supported in TFPOP, such as start effects. This follows from the fact that (a) we wish to explore *new combinations* of features such as temporal uncertainty and control parameters, (b) certain other features were less essential for this case, and (c) unconditionally retaining *all* features of earlier planners is prohibitively complex.

Plan Structures. A *plan* is a tuple $\pi = \langle A, N, L, O \rangle$:

- A is the set of all *actions* occurring in the plan. We define $\mathit{act}(\pi, t)$ as the subset of A executed by thread t.
- N contains, for each $a \in A$, an *invocation node* $\mathit{inv}(a)$ where preconditions must hold, and an *effect node* $\mathit{eff}(a)$ where effects are guaranteed to have taken place.
- L is a set of ground *causal links* $a_i \xrightarrow{\mathsf{f}=v} a_j$ stating that $a_i \in A$ will achieve the condition $\mathsf{f} = v$ for $a_j \in A$.
- O is a set of explicit *ordering constraints* (a_i, a_j) denoting that a_j begins after or at the same time a_i ends. For each thread t, $\mathit{act}(\pi, t)$ must be totally ordered by O.

The transitive closure of O is a strict (irreflexive) partial order denoted by $\prec_O$, or by $\prec$ when the index is obvious from context. This ordering carries over directly to nodes: If $a \in A$ then $\mathit{inv}(a) \prec \mathit{eff}(a)$, and if $a \prec a'$ then $\mathit{eff}(a) \prec \mathit{inv}(a')$.

Executability. Actions can start and end in any order corresponding to a *node sequence* $[n_0, n_1, \ldots, n_{|N|-1}]$ consistent with $\prec$. Executability requires that in all such sequences, no two overlapping actions (where both are invoked but not terminated) affect the same fluent and no action affects a fluent required by an overlapping action. Also, preconditions must be satisfied: Let s_0 be the initial state and for all $0 < i \leq k$:

$$s_i = \begin{cases} s_{i-1} \text{ updated by } n_i & \text{if } n_i \text{ is an effect node,} \\ s_{i-1} & \text{otherwise.} \end{cases}$$

Then for every invocation node n_j corresponding to an action a, we must have $s_j, goal \models pre(a)$.

Note. If the precondition of an invocation node n_j depends on the result of an effect node n_i, the TFPOP algorithm ensures that $(n_i, n_j) \in O$, so that $n_i \prec n_j$. Similarly, any potential *interference* between effects and preconditions will be prevented through constraints in O.

Solutions. An executable plan is a *solution* iff every associated node sequence results in a state s satisfying the goal.

Algorithm. In the following, **choose** refers to standard nondeterministic choice, often implemented by backtracking.

procedure TFPOP
$\langle A, N, L, O\rangle \leftarrow \langle\{a_0\}, \{inv(a_0), eff(a_0)\}, \varnothing, \varnothing\rangle$ *// Initial plan*
repeat
  if goal satisfied **return** $\langle A, N, L, O\rangle$
  choose a thread t to which an action should be added
  choose an action a for t s.t. $pre(a)$ is not *false* at the end of t
  choose, from alternatives provided by *make-true*,
    causal links L' and precedence constraints O' ensuring $pre(a)$ is satisfied, a does not interfere with existing actions and no existing action interferes with a
  add a to A, $\{inv(a), eff(a)\}$ to N, L' to L, and O' to O
  update existing partial states; create new partial state after a

Initialization. The *initial plan / search node* is an executable "empty" plan $\langle A, N, L, O\rangle$ containing only the special initial action a_0, which is associated with all threads. As in standard POCL planning, a_0 has no preconditions but its effects define the initial state. As TFPOP does not use means-ends analysis, no action represents the goal.

Goal Check. If the goal is satisfied, a solution is returned.

Successor. A successor adds one new action to the end of an existing thread t. Threads can be selected using multiple strategies, for example by balancing actions across agents.

Then we must find an executable action a to add at the end of t. Knowing the exact state of the world when a will be invoked is in general impossible due to partial ordering. Instead each action in a thread is associated with a *partial state* mapping each fluent f to a *set* $\{v_1, \ldots, v_{|N|}\}$ containing all values that f *may* take on between that action and the next, or until infinity for the last action in the thread. The state of the last action (possibly a_0) is used to determine whether $pre(a)$ is *definitely false*, in which case a can efficiently be discarded. This is essential for effective precondition control.

If $pre(a)$ is definitely *true*, causal links and precedence constraints must be added. If *unknown*, the reason may be *potential* interference, avoidable by promoting or demoting actions in the partial order. In both cases TFPOP uses *make-true*() (Kvarnström 2011) to find all possible extensions to L and O that result in new executable plans without interference (POCL threats) across actions.

State Inference. A new partial state is created by *strengthening* the previous one using the effects of a and *weakening* it according to all effects from actions that are in other threads and may finish after a. The states of other threads must also be weakened wherever the effects of a may occur.

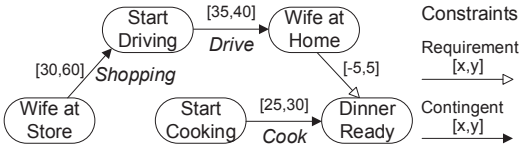

Figure 3: STNU for the Cooking Problem

5 Background: STNs with Uncertainty

Durations d_i of actions A_i can rarely be perfectly predicted, but one can often find context-dependent upper and lower bounds $[l_i, u_i]$. Temporal planners have traditionally (often implicitly) focused on the upper bound. While this can suffice for handling classical *deadlines*, we consider scenarios where events can both occur *too early* and *too late*. STNUs (Vidal and Ghallab 1996) allow us to succinctly model the temporal aspects of such plans.

Definition 1. *A **simple temporal network with uncertainty (STNU)** is a graph $S = \langle N = N_R \cup N_C, E = E_R \cup E_C\rangle$.*

The nodes $N = \{n_1, \ldots, n_{|N|}\}$ are divided into disjoint sets of controlled events N_R *and* contingent events N_C. *Each $n_i \in N$ is associated with a temporal variable t_i.*

An edge $n_i \xrightarrow{[l,u]} n_j$ is either a requirement edge *in E_R, representing the* requirement constraint $l \leq t_j - t_i \leq u$, *or a* contingent edge *in E_C, representing a* contingent constraint $l \leq t_j - t_i \leq u$ *where the outcome will be determined by nature. For contingent edges, $n_j \in E_C$.* □

In Figure 3, a man wants to surprise his wife with a dinner when she returns after a shopping trip. The dinner must not be ready too early, or it will grow cold. It must also not be ready too late, or the wife will have to wait. This is captured by the requirement edge labeled [-5,5], while the *uncontrollable* (but bounded) durations of shopping, driving home and cooking are captured by contingent edges.

Can the man ensure that the requirement is respected regardless of the actual duration outcome for each action (contingent edge), given that he can observe when events in N_C occur (when driving begins) so that he can adapt his execution schedule for nodes in N_R (decide when to start cooking)? Yes, if cooking starts exactly 10 time units after receiving information that the wife starts driving home. This problem is captured by the property of *dynamic controllability* (*DC*, see (Morris, Muscettola, and Vidal 2001) for details):

Definition 2. *A **dynamic execution strategy** is a strategy for assigning timepoints to controllable events during execution, given that at each timepoint, it is known which contingent events have already occurred. The strategy must ensure that all requirement constraints will be respected regardless of the outcomes for the contingent events.*

*An STNU is **dynamically controllable** if there exists a dynamic execution strategy for executing it.* □

6 TFPOP with Uncontrollable Durations

We now add initial support for actions with uncontrollable durations, using E2IDC (Nilsson, Kvarnström, and Doherty 2016) to incrementally verify dynamic controllability.

Planning Problems. For each action a, the planning problem must now specify duration bounds $0 < \mathit{mindur}(a) \leq \mathit{maxdur}(a)$.

Plan Structures. A *plan* is now a tuple $\pi = \langle A, S, L, O\rangle$, where the STNU $S = \langle N = N_R \cup N_C, E = E_R \cup E_C\rangle$ contains the nodes N. Each action $a \in A$ has a controlled *invocation node* $\mathit{inv}(a) \in N_R$ and a contingent *effect node* $\mathit{eff}(a) \in N_C$, except a_0, both of whose nodes are controlled. All explicit ordering constraints $(a_i, a_j) \in O$ will be added as requirement edges $\mathit{eff}(a_i) \xrightarrow{[0,\infty)} \mathit{inv}(a_j)$. Contingent edges represent actions.

Executability. Executability now requires S to be dynamically controllable. Plans will be executed by an *STNU dispatcher*. Previous executability conditions apply to all node sequences $[n_0, n_1, \ldots, n_{|N|-1}]$ compatible with $\prec$ and S.

Note. Using STNUs, it will be technically possible to constrain two TFPOP nodes n_i and n_j to occur at exactly the same time. For invocation nodes, this is unproblematic. If n_j is an invocation node and n_i an effect node, either their order is irrelevant for TFPOP or the node sequence order is also uniquely constrained by $(n_j, n_i) \in O$ or $(n_i, n_j) \in O$ to guarantee precondition support or non-interference. If both are effect nodes, they cannot affect the same fluents and (due to non-interference) the correctness of the plan cannot depend on the order in which the effects are applied.

Algorithm. The TFPOP algorithm is extended as follows.

procedure TFPOP-STNU
$S_0 \leftarrow \langle\{\mathit{inv}(a_0), \mathit{eff}(a_0)\}, \{\mathit{inv}(a_0) \xrightarrow{[0,0]} \mathit{eff}(a_0)\}\rangle$
$\langle A, S, L, O\rangle \leftarrow \langle\{a_0\}, S_0, \varnothing, \varnothing\rangle$
checker $\leftarrow$ new instance of E2IDC initialized for S_0
repeat
  if goal satisfied **return** $\langle A, S, L, O\rangle$
  choose a thread t to which an action should be added
  choose an action a for t s.t. $\mathit{pre}(a)$ is not *false* at the end of t
  choose L' and O' as in original TFPOP
  add a to A, $\mathit{inv}(a)$ to N_R, $\mathit{eff}(a)$ to N_C, L' to L, and O' to O
  // update STNU edges $E = E_R \cup E_C$
  add $\mathit{inv}(a) \xrightarrow{[\mathit{mindur}(a), \mathit{maxdur}(a)]} \mathit{eff}(a)$ to E_C
  for every (a_i, a_j) in O', add $\mathit{eff}(a_i) \xrightarrow{[0,\infty]} \mathit{inv}(a_j)$ to E_R
  for every edge e that was added to E_R, E_C:
    if not checker.update(e) **then fail** (backtrack)
  update existing partial states; create new partial state after a

Initialization. The *initial node* now contains an initial STNU where $\mathit{inv}(a_0)$ and $\mathit{eff}(a_0)$ are controlled (in N_R), with a requirement constraint specifying that these synthetic events are simultaneous. An instance of the incremental DC checker is created and updated with all existing constraints.

STNU Update. When an action is linked into the partial order, the STNU is updated with new nodes as well as edges: A contingent edge representing the action duration and a requirement constraint for each new precedence generated by *make-true*. E2IDC is called once for each new edge to incrementally determine whether the plan remains DC. (Backtracking is synchronized for E2IDC and TFPOP.)

7 Deriving Temporal Constraints

Any STNU generated above is trivially DC: It directly represents a partial order, where any action can be delayed until its predecessors have finished. The full power of STNUs is required when events can occur either too early or too late. Such situations arise from temporal constraints across multiple actions, constraints that are hardcoded in typical STNU examples but that we must instead derive incrementally during planning. Derivation rules must be able to refer to specific *nodes* and *node precedence* ($\prec$), and are not in themselves constraints (such as PDDL plan constraints) – they cause the *derivation* of STNU constraints. We therefore define a new structure for derivation rules. Additional intuitions are given by the examples below.

Definition 3. *A* temporal derivation rule $\langle V, \phi, D\rangle$ *for a plan* $\pi = \langle A, S, L, O\rangle$ *is defined as follows.* V *is a set of variables* $\{a_1, \ldots, a_n\}$ *ranging over actions in* A *and shared between* ϕ *and* D. ϕ *is a first-order formula with equality allowing free variables from* V*, supporting quantification over additional action variables* a_i *and node variables* n_i. *Supported atomic formulas and terms in* ϕ *include:*

- $\mathit{assigns}(a, f, v)$ *to test if action* a *has the effect* $f := v$*, and* $\mathit{holds}(n, f, v)$ *to test if* $f = v$ *in the partial state associated with node* n
- $a_i \prec a_j, n_i \prec n_j$ *to test precedence*
- $\mathit{inv}(a)$, $\mathit{eff}(a)$ *– the invocation / effect node of a*
- $\mathit{act}(n)$ *– the action of node n*

Finally, D *is a set of requirement edges to be derived for every instantiation of the variables in* V *that satisfies* ϕ. □

When an action has been added, TFPOP applies all applicable instances of temporal derivation rules before verifying dynamic controllability (before "for every edge").

Example 1. In a planning version of the "cooking STNU", the rule $\langle\{a_1, a_2\}, \mathit{assigns}(a_1, \mathsf{dinner\text{-}ready}, \mathit{true}) \wedge \mathit{assigns}(a_2, \mathsf{wife\text{-}home}, \mathit{true}), \{\mathit{eff}(a_1) \xrightarrow{[-5,5]} \mathit{eff}(a_2)\}\rangle$ expresses the desire for dinner to be finished at arrival ± 5 minutes. □

Example 2. In the TFPOP formalization of the airplane surveillance problem, an agent can explicitly assign itself as the *surveillor* of a particular bridge. Domain-specific control ensures a surveillor can only fly to the *start* of its bridge (a location and a specific heading) and perform sequences of flights that result in *sweeps*. Before leaving, an airplane must unassign itself from the bridge, so that another can take over. A non-surveillor can fly to a bridge in preparation for taking over, fly to a refueling location, or fly back to its base.

Each sweep is then an action sequence of unknown length whose maximum duration must be constrained to (e.g.) 280 seconds. For each bridge b, we derive such a duration constraint whenever there is an action a_1 that begins at the start of the bridge, a later action a_2 that returns to the start, and (to ensure these nodes cover only a single sweep) no *intermediate* node where the surveillor is at the start of b:
$\langle\{a_1, a_2\}, a_1 \prec a_2 \wedge \mathit{holds}(\mathit{inv}(a_1), \mathsf{loc}(\mathsf{surveillor}(b)), \mathsf{start}(b)) \wedge$
$\mathit{assigns}(a_2, \mathsf{loc}(\mathsf{surveillor}(b)), \mathsf{start}(b)) \wedge$
$\forall n.\mathit{inv}(a_1) \prec n \prec \mathit{eff}(a_2) \rightarrow \neg\mathit{holds}(n, \mathsf{loc}(\mathsf{surveillor}(b)), \mathsf{start}(b)),$
$\{\mathit{inv}(a_1) \xrightarrow{[0,280]} \mathit{eff}(a_2)\}\rangle$ □

Performance. TFPOP already indexes nodes where specific facts hold or are assigned, allowing it to quickly find possible values for a_1 and a_2 above. Common patterns such as finding nodes n where $n_1 \prec n \prec n_2$ are also optimized, allowing efficient use of quantification.

8 Temporal Coverage Goals

A bridge can be considered *surveilled* throughout the execution of each sweep. We now need a way of expressing the goal that a bridge is surveilled *without interruption* throughout longer periods, a form of *temporal coverage goal*, and a way of determining when such goals are satisfied.

Each coverage goal is expressed on the form $\langle t_1, t_2, f\rangle$, denoting the need to ensure the special fluent f remains true throughout $[t_1, t_2]$. For example, a three-hour surveillance goal can be expressed as $\langle 1800, 12600, \mathsf{surveilled}(b)\rangle$. Goals referring to the same fluent (surveilling the same bridge) must have disjoint temporal intervals.

Full coverage can require multiple actions performed by different collaborating agents, where no single agent has the required endurance. Setting f to *true* at the start of each satisfying action and *false* at the end would not suffice: f must hold without interruption, and upholding actions can overlap. We therefore introduce the notion that an action can *uphold* a set of (boolean) fluents *upheld*(a), which are then *true* during exactly those intervals where they are upheld by at least one action. This resembles the use of *durational fluents* in Temporal Action Logic (Doherty and Kvarnström 2008). modify the initialization of TFPOP-STNU as follows, resulting in the new algorithm TFPOP-STNU-CSP:

- Each coverage goal can be *unstarted* (no action upholds it), *started* (partially covered), or *finished* (completely covered). Initially, mark all goals as *unstarted*.
- Then, for each coverage goal $g_i = \langle t_1, t_2, f\rangle$, add two controlled events *beg*(g_i) and *end*(g_i) to N_R, and add *eff*$(a_0) \xrightarrow{[t_1,t_1]}$ *beg*(g_i) and *eff*$(a_0) \xrightarrow{[t_2,t_2]}$ *end*(g_i) to E_R. These are processed by the initialization of E2IDC.

The following is inserted before "**for** every edge":

for each $f \in$ *upheld*(a): *// May be empty!*
  if at least one coverage goal for f is not *finished*:
    $g = \langle t_1, t_2, f\rangle \leftarrow$ the earliest of these goals (minimal t_1)
    if g is marked *started*:
      // Ensure no gaps in coverage – overlap is permitted
      Let a' be the latest added action upholding f
      add *inv*$(a) \xrightarrow{[0,\infty]}$ *eff*(a') to E_R
    else:
      // First action covering interval must start before or at t_1
      add *inv*$(a) \xrightarrow{[0,\infty]}$ *beg*(g) to E_R; mark g *started*
    choose between: *// Described below*
      – add *end*$(g) \xrightarrow{[0,\infty]}$ *eff*(a) to E_R; mark g *finished*
      – pass (do nothing, remain in *started* mode)

Each interval goal g is first *unstarted*. The first action covering it will be constrained to start on time, and the goal will be marked *started*. Subsequent actions are constrained to ensure there are no gaps. Finally, any action (including the first) could potentially be the last to cover the interval. The concrete implementation prioritizes testing this and marking g finished. TFPOP then continues with DC testing.

If the existing actions cannot cover the entire interval of g while preserving DC, this leads to immediate backtracking to the second alternative: Retain partial coverage.

Finally, the goal satisfaction check must be extended to test whether all coverage goals are marked *finished*.

9 Time Slots and Timed Initial literals

In the motivating example, refueling can only be performed during specific time slots. Such slots can be supported through Timed Initial Literals (Edelkamp and Hoffmann 2004), which are implemented in TFPOP as follows.

First, let a planning problem specify a set of timed assignments of fluent values $\langle t, f, v\rangle$ stating that at time t, the fluent f will be assigned the value v – for example, $\langle 3600, \mathsf{can\text{-}refuel}, \mathit{true}\rangle$ and $\langle 4500, \mathsf{can\text{-}refuel}, \mathit{false}\rangle$.

Second, for each fluent f, collect all such assignments in a sequence $\langle t_i, f_i, v_i\rangle_{i=1}^{n}$ sorted by increasing t_i. For each assignment, create a synthetic action a_i having no preconditions, a controllable duration of 0, a single effect $f := v$, and a synthetic execution thread T_f used for timed changes to the fluent f. Add a to A, *inv*(a) and *eff*(a) to N_R, (a_{i-1}, a_j) to O (for the first effect a_1 this relates to the initial action a_0), and *eff*$(a_0) \xrightarrow{[t_i,t_i]}$ *inv*(a_i) and *inv*$(a_i) \xrightarrow{[0,0]}$ *eff*(a_i) to E_R. For each added action, perform DC verification, update existing partial states, and create a new partial state for T_f.

When searching for applicable actions, TFPOP will never add new actions to synthetic threads but can use assignments by synthetic actions to support preconditions. Existing interference (threat) resolution mechanisms will ensure that an action requiring can-refuel is constrained to take place completely within an interval when can-refuel remains *true*.

This can trivially be extended to support *uncertain* timepoints using contingent edges *eff*$(a_0) \xrightarrow{[l,u]}$ *inv*(a_i).

10 Control Parameters, Resources and Time

We now turn to the full problem of introducing control parameters that affect both time and resource requirements. To keep this presentation concise we will use a single control parameter. Multiple parameters can be introduced similarly. We will also focus the current presentation on resources belonging to specific agents. Resource effects then become totally ordered since each agents actions are, and resource constraints are tested relative to this order. Earlier versions of TFPOP support a considerably looser ordering among resource-affecting actions, which can easily be modeled as CSP constraints.

We choose to use an expressive general *constraint solver* to support discrete choice of control parameters, arbitrary non-linear resource usage, general and action-specific resource constraints, certain temporal aspects (Solver 3 below), as well as certain planned future extensions.

Planning Problems. Assume each action a has a single control parameter *ctrl*(a) taking on values c of a specified finite (possibly numeric) type: *ctrl*$(\mathsf{fly}(...)) = \mathsf{speed} \in \{100, \ldots\}$. STNU duration bounds now depend on the value c through the new functions *mindur*(a, c) and *maxdur*(a, c).

Each problem specifies a set of consumable *reservoir type resources*. Each action has a possibly empty set *reseff*(a) of *resource effects* $\langle e, r, v\rangle$, where $e \in \{consume, produce, assign\}$ is the type of effect, r is the resource being affected, and v is a function from control parameter values c to upper and lower bounds $[l, u]$ on consumption, production, or the resulting assigned value. An action has at most one effect on any given resource. The initial state defines the possibly available amount $[avail^-(r), avail^+(r)]$ for each resource r as well as the minimum and maximum permitted amounts *minr*(r) and *maxr*(r). Similar to ordinary fluents, this turns into assignment effects in the initial action a_0.

To ensure that control parameters (together with time and resources) can be separated from the main (action) search space, they must not appear in preconditions or effects.

CSP for Resources. We will now define a Constraint Satisfaction Problem (CSP, (Tsang 2014)) relating control parameters to resource usage/constraints given a candidate plan $\pi = \langle A, S, L, O\rangle$. For simplicity, the presented formulation requires the sequence *resact*(π, r) of actions affecting a resource r to be totally ordered by $\prec_O$, which is true for many thread-local resources such as an agent's fuel. This assumption can be relaxed using techniques such as those employed by TFPOP in (Kvarnström 2011).

- For each action $a \in A$, create a constraint variable ctrl_a representing *ctrl*(a), its domain being the permitted values for this parameter. The value will be selected by the constraint solver or optimizer relative to other constraints.
- For each resource r and each $a \in$ *resact*(π, r), let $\langle e, r, v\rangle$ be the unique resource effect of a that affects r and:
  - Create the numeric constraint variables $\mathsf{avail}^-_{r,a}$ and $\mathsf{avail}^+_{r,a}$ representing the minimum and maximum amounts of r available at the end of a, respectively. Constrain both to be in $[minr(r), maxr(r)]$.
  - For each value c of *ctrl*(a) with resource effect bounds $v(c) = [l, u]$:
    – If $e =$ *assign* (which first occurs for a_0), add the constraint $\mathsf{ctrl}_a = c \rightarrow \mathsf{avail}^-_{r,a} = l \wedge \mathsf{avail}^+_{r,a} = u$.
    – If $e =$ *consume*, a previous action a' referenced the same resource. Add the constraint $\mathsf{ctrl}_a = c \rightarrow \mathsf{avail}^-_{r,a} = \mathsf{avail}^-_{r,a'} - u \wedge \mathsf{avail}^+_{r,a} = \mathsf{avail}^+_{r,a'} - l$
    – If $e =$ *produce*, add the constraint $\mathsf{ctrl}_a = c \rightarrow \mathsf{avail}^-_{r,a} = \mathsf{avail}^-_{r,a'} + l \wedge \mathsf{avail}^+_{r,a} = \mathsf{avail}^+_{r,a'} + u$

Expressive CSP solvers allow succinct and efficient *array lookup* representations of the constraints above (indexed by *ctrl*(a)), which our implementation uses for performance. Note that all constraints are bidirectional, allowing resources to constrain possible values for control parameters.

Inter-Action Resource Usage. In many domains the formalization above is sufficient to completely characterize resource usage and resource constraints. However, airplanes will continue consuming fuel as long as they are airborne, regardless of whether they are explicitly executing actions or *loitering* while waiting for (for example) the correct time to take over surveillance. Techniques for computing bounds on such *loiter durations* and the associated *loiter resource usage* have been implemented through analysis of the entire STNU structure.

Example 3. Agent X executes A and B, and Agent Y executes C. The uncertain duration between A and B must be derived from the STNU because it affects possible resource usage for X. Action B must occur after both A and C, so the time between A and B depends on when actions A and C end. This in turn depends on other plan aspects.

While loiter effects are applied in our empirical testing, the techniques themselves are outside the scope of this paper and will be published in another forthcoming paper. Integrating such *inter-action resource usage* in the constraints above is straight-forward, resulting in one more resource consumption term between each pair of actions in *resact*(π, r).

Action-Specific Resource Constraints. Actions can declare resource constraints: At the start of in-air refueling, at least 750 kg of fuel must be available ($\mathsf{avail}^-_{\mathrm{fuel},a'} \geq 750$, where a' is the previous action affecting fuel).

Solver 1: Generate-and-Test. The full *expressivity* of control parameters could be achieved with minimal integration: Each time a new action is added to π, generate and solve the above constraint problem. The control parameter values ctrl_a selected by the solution are "resource-feasible". To determine temporal feasibility, generate an STNU where for each action a, the uncontrollable duration is $[mindur(a, \mathsf{ctrl}_a), maxdur(a, \mathsf{ctrl}_a)]$. If this STNU is not DC, find an alternative solution to the CSP, create an STNU, and test again.

This is very inefficient: As there is no information about *why* a CSP solution is non-DC, we must blindly test many solutions that may only differ in non-essential parameters.

Solver 2: Integrating CSP and STNU. Many constraint solvers are based on a combination of *constraint propagation* and *search*. In every search node, each variable v is associated with a *domain* D_v, a set of possible values. Initially the domain consists of all values of the given type. In descendant nodes, domains can be reduced by constraint propagation or through assumptions made in the search process, but not expanded – information increases monotonically.

Therefore, an integrated CSP and STNU solver can incrementally add a contingent edge for an action as soon as its control parameter has been given a value. If this violates DC, the CSP search process can safely backtrack to try other alternatives: All descendant nodes would include the same constraints (and more), and would also be non-DC.

We first modify the end of TFPOP-STNU as follows.

procedure TFPOP-CSP-STNU (changes for the final part)
  $P_S \leftarrow \varnothing$ *// Singleton control parameter choices*
  …
  add a to A, *inv*(a) to N_R, *eff*(a) to N_C, L' to L, and O' to O
  // update STNU requirement edges in E_R
  for every (a_i, a_j) in O', add *eff*$(a_i) \xrightarrow{[0,\infty]}$ *inv*(a_j) to E_R
  for every edge e that was added to E_R:
    if not checker.update(e) **then fail** (backtrack)
  generate constraint satisfaction problem *csp*
  if interleave-csp-stnu(A,S,*csp*,P_S,checker) signals error **then**

fail
update existing partial states; create new partial state for t

The STNU generated by TFPOP now only contains the (control-independent) requirement edges. Contingent edges can only be added when control parameters are determined. The set P_S keeps track of all control parameter choices for contingent constraints that are added to the STNU. These contingent constraints are only added to the STNU when their corresponding control parameters have a single choice value left.

The basis for interleave-csp-stnu is a standard constraint solver as described above. In each search node, after constraint propagation, it calls the procedure below to update contingent edges in the STNU.

procedure incrementally-check-STNU(A,S,*csp*,P_S,checker)
$P_{\text{target}} = \{\mathsf{ctrl}_a = x \mid a \in A \wedge \mathit{csp}.D_{\mathsf{ctrl}_a} = \{x\}\}$
revert S to state corresponding to knowing only $P_{\text{target}} \cap P_S$
$P_S = P_{\text{target}} \cap P_S$
for each $\mathsf{ctrl}_a = x$ in $P_{\text{target}} - P_S$:
$P_S = P_S \cup \{\mathsf{ctrl}_a = x\}$
add $e = (\mathit{inv}(a) \xrightarrow{[\mathit{mindur}(a,\mathsf{ctrl}_a),\mathit{maxdur}(a,\mathsf{ctrl}_a)]} \mathit{eff}(a))$ to E_C
if not checker.update(e) **then fail**

Our objective is for the STNU to have one contingent edge for each control parameter currently assigned a definite (single) value by the CSP, represented by P_{target}.

The STNU currently *has* such edges corresponding to assignments in P_S (initialized to $\varnothing$ before the first call, and updated incrementally within each call). However, the CSP solver may have backtracked and removed assignments since the last call. Thus, edges in S not motivated by the current shared assignments $P_{\text{target}} \cap P_S$ must be *removed* by reverting S to the state it had when only these assignments existed. Then new edges corresponding to $P_{\text{target}} - P_S$ can be *added*. If this fails, the constraint solver will backtrack.

Solver 3: Temporal Projections in the CSP. Interleaving general constraint solving with DC verification provides *early feedback* as soon as a subset of control parameters are set in a way that violates DC. This is distinct from the stronger *bidirectional* relationship between control parameters and resources within the CSP: If an airplane has limited fuel, one can directly eliminate a range of values for the speed control parameter, which can then lead to additional chains of inferences through constraint propagation as opposed to search. Could we then gain additional performance by also introducing time into the CSP?

A potential approach would be to also encode the DC verification problem as a constraint problem, placing all constraints in a single solver. However, because such encodings require disjunctive constraints (Cui et al. 2015), interleaving a dedicated DC verification algorithm such as E2IDC is much faster than processing the complete STNU with a general solver (up to 60 times in our testing).

Instead we encode *some* STNU aspects that are necessary for dynamic controllability and can be efficiently handled by constraint solving, use these for improved constraint propagation (including detecting many of the control choices leading to non-DC STNUs), and retain the interleaved use of E2IDC to find cases that fall through the first "filter".

Recall that a contingent edge $n \xrightarrow{[l,u]} n'$ represents a need to handle all possible duration outcomes in $[l, u]$. A *projection* (Vidal and Ghallab 1996) replaces such edges with requirement edges $n \xrightarrow{[v,v]} n'$ where $v \in [l, u]$, representing the need to handle a particular outcome. The result is an STN (Dechter, Meiri, and Pearl 1991) whose constraint encoding is far less complex. If the STN is inconsistent, the original STNU was not DC.

The *AllMax* projection uses $v = u$, representing the case where all actions happened to have their maximum durations, while *AllMin* uses $v = l$. Both can be very useful, but even their combination does not catch all cases: In the cooking example, we would miss the case where driving takes maximal time and cooking takes minimal time. Similarly, surveillance planning would miss cases such as one airplane finishing its sweeps in minimal time while its replacement requires maximal time to reach the surveillance area.

To add the *AllMax* projection to the CSP:

- For each node $n_i \in N$, create a corresponding constraint variable tmax_i representing its execution time.
- For each requirement edge $n_i \xrightarrow{[l,u]} n_j$ in E_R, create a constraint $l \leq \mathsf{tmax}_j - \mathsf{tmax}_i \leq u$.
- For each action a with invocation and effect nodes $n_i = \mathit{inv}(a)$ and $n_j = \mathit{eff}(a)$, and each value c of $\mathit{ctrl}(a)$, create a constraint $\mathsf{ctrl}_a = c \rightarrow \mathsf{tmax}_j - \mathsf{tmax}_i = \mathit{maxdur}(a, c)$.
- Finally, let $n_i = \mathit{inv}(a_0)$ and $n_j = \mathit{eff}(a_0)$. Then add the constraints $\mathsf{tmax}_i = 0$ and $\mathsf{tmax}_j = 0$, representing the fact that the plan starts at time 0.

For *AllMin*, replace tmax with tmin and use $\mathit{mindur}(a, c)$. Time variables are separate, but resources and projections are connected through shared control variables ctrl_a.

Performance and Redundancy. While the problem formulation above contains all information that is required to characterize correct solutions, many constraint solvers (and many planners) can benefit from a bit of targeted redundancy in order to find solutions more quickly. In our implementation, based on the JaCoP solver (http://jacop.osolpro.com/), we have found the following additional information to be of assistance in the *AllMin* projection:

- For each action $a_i \in A$, a constraint variable dmin_i representing the lower bound on its duration.
- For each action $a_i \in A$ and each value c of $\mathit{ctrl}(a)$, create a constraint $\mathsf{ctrl}_a = c \rightarrow \mathsf{dmin}_i = \mathit{mindur}(a, c)$.
- For each *finished* temporal coverage goal $\langle t_1, t_2, f \rangle$ intended to be covered by the actions $a_{k_1}, \ldots, a_{k_n}$, the constraint $\mathbf{sum}(\mathsf{dmin}_{k_1}, \ldots, \mathsf{dmin}_{k_n}) \geq t_2 - t_1$.

This permits the solver to more efficiently rule out control parameter values that do not allow finished interval goals to be completely covered when uncertain durations have their worst-case (minimal-duration) outcomes.

11 Empirical Evaluation

We will now analyze and evaluate the presented approach to supporting temporal uncertainty, control parameters, and

non-linear resource consumption. To this end we have generated a total of 576 instances of the surveillance problem, varying the following parameters:

- Number of airplanes: $\{3, 4, 5\}$.
- Start of the first surveillance interval: $\{1800, 3600\}$ seconds. This has a large effect on the required speed for the initial flight, and thereby the initial fuel consumption.
- Total surveillance duration: $\{600, 1200, 1800, 3600, 5400, 7200, 9000, 10800\}$ seconds (10 minutes to 3 h).
- Flight speed discretization: $\{10, 20, 30, 40\}$ different flight speed levels.
- Fuel capacity for an airplane: $\{2500, 3000, 3500\}$ kg. The effect on surveillance endurance is significant, as a 500 kg reserve is required and part of the fuel is used to fly to and from the surveillance area.

Benchmarks are single-threaded and were run on an older compute cluster running AMD Opteron 6174 processors. Modern CPUs are typically around 3 times faster per core. TFPOP, E2IDC and the JaCoP constraint solver (http://jacop.osolpro.com/) are implemented in Java (version 8). Solutions contained 34 to 366 actions, mainly depending on duration.

Evaluation Purpose. The evaluation will not be based on comparisons between different planners. This is partly because no other planner directly supports this particular combination of features as is discussed in the related work section, and partly because the most expressive planners tend to use hand-coded guidance, whether it is expressed as control formulas or HTN tasks and methods. As identical guidance could not be used, comparisons would be less than enlightening. We therefore focus on the following aspects.

What is the impact of interleaving constraint propagation and DC verification? Figure 4 shows, for all solvers, the proportion of problem instances solvable within a given period of time. The "non-incremental" curve shows the baseline (solver 1), with no interleaving. To allow this solver to produce plans we added problems with artificially low surveillance durations of 600 / 1200 seconds, which could be solved within 7 minutes, and 1800 seconds, which could not be solved within several hours. As expected, this generate-and-test solver is unable to handle more complex problems.

Interleaving DC verification (solver 2) results in an intermediate curve ("Incremental"), where no problem instance requires more than 1140 sec (19 minutes). The instances solved by the baseline can now be solved within 1.1 sec.

What is the impact of introducing temporal projections? As shown by the topmost line in Figure 4, this provides an additional performance improvement, if not as significant as that of interleaving DC verification. The maximum time requirement is now reduced to 2.5 minutes. A closer inspection of the constraint solving process shows that much of this is due to enabling longer chains of inferences where time and resources interact through control parameters.

What is the overall impact of temporal uncertainty? Profiling shows that for shorter plans, at least 80% of the CPU time is spent on DC verification in the final solver. For longer plans the proportion rises to above 95%. This is also in line

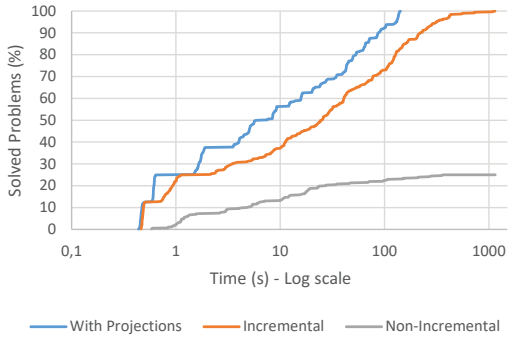

Figure 4: Performance of Three Solvers

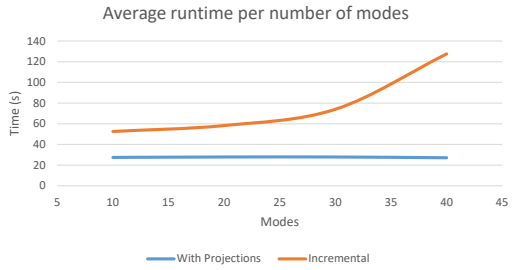

Figure 5: The Impact of Discretization

with expectations, E2IDC requires time cubic in the number of nodes, which increases linearly with plan size.

What is the impact of a denser discretization of control parameters? As shown in Figure 5, the incremental solver on average takes slightly over twice as much time with 40 distinct airspeeds compared to 10.

In contrast, runtimes for the final solver are close to independent of the number of airspeeds. In this domain, the interaction between time and resources allows the planner to efficiently detect feasible parameter values.

12 Conclusions

We have presented a new planner with support for control parameters affecting uncertain bounds on action durations as well as on non-linear resource consumption. Extracting these specific aspects of planning into a separate problem allows the main planner to focus on the non-metric problem of finding actions that can be combined to achieve the final goal, without commitment to control parameterization. A combination of constraint solving and efficient algorithms for dynamic controllability verification in STNUs is then used to find suitable control parameters allowing these actions to satisfy temporal and resource constraints.

13 Acknowledgments

This work is partially supported by Vinnova NFFP6 Project 2013-01206, the Swedish Research Council (VR) Linnaeus Center CADICS, the ELLIIT network organization for Information and Communication Technology, and the Swedish

Foundation for Strategic Research (Smart Systems: RIT 15-0097).

References

Bacchus, F., and Ady, M. 1999. Precondition control. http://www.cs.toronto.edu/%7Efbacchus/Papers/BApre.pdf.

Benton, J.; Coles, A.; and Coles, A. 2012. Temporal Planning with Preferences and Time-Dependent Continuous Costs. In *Proc. 22nd International Conference on Automated Planning and Scheduling*, 2–10. AAAI Press.

Cimatti, A.; Do, M.; Micheli, A.; Roveri, M.; and Smith, D. E. 2018. Strong temporal planning with uncontrollable durations. *Artificial Intelligence* 256:1–34.

Coles, A.; Coles, A.; Fox, M.; and Long, D. 2009. Temporal Planning in Domains with Linear Processes. In *Proc. 21st International Joint Conference on Artifical Intelligence (ICAPS)*, 1671–1676. Morgan Kaufmann Publishers Inc.

Coles, A.; Coles, A.; Fox, M.; and Long, D. 2010. Forward-Chaining Partial-Order Planning. In *Proc. 20th International Conference on Automated Planning and Scheduling (ICAPS)*, 42–49. AAAI Press.

Cui, J., and Haslum, P. 2017. Dynamic controllability of controllable conditional temporal problems with uncertainty. In *Proc. 27th International Conference on Automated Planning and Scheduling (ICAPS)*.

Cui, J.; Yu, P.; Fang, C.; Haslum, P.; and Williams, B. C. 2015. Optimising bounds in simple temporal networks with uncertainty under dynamic controllability constraints. In *Proc. 25th International Conference on Automated Planning and Scheduling (ICAPS)*, 52–60.

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

Doherty, P., and Kvarnström, J. 2008. Temporal Action Logics. In Lifschitz, V.; van Harmelen, F.; and Porter, F., eds., *The Handbook of Knowledge Representation*. Elsevier. chapter 18.

Dvorák, F.; Barták, R.; Bit-Monnot, A.; Ingrand, F.; and Ghallab, M. 2014. Planning and Acting with Temporal and Hierarchical Decomposition Models. In *Proc. 26th International Conference on Tools with Artificial Intelligence*, 115–121. IEEE Computer Society.

Edelkamp, S., and Hoffmann, J. 2004. PDDL2.2: The language for the classical part of the 4th international planning competition. Technical report.

Fernández-González, E.; Karpas, E.; and Williams, B. C. 2017. Mixed discrete-continuous planning with convex optimization. In *Proc. 31st AAAI Conference on Artificial Intelligence*, 4574–4580.

Kvarnström, J. 2011. Planning for Loosely Coupled Agents Using Partial Order Forward-Chaining. In Bacchus, F.; Domshlak, C.; Edelkamp, S.; and Helmert, M., eds., *Proc. 21st International Conference on Automated Planning and Scheduling (ICAPS)*. AAAI Press.

Li, H. X., and Williams, B. C. 2008. Generative planning for hybrid systems based on flow tubes. In *Proc. 18th International Conference on Automated Planning and Scheduling (ICAPS)*.

Liu, D.; Wang, H.; Qi, C.; Zhao, P.; and Wang, J. 2016. Hierarchical task network-based emergency task planning with incomplete information, concurrency and uncertain duration. *Knowledge-Based Systems* 112:67–79.

Mayer, M. C.; Orlandini, A.; and Umbrico, A. 2016. Planning and execution with flexible timelines: a formal account. *Acta Informatica* 53(6-8):649–680.

Morris, P.; Muscettola, N.; and Vidal, T. 2001. Dynamic control of plans with temporal uncertainty. In *Proc. 17th International Joint Conference on Artificial Intelligence (IJCAI)*, 494–499. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Nilsson, M.; Kvarnström, J.; and Doherty, P. 2016. Efficient Processing of Simple Temporal Networks with Uncertainty: Algorithms for Dynamic Controllability Verification. *Acta Informatica* 53(6-8):723–752.

Pralet, C.; Verfaillie, G.; Maillard, A.; Hebrard, E.; Jozefowiez, N.; Huguet, M.; Desmousceaux, T.; Blanc-Paques, P.; and Jaubert, J. 2014. Satellite data download management with uncertainty about the generated volumes. In *Proc. Twenty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2014, Portsmouth, New Hampshire, USA, June 21-26, 2014*.

Qi, C.; Wang, D.; Muoz-Avila, H.; Zhao, P.; and Wang, H. 2017. Hierarchical task network planning with resources and temporal constraints. *Knowledge-Based Systems* 133:17–32.

Savas, E.; Fox, M.; Long, D.; and Magazzeni, D. 2016. Planning Using Actions with Control Parameters. In *Frontiers in Artificial Intelligence and Applications*, volume 285, 1185–1193.

Stock, S.; Mansouri, M.; Pecora, F.; and Hertzberg, J. 2015. Online Task Merging with a Hierarchical Hybrid Task Planner for Mobile Service Robots. In *Proc. International Conference on Intelligent Robots and Systems (IROS), 2015*, 6459–6464. IEEE.

Tsang, E. 2014. *Foundations of constraint satisfaction*. Books on Demand.

Umbrico, A.; Cesta, A.; Mayer, M. C.; and Orlandini, A. 2017. PLATINUm: A new framework for planning and acting. In *Proc. Conference of the Italian Association for Artificial Intelligence (AI*IA)*, 498–512.

Umbrico, A.; Orlandini, A.; and Mayer, M. C. 2015. Enriching a temporal planner with resources and a hierarchy-based heuristic. In *Proc. 14th Conference of the Italian Association for Artificial Intelligence (AI*IA)*, 410–423.

Vidal, T., and Ghallab, M. 1996. Dealing with uncertain durations in temporal constraints networks dedicated to planning. In *Proc. 12th European Conference on Artificial Intelligence (ECAI)*, 48–52.

Weld, D. S. 1994. An introduction to least commitment planning. *AI magazine* 15(4):27.

Zhao, P.; Wang, H.; Qi, C.; and Liu, D. 2017. HTN planning with uncontrollable durations for emergency decision-making. *Journal of Intelligent & Fuzzy Systems* 33(1):255–267.