Proceedings of the 1st Workshop on

Planning in Continuous Domains

PCD 2013

Rome, Italy - June 11, 2013

Edited By:

Maria Fox, Derek Long, Daniele Magazzeni,
Brian Williams, Masahiro Ono

Organizing Commitee

Maria Fox
King's College London, UK

Derek Long
King's College London, UK

Daniele Magazzeni
King's College London, UK

Brian Williams
Massachusetts Institute of Technology, USA

Masahiro Ono
Massachusetts Institute of Technology, USA

Program committee

Alessandro Cimatti, FBK-irst, Italy
Maria Fox, King's College London, UK
Emilio Frazzoli, MIT, USA
Michael Hofbaur, UMIT, Austria
Hadas Kress-Gazit, Cornell University, USA
Martin Leucker, University of Lübeck, Germany
Derek Long, King's College London, UK
Daniele Magazzeni, King's College London, UK
Masahiro Ono, MIT, USA
Erion Plaku, Catholic University of America, USA
Gregory Provan, University College Cork, Ireland
Stefan Ratschan, Czech Academy of Sciences, Czech Republic
Wheeler Ruml, University of New Hampshire, USA
Martin Sachenbacher, Technische Universität München, Germany
Scott Sanner, NICTA, Australia
Louise Travé-Massuyès, LAAS-CNRS, France
Brian C. Williams, MIT, USA

Foreword

This volume contains the papers presented at PCD 2013, the First Workshop on Planning in Continuous Domains, which gathered together experts from planning and related disciplines including model-based reasoning, control and verification.

The motivation of the workshop is that real domains are often described in terms of both logical change and physical dynamics, and planning with such hybrid domains is an important challenge. One way to address robust control of hybrid systems is to establish bi-directional communication between the continuous control of the sub-systems and the supervisory control of the overall system. The form of the communications, and the resulting type of integration between the different levels of control, have been topics of robotics research around the world since Shakey was developed at SRI in the late 1960s. However, if the goal is to have intelligent robots that can plan, act, react to world and replan in a robust, effective and enduring way, existing methods have not solved the problem.

Making this work is an important challenge for AI Planning, because a seamless connection between high level goal-directed reasoning and sub-system control is required if ever planning is to provide the supervisory control of persistently autonomous robots operating competently in human environments.

The papers in this volume explore new approaches to planning and verification with hybrid systems and integrating planning and control.

Maria Fox, Derek Long, Daniele Magazzeni,
Masahiro Ono, Brian Williams
Workshop Organizers
June 2013

Table of Contents

SMT-based Verification of Hybrid Systems

Alessandro Cimatti
Fondazione Bruno Kessler
Trento, Italy

Complex embedded systems are increasingly present in our daily lives, whenever a computer-based system interacts with some physical plant or environment. Some application domains of interest are industrial production, automotive, railways, and aerospace. The key feature of such complex system, often known as hybrid systems, is the combination of discrete dynamics (e.g. from the control logic) and the continuous dynamics (e.g. from the physical system). Discrete dynamics represent, for example, control states and operation modes, while continuous dynamics take into account the physical aspects such as duration of activities, speed and position of moving objects, and profiles for resource consumption.

The ability to reason about such systems is important in two complementary dimensions. In the design phases, there is a need to predict the behaviour of the control algorithms before they are put into operation. In the operation phases, the ability to reason about such dynamic systems is a backbone for plan generation, plan validation, plan execution and monitoring, fault detection/isolation/recovery (FDIR), and replanning.

The objective of the talk is to present a formal account for modeling hybrid systems, and a set of powerful techniques to reason about them. The presentation will be grounded in the well-studied formalism of hybrid automata, that encompasses instantaneous (mode) transitions and continuous (timed) transitions.

We will introduce a symbolic representation in form of Satisfiability Modulo Theories formulae, which can be thought of as (fragments of) first-order logic where mathematical symbols are interpreted according to suitable theories (e.g. linear arithmetic). The (combinational) backends are SMT solvers, that can be seen as a tight integration of SAT (to deal with the boolean reasoning) with dedicated constraints solvers (to deal with theory reasoning).

The algorithms for reasoning about hybrid systems are able to carry out various forms of reachability analysis, and can be classified in two types. The basic ones, that lift to the case of SMT the SAT-base algorithms developed for the case of finite state model checking (including for example bounded model checking, induction, interpolation-based analysis, and IC3).

More advanced ones take into account the distinguishing features of networks of hybrid automata. We concentrate on the problem of scenario-based verification, i.e. checking if a network of hybrid automata accepts some desired interactions among the components, expressed as Message Sequence Charts (MSCs). We conclude by investigating the problem of requirements analysis for hybrid systems.

Sampling-based Motion Planning with Temporal Logic Specifications

Sertac Karaman
Department of Aeronautics and Astronautics
Massachusetts Institute of Technology, USA

One of the most fundamental planning problems in robotics is the motion planning problem: find a dynamically-feasible trajectory for a robot to reach a goal configuration starting from an initial configuration while avoiding collision with obstacles.

Extensive research throughout the last decade has lead to practical sampling-based motion planning algorithms, such as the RRTs, that were demonstrated on various robotic platforms. However, existing algorithms can only handle a simple task specification: "reach the goal configuration and avoid collision with the obstacles".

In this talk, we describe motion planning problems where the objective is to fulfill a given high-level task specification given using temporal logics. These problems combine motion planning problems that involve continuous domains (such as positions, orientation, and velocities) with discrete planning problem that embed discrete task requirements (such as ordering, reachability, safety, liveness). We propose a class of sampling-based algorithms that can solve such problems with formal guarantees on completeness, computational complexity, and asymptotic optimality. We also discuss the implementation of the proposed algorithms on an autonomous car driving in urban traffic

Motion Planning with Linear Temporal Logic for Underwater Vehicles Operating in Constrained Environments

Erion Plaku[1] and **James McMahon**[1,2]
[1]Dept. of Electrical Engineering and Computer Science
Catholic University of America, Washington DC 20064 USA
[1,2]U.S. Naval Research Laboratory, Washington DC 20375 USA

As maritime operations shift to the littoral, it becomes increasingly important to enhance the mission and motion-planning capabilities of AUVs. Operating in the littoral brings new challenges which are not adequately addressed by existing approaches. As the littoral is characterized by highly-trafficked zones, the AUV must operate near the ocean floor in order to avoid collisions with medium- to small-sized ships. Operations are further complicated when tankers are present, which often cause large drafts, or when fishing vessels are trawling, which could cause the AUV to get entangled. Additional complications arise from boulders, wreckage, and other objects found at the bottom of the ocean, sudden changes in elevation of the ocean floor, and varying ocean currents associated with tides and weather patterns. Dealing with these challenges requires efficient motion planning that take into account the vehicle dynamics and quickly plan feasible motions that enable the AUV to operate close to the ocean floor while avoiding collisions.

The demand to enhance the capabilities of AUVs operating in the littoral becomes more pressing when coupled with critical missions, such as monitoring harbors or searching for mines and other objects of interest. AUV operations, however, are currently limited to simple missions given in terms of following a set of predefined waypoints or covering an area of interest by following specific motion patterns. Conventional approaches to motion planning for AUVs based on potential fields, numeric optimizations, A*, or genetic algorithms have generally focused on largely unobstructed and flat environments where dynamics and collision avoidance do not present significant problems. As a result, while conventional approaches are able to plan optimal paths in 2D environments, they become inefficient when dealing with the challenges arising from dynamics and the need to operate close to the ocean floor in constrained 3D environments. The restrictions to follow predefined waypoints or motion patterns and to operate in essentially flat environments greatly limits the feasibility of such approaches being used for complex missions in constrained 3D environments.

To enhance the mission and motion-planning capabilities of AUVs, this work develops an approach that makes it possible to express maritime operations in Linear Temporal Logic (LTL) and automatically plan collision-free and dynamically-feasible motions to carry out such missions. As missions are characterized by events occurring across a time span, LTL allows for the combination of these events with logical (and, or, not) and temporal operators (always, eventually, until, next). As an illustration, the mission of inspecting several areas of interest while avoiding collisions can be expressed in LTL as

$$\Box \pi_{safe} \wedge \Diamond \pi_{A_1} \wedge \ldots \wedge \Diamond \pi_{A_n},$$

where π_{safe} denotes the proposition "no collisions" and π_{A_i} denotes "searched area A_i." As another example, searching A_1 or A_2 before A_3 or A_4 is written as

$$\Box \pi_{safe} \wedge ((\neg \pi_{A_3} \wedge \neg \pi_{A_4}) \cup ((\pi_{A_1} \vee \pi_{A_2}) \wedge \bigcirc (\pi_{A_3} \vee \pi_{A_4}))).$$

The expressive power of LTL makes it possible to consider sophisticated missions. This makes planning even more challenging. Motions need to be coupled with decision making to determine the best course of action to accomplish the mission. The discrete aspects, which relate to determining the propositions that need to be satisfied to obtain a discrete solution, are intertwined with the continuous aspects, which need to plan collision-free and dynamically-feasible motions in order to implement the discrete solutions.

To address the challenges imposed by the intertwined dependencies of the discrete and continuous aspects of the combined planning problem, the proposed approach simultaneously conducts the search in the discrete space of LTL sequences and the continuous space of feasible motions. A key aspect of the technical contribution is the introduction of roadmap abstractions in configuration space to guide a sampling-based exploration of the state space. The roadmap is constructed by sampling collision-free configurations and connecting neighboring configurations with simple collision-free paths. The roadmap represents solutions to a simplified motion-planning problem that does not take into account the vehicle dynamics. These solutions are converted into heuristic costs and heuristic paths to guide sampling-based motion planning as it expands a tree of feasible motions in the state space, taking into account the vehicle dynamics, drift caused by ocean currents, and collision avoidance. The roadmap abstraction also facilitates the decision-making mechanism by providing estimates on the feasibility of reaching intermediate goals as part of the overall mission specified by the LTL formula. Simulation experiments with accurate AUV models carrying out different missions provide promising validation.

p-Sulu, a Risk-Sensitive Plan Executive: from Building to Spacecraft

Masahiro Ono
Masahiro.Ono@jpl.nasa.gov
Jet Propulsion Laboratory
4800 Oak Grove Drive
Pasadena, CA 91109 USA

Brian C. Williams
williams@mit.edu
Massachusetts Institute of Technology
77 Massachusetts Avenue
Cambridge, MA 02139 USA

Abstract

This paper gives an overview of a new risk-sensitive, continuous plan executive called p-Sulu, as well as its full-horizon variant, the p-Sulu Planner. p-Sulu and the p-Sulu Planner provide two capabilities: 1) goal-directed planning in a continuous domain and 2) risk-sensitive planning that respects a user-specified upper bound on the probability of constraint violations. They both take as an input a new plan representation called a chance-constrained qualitative state plan (CCQSP), through which users specify the desired evolution of the plant state as well as the acceptable level of risk. Given a CCQSP, p-Sulu performs a real-time plan execution over a relatively short, receding planning horizon, while the p-Sulu Planner generates a sequence of actions over the entire duration of the plan in one shot, with paying more attention to optimality. We also present a distributed extension of p-Sulu, called dp-Sulu. We have deployed p-Sulu, dp-Sulu, and the p-Sulu Planner on various systems, including aerial vehicles, space vehicles, as well as buildings, and demonstrated their capabilities by simulations and experiments.

Overview

p-Sulu is a risk-sensitive plan executive that works in a continuous domain. In the presence of stochastic exogenous disturbance, p-Sulu generates a sequence of actions that guarantees that the risk of violating a given set of state constraints is below a user-specified bound. For example, the position and velocity of an unmanned aerial vehicle is uncertain due to potential wind turbulence. p-Sulu plans a path that guarantees that the probability of penetrating into a no-fly zone (NFZ) is below 1%, for example, while achieving given time-evolved goals, as illustrated in Figure 1. The objective of this paper is to give a high-level overview of p-Sulu, as well as references for interested readers.

Input: CCQSP

p-Sulu takes as an input a new plan representation called a *chance-constrained qualitative state plan (CCQSP)*. It is an extension of qualitative state plan (QSP), developed and used by Léauté and Williams (2005). CCQSP specifies a desired evolution of the plant state over time, and is defined by a set of discrete *events*, a set of *episodes*, which impose constraints on the plant state evolution, a set of *temporal*

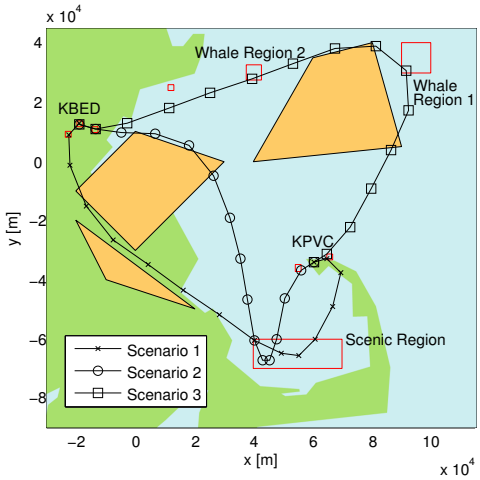


Figure 1: A PTS flight scenarios and planning results.

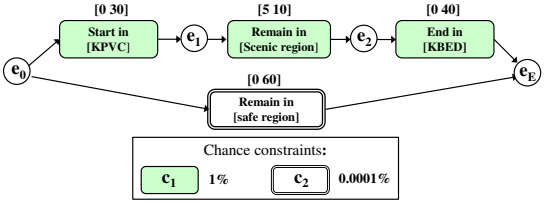


Figure 2: An example of a CCQSP for the PTS scenario in Figure 1.

constraints between events, and a set of *chance constraints* that specify reliability constraints on the success of sets of episodes in the plan.

A CCQSP may be depicted as a directed acyclic graph, as shown in Figure 2. The circles represent events and squares represent episodes. Flexible temporal constraints are represented as a simple temporal network (STN) (Dechter, Meiri, & Pearl, 1991), which specifies upper and lower bounds on the duration between two events (shown as the pairs of numbers in parentheses). The plan in Figure 2 describes the PTS scenario depicted in Figure 1, which can be stated informally

as:

> "Start from Provincetown (KPVC), reach the scenic region within 30 time units, and remain there for between 5 and 10 time units. Then end the flight in Bedford (KBED). The probability of failure of these episodes must be less than 1%. At all times, remain in the safe region by avoiding the no-fly zones and the storm. Limit the probability of penetrating such obstacles to 0.0001%. The entire flight must take at most 60 time units."

Outputs

The p-Sulu Planner has two outputs. The first is an executable control sequence over a planning horizon that satisfies all constraints specified by the input CCQSP. In the case of the PTS scenario, the outputs are the vehicle's actuation inputs, such as acceleration and ladder angle, that result in the nominal paths shown in Figure 1. The second output is the optimal schedule, a set of execution time steps for events in the input CCQSP that minimizes a given cost function. In the case of the PTS scenario, a schedule specifies when to leave the scenic region and when to arrive at Bedford, for example.

Approach

Many existing AI planners handle continuous variables by discretizing them. In contrast, our approach is to perform planning directly in a continuous domain, building upon model predictive control (MPC), an optimal control method that can consider constraints. Time evolved goals as well as temporal constraints are encoded into a set of constraints imposed on continuous variables. This approach has been originally developed by (Léauté & Williams, 2005), and extended by (Ono, 2012) to include risk constraints.

Implementations

p-Sulu has three implementations: the p-Sulu Planner, p-Sulu (in a narrow sense), and dp-Sulu. Although three implementations share the same objective with the same format of input (i.e., CCQSP), their planning approach is different, and hence they have different strengths.

The p-Sulu Planner

The p-Sulu Planner (Ono, Williams, & Blackmore, 2013) is a *full-horizon* planner, meaning that it generates a sequence of actions over the entire duration of the plan in one shot. Its strength is optimality; its solution is strictly optimal besides a relatively small conservatism introduced by risk allocation (Ono, 2012). However, it takes significantly longer computation time compared to p-Sulu. Therefore, the p-Sulu Planner is suitable for a problem that does not have non-convex state constraints, which would make numerical optimization significantly harder. The p-Sulu Planner is built upon the non-convex iterative risk allocation (NIRA) algorithm, which is described in (Ono et al., 2013).

p-Sulu

p-Sulu (Ono, Graybill, & Williams, 2012; Ono, 2012), in its narrow sense, refers to a model-based plan executive that runs on real time with a continuous replanning over a *receding* horizon. This means that, for example, at $t = 1$, it plans for $t = 1 \cdots 10$; in the next time instance, $t = 2$, it generates a plan for $t = 2 \cdots 11$, and so on. Compared to the p-Sulu Planner, p-Sulu has strength in computation time as it has to run on real time. Unlike the p-Sulu Planner, p-Sulu does not guarantee strict optimality, though in practice the solution quality of p-Sulu is comparable to the p-Sulu Planner in many cases. p-Sulu is built upon the iterative risk allocation (IRA) algorithm, which is presented in (Ono & Williams, 2008).

dp-Sulu

dp-Sulu (Ono, 2012) is an extension of p-Sulu that can control a multi-agent system in a distributed manner. Each agent is controlled by p-Sulu, while a central module supervises the entire system to resolve conflicts between agents.

Applications

p-Sulu provides a general planning capability that can be applied to a broad range of problems. In this section we introduce three applications.

Space Rendezvous

First, we present an application of the p-Sulu Planner to an autonomous space rendezvous scenario of the H-II Transfer Vehicle (HTV), shown in Figure 3. HTV is an unmanned cargo spacecraft developed by the Japanese Aerospace Exploration Agency (JAXA), which is used to resupply the International Space Station (ISS). In order to assure safety, a very detailed rendezvous sequence is prescribed, as shown in Figure 4. We encode this rendezvous sequence into a CCQSP as shown in Figure 5. In addition to the time-evolved goals specified in the actual rendezvous sequence, we specify temporal constraints and chance constraints in the simulation, as shown in the CCQSP.

Figure 3: H-II Transfer Vehicle (HTV), a Japanese unmanned cargo vehicle, conducts autonomous rendezvous with the International Space Station. Image courtesy of NASA.

Figure 6 shows the planning result of the p-Sulu Planner. We compare the result with Sulu, a deterministic predecessor of p-Sulu, as well as a nominal planning approach, in

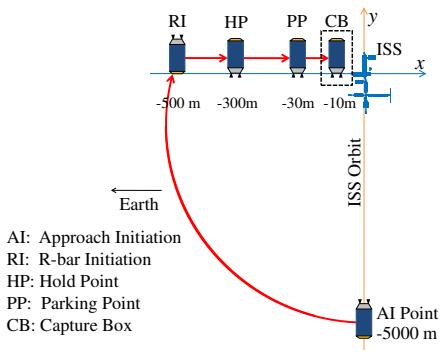


Figure 4: HTV's final approach sequence (Japan Aerospace Exploration Agency, 2009).

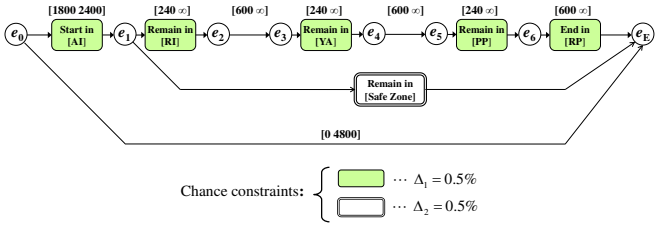


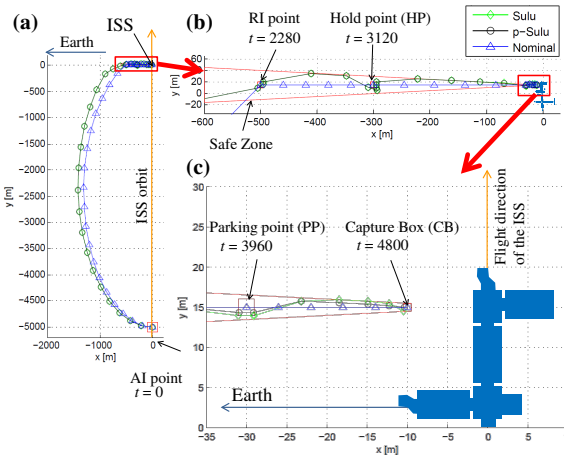
Figure 5: A CCQSP representation of the HTV's final approach sequence. We assume the same time-evolved goals as the ones used for actual flight missions. The temporal constraints and the chance constraints are added by the authors.

which we assume that HTV moves from AI to RI using a two-impulse transition (called "CW guidance law") (Matsumoto, Dubowsky, Jacobsen, & Ohkami, 2003; Vallado, 2001). From RI to CB, it follows a predetermined path that goes through the center of the Safe Zone, as shown in Figure 6-(b), with a constant speed.

Our simulation results show that the probabilities of failure of the path generated by the p-Sulu Planner are less than the risk bounds specified by the CCQSP. On the other hand, Sulu's path results in almost 100% probability of failure. This is because Sulu minimizes the fuel consumption without considering uncertainty. The results also show that the plan generated by the p-Sulu Planner achieves 1.42 m/sec reduction in Delta V compared to the nominal approach, which is equivalent to an 11.9 kg saving of fuel, assuming the $16,500$ kg mass of the vehicle and the 200 sec specific impulse (I_{SP}) of the thrusters. The computation time was 11.4 seconds. Please refer to (Ono et al., 2013) for more detailed presentation of the results.

Building Control

Improving the energy efficiency of residential buildings plays a significant role in addressing the global climate challenge. In the U.S., for example, residential buildings accounted for 21.52% of total energy usage in the country in 2008 (U.S. Energy Information Administration, 2010).



AI: Approach Initiation, RI: R-bar Initiation, YA: Yaw-around

Figure 6: Planning results of Sulu, the p-Sulu Planner, and a nominal planning approach. The input CCQSP is shown in Figure 5.

Heating and cooling accounted for the largest portion of the residential energy consumption: 7.99 quadrillion Btu or 38.2% of the energy consumption in the residential sector.

We deploy p-Sulu on a smart building called Connected Sustainable Home, shown in Figures 7. The objective is to optimally control the indoor temperature so that energy saving is maximized while limiting the risk of violating constraints on room temperature.

Figure 7: Artist's concept of the Connected Sustainable Home.

Figure 8 shows an example of a CCQSP, where the three temperature states are defined as in Figure 9. Given a CCQSP, p-Sulu outputs an optimal profile of indoor temperature over a day, as well as a suggestion on the schedule of the resident to minimize the energy use, if the residents have flexibility in their schedule. Our simulation studies demonstrated that p-Sulu achieves significant savings in the energy consumption over a classical set-point controller with a fixed set point, while dramatically improving robustness over a deterministic plan executive, Sulu. The results are presented in detail in (Ono et al., 2012).

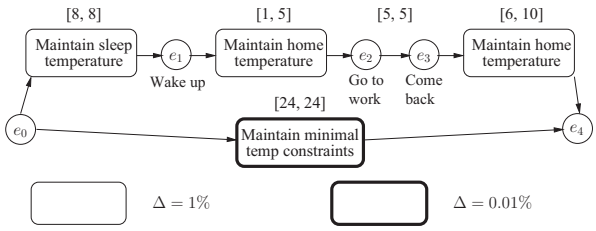


Figure 8: An example of a CCQSP for a resident's schedule in a planning problem for the Connected Sustainable Home. The event e_0 represents the midnight. The CCQSP requires that the resident must go out for work for five hours, but the time to go to work is flexible between 9 a.m. and 1 p.m.

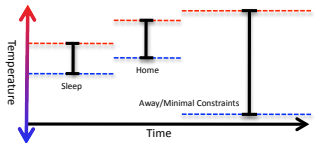


Figure 9: The state space of the CCQSP in Figure 8.

Aerial Personal Transportation System

Finally, we present an application of dp-Sulu for Personal Transportation System (PTS), envisioned by Boeing as shown in Figure . The PTS consists of a fleet of small personal aerial vehicles (PAV) that enable the flexible point-to-point transportation of individuals and families. The flight plan for each PAV is specified by a CCQSP, as shown in Figure 2. We also use CCQSP to specify constraints that involves multiple vehicles, such as minimum time interval between two landings and collision avoidance. dp-Sulu computes optimal flight paths as well as schedules in a distributed manner, as illustrated in Figure 11. The simulation results are presented in detail in (Ono, 2012).

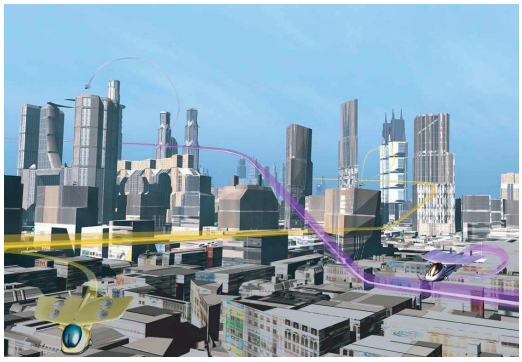


Figure 10: Personal Transportation System (PTS). (Courtesy of the Boeing Company)

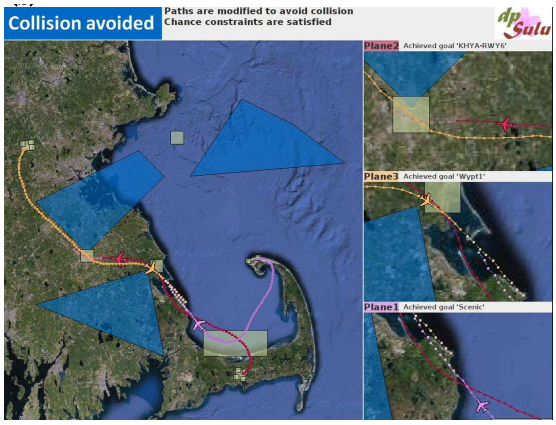


Figure 11: A screen shot of dp-Sulu controlling a fleet of three personal aerial vehicles

Conclusions

This paper gave a high-level overview of the p-Sulu Planner, p-Sulu, and dp-Sulu, three implementation of a risk-sensitive planner that works in a continuous state space. We presented three applications of the planners in three domains: space rendezvous, building control, and an aerial personal transportation system.

Acknowledgments

This paper is based upon work supported in part by the Boeing Company under Grant No. MIT-BA-GTA-1, the National Science Foundation under Grant No. IIS-1017992, and by Siemens AG under Addendum ID MIT CKI-2010-Seed_Fund-008. The research on Connected Sustainable Home was conducted within the *Green Connected Home Alliance* between the Mobile Experience Lab, at the Massachusetts Institute of Technology and the Fondazione Bruno Kessler in Trento, Italy. The research described in this paper was performed when the first author was at MIT and Keio University, and is not related to the work he performs at Jet Propulsion Laboratory. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author and do not necessarily reflect the view of the sponsoring agencies.

References

Dechter, R., Meiri, I., & Pearl, J. (1991). Temporal constraint networks. *Artificial Intelligence*, *49*, 61–95.

Japan Aerospace Exploration Agency (2009). HTV-1 mission press kit. Available on-line at `http://www.jaxa.jp/countdown/h2bf1/pdf/presskit_htv_e.pdf`.

Léauté, T., & Williams, B. C. (2005). Coordinating agile systems through the model-based execution of temporal plans. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI)*.

Matsumoto, S., Dubowsky, S., Jacobsen, S., & Ohkami, Y. (2003). Fly-by approach and guidance for uncontrolled rotating satellite capture. In *Proceedings of AIAA Guidance, Navigation, and Control Conference and Exhibit*.

Ono, M. (2012). *Robust, Goal-directed Plan Execution with Bounded Risk*. Ph.D. thesis, Massachusetts Institute of Technology.

Ono, M., Graybill, W., & Williams, B. C. (2012). Risk-sensitive plan execution for connected sustainable home:. In *Proceedings of the 4th ACM Workshop On Embedded Systems (BuildSys)*.

Ono, M., Williams, B., & Blackmore, L. (2013). Probabilistic planning for continuous dynamic systems. *Journal of Artificial Intelligence Research*, *46*, 449 – 515.

Ono, M., & Williams, B. C. (2008). An efficient motion planning algorithm for stochastic dynamic systems with constraints on probability of failure. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI-08)*.

U.S. Energy Information Administration (2010). Annual energy outlook..

Vallado, D. A. (2001). *Fundamentals of Astrodynamics and Applications, Second Edition*. Microcosm Press.

Planning for Agile Earth Observation Satellites

Johannes Aldinger and Johannes Löhr
Albert-Ludwigs-Universität Freiburg
Institut für Informatik
Georges-Köhler-Allee 52
79110 Freiburg, Germany
{aldinger, loehr}@informatik.uni-freiburg.de

Abstract

Agile Earth observation satellites are satellites orbiting Earth with the purpose to gather information of the Earth's surface by slewing the satellite toward regions of interest. Constraints arise not only from dynamical and kinematic aspects of the satellite and its sensors. Regions of interest change over time and bad weather can conceal important observation targets. This results in a constant need to replan the satellite's tasks and raises the desire to automatize this planning process. We consider the Earth observation problem with the help of the module extension of the numerical planning system Temporal Fast Downward. Complex satellite slew maneuvers are calculated within modules, while the planner selects and schedules the regions to be scanned. First results encourage deeper research in this area so that forthcoming satellite space missions can draw on automated planning to improve the performance of agile Earth observation tasks.

Introduction

We are interested in the feasibility of automated planning techniques in the context of Earth observation scenarios. The task of Earth observation missions is to scan regions of interest, straight stripes referred to as *patches*, during the flyover.

A task in the context of an Earth observation mission is to select and to schedule a sequence of observation patches. Determining the sequence of patches is a rather simple discrete planning problem for current automated planning systems. However, complex numerical calculations have to be performed to determine the slew maneuver of the satellite to approach and scan a patch. The feasibility of the discrete plan tightly depends on the continuous aspects since it has to consider the satellite's orbital motion, its attitude and angular rate as well as its torque capability in realistic scenarios. Instrument alignment and required scan velocity pose additional constraints. The feasibility of slews between two successive scans depends on the satellite's attitude, angular rate and position and is varying in time. Any decision to scan a certain patch at a certain position in orbit may affect the feasibility of future scan maneuvers. This makes the problem difficult to solve in case of larger sets of patch observations. Nevertheless, it allows to decouple the deterministic planning task, form the numeric calculations which makes Earth observation an appealing task for modular planning systems e.g. Temporal Fast Downward with Modules (TFD/M) (Dornhege et al. 2009).

In recent years, potent automated planning systems emerged from the planning community. Often, these planners are tested on IPC benchmark domains. While these benchmarks are well suited to determine strengths and weaknesses of different planning systems, the potential for industrial applications has not been fully exploited yet. We aim to solve realistic problems relevant to industrial needs. Other examples of successful industrial applications include work of Penna et al. (2010) and Fox, Long, and Magazzeni (2011). Thereby, weaknesses of current planning systems are discovered, so that future research can support the applicability of real world problems.

We found the planning system TFD/M to be suitable for our purpose. In TFD, numerical calculations, in our case slew maneuvers of the satellite, are outsourced into modules, while the basic planning problem, the selection and scheduling of the patches, is performed by TFD.

Basics

In this section we define the Earth observation task. Afterwards, we will show how to solve the problem by an automated planning system. We contemplate over automated planning systems in general at first before considering the TFD/M planning system.

Earth observation

Earth observation missions are an important topic in aerospace where the goal is to scan the Earth's surface with the help of satellites. Earth observation applications include among others geodesy, cartography, climatology and weather forecast. Depending on the desired application, different sensors from radar over infrared to visual sensors are used. The patches of interest are straight stripes that have to be scanned at a constant scan velocity which depends on the sensors used. The satellite has only restricted storage capacities and has to dump collected data to a ground station from time to time. The regions of interest change over time, and weather influences the visibility of interesting patches. Earth observation tasks vary in the agility of sensors which usually depends on the sensor's weight. Some instruments can be aligned to regions of interest without altering the attitude of the satellite. *Agile* Earth observation tasks considered in this

paper carry heavy instruments which are firmly fixed to the satellite. Patches are scanned by slewing the satellite's line of site towards the regions of interest.

As of now, human experts identify reasonable and feasible maneuvers by hand. The identified maneuvers are then optimized and verified by potent physics simulation tools, before they are transmitted to the satellite.

Modeling a Planning Problem

To solve a real world planning problem with an automatized planning system, it is necessary to model it first. Formally, a planning task is defined as tuple $\langle \mathcal{V}, s_0, s_\star, \mathcal{O} \rangle$. The set of variables $\mathcal{V}$ contains Boolean variables with domain $\{\top, \bot\}$ as well as numeric variables with domain $\mathrm{dom}(v) \subseteq \mathbb{Q}$ for $v \in \mathcal{V}$. The *states* of the planning problem are assignments of all variables $v \in \mathcal{V}$ to a value in their domain and s_0 is the initial state. The goal states $s_\star$ are defined by a partial assignment over some of the logical variables. The set of operators $\mathcal{O}$ contains operator triples $\langle C, E, c \rangle$ consisting of preconditions C, effects E and an action cost c.

Driven by the International Planning Competition IPC, the predominant language to describe planning tasks is the Planing Domain Description Language (PDDL) (Ghallab et al. 1998). The interface for "semantic attachments" in PDDL (Dornhege et al. 2009) allows the addition of three types of modules to the planning domain: `conditionchecker` modules, `cost` modules and `effect` modules. `Conditionchecker` modules evaluate to logical (Boolean) variables that occur in the precondition of a planning operator. Similarly, `cost` modules represent numeric variables. When `cost` modules occur in the precondition of a planning operator, the numeric value is compared to another numerical statement with a comparison operator $\{<, \leq, =, \geq, >\}$ while it can also be used directly to determine the action cost. Finally, `effect` modules modify a set of variables in the planner state. The modified variables can be either logical or numeric variables.

Temporal Fast Downward with Modules

To solve a continuous planning problem such as the Earth observation problem, a planning system capable of dealing with numeric variables is required. Even though numeric domains can be successfully solved with a numeric planner (Löhr et al. 2012) it is favorable to outsource numeric complexity into modules.

An extension of the fast downward (FD) planning system (Helmert 2006) to allow for temporal and numerical aspects (TFD) has been proposed by Eyerich, Mattmüller, and Röger (2009). TFD extends Fast Downwards Context Enhanced Additive Heuristic to numerical variables. While TFD supports numerical state variables, heuristic estimates in numerical rich domains are coarse. To handle complex numerical processes, the calculation should be separated from the logical planning task. The modules extension TFD/M (Dornhege et al. 2009) allows to access "semantic attachments", modules that outsource the numerical calculations from the logical planning level. TFD/M interleaves the causal planning problem "what to do" with the numerical task "how to achieve it". Neither a top-down nor a bottom-up approach can satisfy the interdependency between low level calculations and high level plan structure, and we therefore rely on an interleaved approach. A classical top-down decomposition of the planning task solves the problem on an abstract symbolic domain, and then refines that symbolic plan. In the case of Earth observation the planner would first schedule the sequence of patches to be scanned, while the maneuvers to follow this sequence would then be calculated in a refinement step. The drawback of a top-down approach is, that high-level solutions can be incorrect or pose contingencies for low-level planners. Even if the maneuvers are feasible, the resulting plan is unlikely to be good. The opposite approach, a bottom-up decomposition, precomputes *all* refined solutions, so that a higher level symbolic planner can then draw on the lower level plans. While the resulting plans are usually optimal, precomputing all low level solutions requires excessive memory and runtime. In continuous settings there are infinitely many low level plans. In the Earth observation scenario, all possible maneuvers would have to be precomputed which is intractable even for coarse discretizations. The semantic attachments of TFD evaluate the decomposition of a symbolic action on demand and can thus involve the interdependency between high level symbolic actions and low level numeric calculations. This allows TFD/M to solve an Earth observation task as presented in the next section.

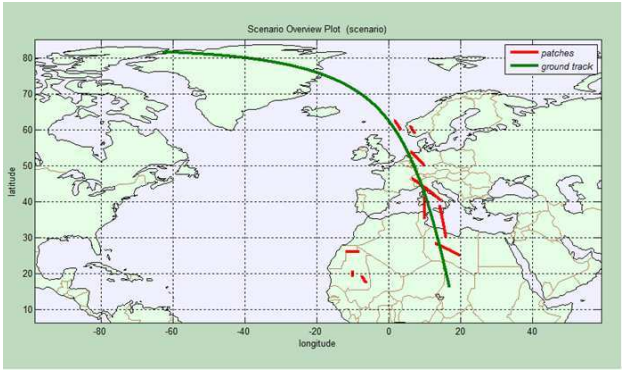


Figure 1: Earth observation scenario with subtrack of the satellite and observation sites to be scanned.

Planning the Earth observation task

We described the Earth observation scenario problem, and a tool to solve it: TFD/M. When solving Earth observation problems with TFD/M, the continuous world has to be discretized. Then, we can exploit the strength of modern planning systems: the selection and scheduling of the good actions.

Our general framework is a three step process. At first, we precompile the real world problem into a planning task, at second solve this planning task with TFD/M and finally verify the planned results with a physics simulator. The preprocessing step reduces some of the numerical complexity from the planning problem. We consider the subtrack obtained by projecting the satellite perpendicular to the earth

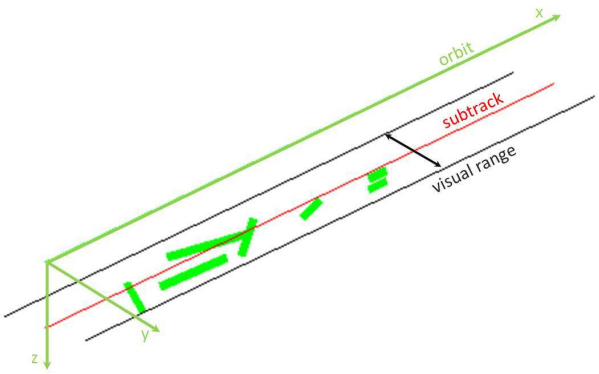


Figure 2: Planning Problem obtained from the Earth observation scenario of Figure 1 after preprocessing.

surface and peel off a stripe of the Earth's surface following the subtrack. The width of the peeled off stripe includes all patches that are within the satellite's visual range determined by its maximal angular deflection. For an example, consider the Earth observation scenario in Figure 1. The ground track of the satellite is depicted by a green line. The patches to be scanned are depicted in red. Some of the patches in north west Africa are out of range of the satellite's sensors. The planning problem that arises after preprocessing is depicted in Figure 2. The green patches correspond to the observation sites in Figure 1. In the planning problem we omit Earth's curvature and treat the surface of the precompiled problem as long plane with a satellite flying over it on an orbit depicted in olive green (Figure 2). If the problem horizon entails multiple satellite orbits, the same "patch" can be visible from different orbit positions. This results in multiple instances of the same patch in the precompiled stripe, usually in different orientation. To distinguish such patches, we use the term *observation site* for a site on Earth that has to be scanned, and use *patch* for a concrete instance observed from the current orbit.

Scanning patches corresponds to achieving soft goals because it is not always possible to scan all patches in the planning problem. Following Keyder and Geffner (2009) we introduce an action to `ignore` an observation site, which results in a high penalty cost. By modifying the penalty for ignoring a patch, the observation sites can be given different priorities. The goal of the planning problem is to *deal* with all observation sites, which can be done by either scanning a patch, or by actively ignoring it. While scanning occurs at a cost depending on the optimization criterion of the planning problem (e.g. available time or energy consumption) ignoring an observation site occurs at a much larger penalty cost.

The state variables $\mathcal{V}$ of the Earth observation planning problem contain logical variables as well as numerical ones. Some "variables" can not change their value during the planning process and we refer to them as *constants*. We use the common notion of *fluents* for variables that can be manipulated by the planning operators. The state of the Earth observation scenario contains Boolean constants (e.g. `(belongsto ?patch ?osite)` describing that a patch belongs to an observation site) Boolean fluents (e.g. `(dealt ?osite)` describing that an observation site has been processed) numerical constants describing satellite parameters (e.g. `(roll-max ?sat)` describing the maximal roll angle of the satellite) as well as numerical fluents (e.g. `(roll-angle ?sat)` describing the current roll angle of the satellite).

In the initial state s_0, no observation site has been *dealt* with, the numerical fluents describe the satellite's current orbit position, attitude and angular rates. The numeric constants model the attitude constraints of the satellite such as maximal angle deflection and maximal angular rates. The goal states $s_\star$ of the planning task are all states, where all observation sites have been *dealt* with. The planning operators $\mathcal{O}$ are `scan` to scan a patch and `ignore` to ignore an observation site.

Discretization

To model the Earth observation scenario, each state of the Earth observation planning problem describes a "snapshot" of the continuous world. Usually the satellite has just scanned a patch and the numeric state fluents describe the attitude and rate of the satellite in this position with line of sight toward the end of the patch. Discrete planning decisions are made between these snapshot states. The available actions at such a state are to either scan or to ignore one of the remaining patches. While ignoring a patch results in a discrete successor state only altering Boolean fluents, determining the successor snapshot state of the planner after applying a `scan` action is not obvious. As the world is continuous, deciding for the next patch to scan could result in infinitely many possible successor states, since it is possible to scan a patch from different positions in the orbit. To commit to one discrete successor state after deciding for a patch to scan, we make the following assumption:

Assumption 1. *It is always best to scan a patch as soon as possible.*

We assume that it is always best to scan the chosen patch as soon as possible, thus leaving wider scope for future actions. This assumption implies that it is more important to scan many patches than to scan them with a good image quality which is usually obtained when the angular deflection of the satellite's line of sight is minimal. It is not obvious how to calculate the earliest possible satellite state to scan the patch.

In the following we will show how this "earliest possible" satellite state at the start of the scan maneuver can be estimated. We will first look at the extreme positions and omit the constraints posed from other patches. Then we will propose a method based on interval nesting to determine the earliest possible orbit position to start scanning the selected patch. At first we consider the case, where the patch to scan is far away from the satellite's current orbit position. The earliest possible approach configuration of a satellite is obtained by deflecting the satellite as far possible. The maxi-

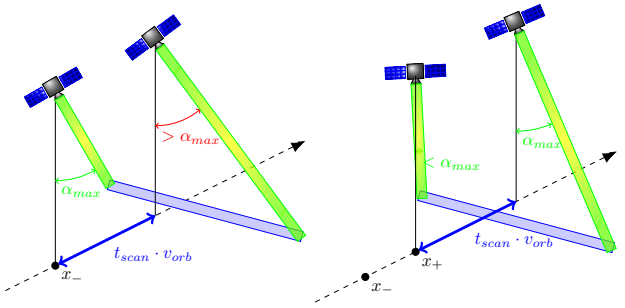


Figure 3: Satellite states to determine the earliest possible position to scan the patch

mal deflection is limited by the maximal angle under which the sensor can operate.

An example is illustrated in Figure 3. The subtrack of the satellite is depicted by the dashed line. The left satellite in the left graphic depicts the state of the satellite at the earliest orbit position x_- to scan the patch. There, the satellite is deflected with the maximal angle α_{max} towards the patch. However, the attitude dynamic constraints of the satellite could be violated by scanning the patch starting form x_-. After scanning, the satellite would be in the infeasible state depicted on the right side of the left graphic in Figure 3. In this case, the state of the satellite when exiting the patch is critical to approach the patch as early as possible. The right satellite in the graphic on the right depicts the earliest possible satellite state to finish scanning the patch. This state is reached, if the scan started at orbit position x_+. The orbit position x_+ can be calculated because the scan time needed for scanning the patch t_{scan} and the orbital velocity v_{orb} are known. Depending on the scanning speed and the orientation of the patch relative to the satellite's subtrack position, either x_- or x_+ can be the critical earliest possible orbit positions to scan the patch. The satellite state s_{first} is the state adopted at the earliest possible orbit position $max(x_-, x_+)$.

Analogously the last possible orbit position to scan the patch can be computed by minimizing over the latest attitude under which the satellite can start or end a scan, where the satellite adopts state s_{last}. All feasible maneuvers to scan the patch are in the interval between the orbit positions at s_{first} and s_{last}. In real planning problems with multiple patches it is not likely that all patches can be scanned as early as s_{first}. Instead, scanning a patch should start as soon as possible from the satellite's current state. Unfortunately, neither the satellite's orbit position nor its attitude after executing this best slew maneuver are known in advance. The principal problem of finding the earliest orbit position to start the next scan builds a non-linear equation system for which no closed form solutions methods are known to us. We therefore approximate the satellite state with the help of interval nesting.

The flow chart in Figure 4 illustrates the mathematical calculations needed inside a `scan` planning operator, which approximates the earliest orbit position to scan the patch as well as the corresponding slew maneuver. Two types of

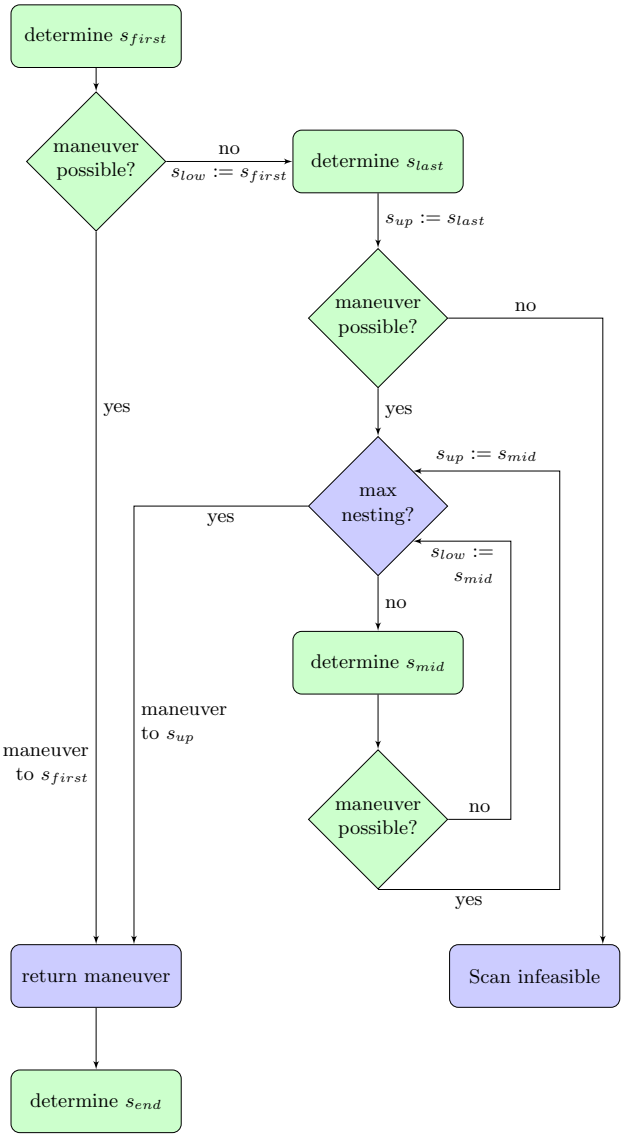


Figure 4: Inside a `scan-patch` planning operator. Green boxes can be calculated by Function 1 while green diamonds are calculated by Function 2.

mathematical calculations have to be performed frequently in an Earth observation planning task:

Function 1. *Determine the satellite's state (attitude and angular rates) from a given orbit position when pointing towards a patch, with angular rates satisfying the vectorial velocity for scanning the patch.*

Function 2. *Determine the feasibility of a maneuver given two satellite states.*

We will take a deeper look at the mathematical calculation of Function 1 and Function 2 in the next section and assume for now that both functions can be computed efficiently. The green boxes in the flow chart in Figure 4 are

all instances of Function 1 while the green diamonds can be calculated with Function 2. The orbit position after the optimal slew maneuver from the current state of the satellite to the best approach state lies in the interval between the orbit position at s_{first} and s_{last}. If a maneuver from the current satellite state $s_{current}$ to the earliest possible scan configuration is possible, this maneuver from $s_{current}$ to s_{first} is optimal, and can be returned. After scanning the patch the satellite adopts state s_{end}, with an orbit position depending on orbital velocity and the scanning time, both of which are given in the domain description. The deliberations made for s_{first} and depicted in Figure 3 ensure that s_{end} is valid. If the maneuver from $s_{current}$ to s_{first} is infeasible, it is used as lower bound s_{low} of an interval nesting, and s_{last} is calculated as latest possible orbit position to scan the patch. If the maneuver to s_{last} is infeasible, the patch can not be scanned at all. Otherwise, the satellite's state of a time optimal maneuver is found between the unreachable lower bound s_{low} and the reachable but not time optimal upper bound satellite state $s_{up} = s_{last}$. Unless the maximal nesting depth is exceeded, we determine the satellite state s_{mid} in the middle of the boundaries with the help of Function 1, and check if the maneuver from the current satellite configuration to the middle configuration is feasible with the help of Function 2. If the maneuver is feasible, a better upper bound has been found, while the s_{mid} is used as lower bound if the time slew time is exceeded.

In our implementation we limit the depth of the interval nesting to 10 which offers a good trade-off between run-time (less than 1 ms) of the operation and precision (approximately 10 m deviation). We note that more sophisticated interval nesting methods to determine the orbit position of the most promising middle configuration could be used.

While we calculate the satellite's attitude and angular rates s_{end} after scanning, we do not check if the maneuver from s_{up} to s_{end} is feasible with our method described in Function 2.

Assumption 2. *When entry and exit configuration are feasible, the whole scan maneuver is.*

The maneuver of the satellite during a scan is given by kinematical constraints and not considered here in this paper.

As mentioned earlier, the extension of the PDDL planning language allows planning operators to contain three types of modules: `conditionchecker` modules, `cost` modules and `effect` modules. In our implementation, scanning a patch is decoupled into two actions `approach-patch` and `scan-patch`. This is mostly done for technical reasons, since it is easier to determine the satellite state after each operator execution during planning which makes it easier to verify the feasibility of the plan during post processing. Within these two operations, we use five modules, which all calculate parts of the flow chart of Figure 4. To avoid the recalculation of the same function, a database stores all calculations performed by Function 1 and Function 2. The time intensive interval nesting is therefore only computed once for each configuration.

The modules executed by the `approach-patch` operator all follow the flow chart diagram (Figure 4) and basically compute the same thing. A `conditionchecker`-module `approach-patch-possible` tests, if approaching a patch is possible, a `cost`-module `approach-time` determines the maneuver time to approach the patch, and finally an `effect`-module `approach-effect` modifies the planning state and sets the planning variables of the satellite to s_{up}. The modules used in the `scan-patch` operator are a rather simple module `scan-time` that calculates the time needed for scanning. We do not need a `conditionchecker` module, since Assumption 2 ensures that the feasibility of each scan is already checked by approaching the patch. The `scan-effect` module obviously sets the planner state of the variables concerning the satellite to s_{end}. Additionally the planning operator sets the corresponding observation site `dealt` variable to true.

Satellite Attitude Dynamics

In the previous section we have identified two mathematical functions calculating the attitude and angular rates of the satellite and the feasibility of the slew maneuvers. Both have to be calculated frequently within the modules of the planner. Function 1 consists of two parts: determining the *attitude* of the satellite when pointing towards the targeted patch, and the *angular rates* of the satellite that is necessary to scan the patch. Afterwards we will present the calculations for Function 2 that checks the feasibility of the slew maneuver.

Coordinate Systems The Earth observation scenario is preprocessed from the Earth spherical coordinates (see Figure 1) to a flat cartesian coordinate system along the subtrack of the satellite (see Figure 2). The x-axis points in flight direction, the z-axis points towards center of Earth (Nadir) and the y-axis completes the right hand coordinate system. The center of the coordinate system lies within the center of mass of the satellite at time t_0. This Earth fixed planning frame is notated as N-frame. The satellite's body-fixed frame is referred to as B-frame, where the z-axis is assumed to be coaxial with the instruments line of sight. A vector $\mathbf{x}$ in N-frame is notated as $\mathbf{x}^N$, in body fixed coordinates as $\mathbf{x}^B$, respectively and its norm is denoted as $\|\mathbf{x}^N\|$. The components of a vector are identified by stating the respective axis in subscript e.g. $\mathbf{p}^N = [p_x^N, p_y^N, p_z^N]$.

Attitude Determination We define the attitude of the satellite's fixed body-frame, the B-frame by its Euler angles with respect to the N-frame. The roll angle ϕ is defined by a rotation of the satellite around the x-axis and the roll rate is referred to as $\dot{\phi}$. The pitch angle θ and pitch rate $\dot{\theta}$ are defined around the y-axis and the yaw angle ψ and yaw rate $\dot{\psi}$ are defined around the z-axis respectively. Vectors defined in the N-frame are transformed into the B-frame by the direct cosine rotation matrix

$$
\begin{aligned}
DCM^{BN} &= DCM_\theta \; DCM_\phi \; DCM_\psi, \text{where} \\
DCM_\theta &= \begin{pmatrix} \cos -\theta & 0 & \sin -\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos -\theta \end{pmatrix} \\
DCM_\phi &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos -\phi & \sin\phi \\ 0 & \sin -\phi & \cos -\phi \end{pmatrix} \\
DCM_\psi &= \begin{pmatrix} \cos -\psi & \sin\psi & 0 \\ \sin -\psi & \cos -\psi & 0 \\ 0 & 0 & 1 \end{pmatrix}.
\end{aligned} \tag{1}
$$

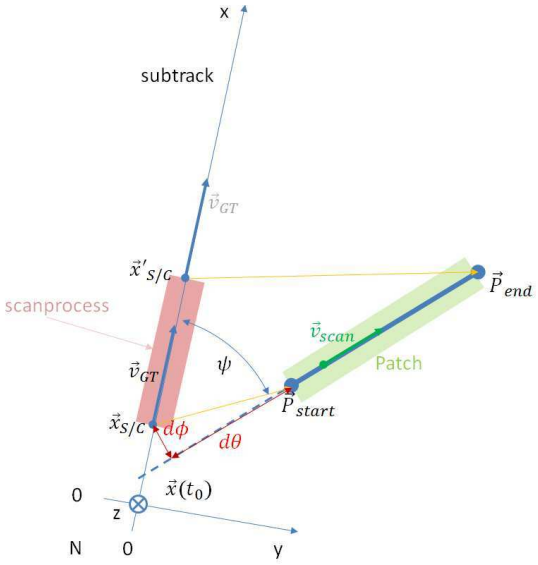


Figure 5: Geometry of an exemplary patch position

The patch to be observed is specified by a start coordinate $\mathbf{P}^N_{start}$ and an end coordinate $\mathbf{P}^N_{end}$ and has to be scanned with a constant scan velocity v_{scan}. The direction of the patch is given by

$$\mathbf{p}^N = \mathbf{P}^N_{end} - \mathbf{P}^N_{start}$$

with length $l = \|\mathbf{p}^N\|$ which leads to the scan time

$$t_{scan} = \frac{l}{v_{scan}}.$$

To scan a patch from an orbit position $\mathbf{x}^N_{S/C}$, the instrument's line of sight has to point to the start position of the patch, which corresponds to a specific attitude DCM^{BN} of the B-frame with respect to the N-frame. Additionally the angular rate of the satellite $\boldsymbol{\omega}_{S/C} = [\dot{\phi}, \dot{\theta}, \dot{\psi}]^T$ is specified by the scan velocity of the patch. The attitude and angular rate of the satellite after scanning a patch can be obtained similarly. Here the satellite has also a specified attitude DCM^{BN} and angular rate $\boldsymbol{\omega}_{S/C}$ depending on the new orbit position at $\mathbf{x}^{'N}_{S/C} = \mathbf{x}^N_{S/C} + \mathbf{v}^N_{GT}\, t_{scan}$ and the end position of the patch. Both, attitude and angular rate are functions of $\mathbf{x}^N_{S/C}$ and patch point $\mathbf{P}^N$ to be aimed at. In order to calculate the attitude DCM^{BN} we sequence rotations around the yaw axis, the roll axis and the pitch axis. The satellite yaws with angle ψ, as depicted in Figure 5.

$$\psi = \arctan \frac{\mathbf{p}^N_y}{\mathbf{p}^N_x}$$

The line of sight vector $\mathbf{r}^N = [\mathbf{P}^N - \mathbf{x}^N_{S/C}]$ points from the spacecraft to a start point or to an end point of a patch. This is firstly rotated to a auxiliary frame H with angle ψ

$$\mathbf{r}^H = DCM^{HN}_\Psi \mathbf{r}^N$$

which yields the roll angle

$$\phi = \arctan(\frac{d\phi}{h})$$

and the pitch angle

$$\theta = \arctan(\frac{d\theta}{\sqrt{h^2 + d\phi^2}}),$$

where $d\phi = r^H_y$ and $d\theta = r^H_x$. The direct cosine matrix, that defines the attitude of the satellite pointing the instrument to a start or end point of a patch, can finally be computed by Equation 1.

Angular Rate Determination The required scan velocity of the line of sight at the patch

$$\mathbf{v}^N_{scan} = v_{scan} \frac{\mathbf{p}^N}{\|\mathbf{p}^N\|}$$

is generated by the subtrack velocity $\mathbf{v}^N_{GT}$ and a compensating velocity induced by a rotation of the satellite

$$\mathbf{v}^N_{comp} = \mathbf{v}^N_{scan} - \mathbf{v}^N_{GT}.$$

Using the compensation velocity transformed to the B-frame

$$\mathbf{v}^B_{comp} = DCM^{BN} \mathbf{v}^N_{comp}$$

and the line of sight vector in body frame

$$\mathbf{r}^B = DCM^{BN} \mathbf{r}^N$$

the angular rate in body frame $\boldsymbol{\omega}^B$ can be implicitly obtained by

$$\mathbf{v}^B_{comp} = \boldsymbol{\omega}^B \times \mathbf{r}^B$$

The compensation velocity is a cross product of the satellite rotation and the line of sight vector

$$\mathbf{v}^B_{comp} = \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ \|\mathbf{r}\| \end{bmatrix} = \begin{bmatrix} \omega_y \|\mathbf{r}\| \\ -\omega_x \|\mathbf{r}\| \\ 0 \end{bmatrix}$$

which leads to $\omega^B_y = \frac{\mathbf{v}^B_{comp.x}}{\|\mathbf{r}\|}$ and $\omega^B_x = -\frac{\mathbf{v}^B_{comp.y}}{\|\mathbf{r}\|}$. The angular rate in N-frame finally is given by

$$\boldsymbol{\omega}^N = DCM^{BN} \boldsymbol{\omega}^B.$$

Slew Feasibility We are interested to check the feasibility of a slew maneuver between two arbitrary patches meaning a maneuver that slews the satellite from an arbitrary initial attitude and angular rate to a desired attitude and angular rate. Ideally the necessary torque profile in B-frame is analysed. This is computationally expensive and exceeds the acceptable runtime of modules by far. Therefore we introduce conservative assumptions to be able to quickly check the feasibility of a slew.

Note on Maximum Torques Torques $\mathbf{T}$ acting on a rigid body and the resulting angular rates $\boldsymbol{\omega}$ in body fixed coordinates are connected by the well known Euler equation

$$\mathbf{T}^B = J^B\boldsymbol{\omega}^B + \boldsymbol{\omega}^B \times J^B\boldsymbol{\omega}^B, \tag{2}$$

where J is the (here diagonal) inertia matrix of the satellite. During the slew the coupling of the axes is compensated by nonlinear feedback control such that

$$\mathbf{T}^B = \mathbf{T}^B_C + \mathbf{T}^B_{max}, \text{ where} \tag{3}$$

$$\mathbf{T}^B_C = \boldsymbol{\omega}^B \times J^B\boldsymbol{\omega}^B. \tag{4}$$

This allows to assess the maximum angular acceleration which can be realized during a slew around an axis i of the B-frame

$$\dot{\omega}^B_{max,i} = \frac{T^B_{max,i}}{J_{ii}}. \tag{5}$$

Slew Feasibility The calculation of the slew maneuver is done in the N-frame to reduce the computational cost. However, the torques have physically to be generated in the body fixed frame. Therefore we use a conservative upper border for the maximum allowable angular acceleration in N-frame

$$\dot{\omega}^N_{max} = \min_i \sqrt{\frac{(\dot{\omega}^B_{max,i})^2}{3}} \tag{6}$$

such that the resulting torques in the body fixed B-frame cannot be exceeded. We use a steplike angular acceleration profile with duration Δt see Figure 6.

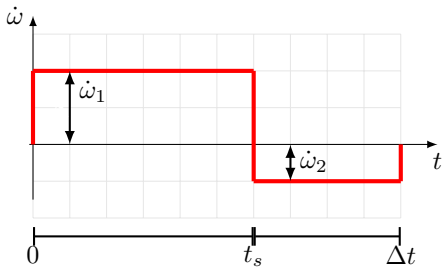


Figure 6: Generic steplike torque profile around one axis of the N-frame

The slew is calculated around each axis separately. Starting with the angular rate $\omega(t_0)^N$ at the beginning of the slew and the initial Euler angle $\alpha(t_0)$ corresponding to $\phi(t_0)$, $\theta(t_0)$ or $\psi(t_0)$ and the desired angular rate $\omega(t_0 + \Delta t)$ and angle $\alpha(t_0 + \Delta t)$, respectively, the necessary change of the states in each axis is given by

$$\Delta\alpha = \alpha(t_0 + \Delta t) - \alpha(t_0)$$
$$\Delta\omega = \omega^N(t_0 + \Delta t) - \omega^N(t_0).$$

Integration yields

$$\Delta\omega = \int_0^{\Delta t} \dot{\omega}^N(t)\, dt = \dot{\omega}_1\, t_s + \dot{\omega}_2\, (\Delta t - t_s), \tag{7}$$

corresponding to Figure 6. We denote t_s as switching time in between both angular accelerations. The change in the angle is

$$\begin{aligned}\Delta\alpha =& \iint_0^{\Delta t} \dot{\omega}^N(t)\, dt\, dt \\ =& \frac{1}{2}\dot{\omega}_1 t_s^2 + \omega(t_0)t_s + \frac{1}{2}\dot{\omega}_2(\Delta t - t_s)^2 \\ &+ (\omega(t_0) + \dot{\omega}_1 t_s)(\Delta t - t_s).\end{aligned} \tag{8}$$

Equation 7 can be converted to

$$\dot{\omega}_2 = \frac{\Delta\omega - \dot{\omega}_1 t_s}{\Delta t - t_s}$$

while Equations 7 and 8 yield

$$t_s = \frac{\Delta\alpha - \frac{1}{2}\Delta\omega\Delta t - \omega_0\Delta t}{\frac{1}{2}\dot{\omega}_1\Delta t - \frac{1}{2}\Delta\omega}. \tag{9}$$

Equation 9 solely depends on the unknown initial angular acceleration $\dot{\omega}_1$. We set

$$\dot{\omega}_1 := \pm\dot{\omega}^N_{max}$$

under the assumption that it is always reasonable to begin a slew with maximum angular acceleration in order to have a maximum scope for the choice of t_s and $\dot{\omega}_2$. The correct sign of $\dot{\omega}_1$ can be found by evaluation of t_s.

$$\dot{\omega}_1 = \begin{cases} \dot{\omega}^N_{max}, & t_s > 0 \\ -\dot{\omega}^N_{max}, & t_s < 0 \end{cases}$$

Equation 9 is evaluated again, if necessary. If t_s or $\dot{\omega}_2$ holds one of the following equations

$$t_s < 0$$
$$t_s > \Delta t$$
$$|\dot{\omega}_2| > \dot{\omega}^N_{max}$$

for any axis of the N-frame, the maneuver is infeasible[1].

Experimental Results

To test the feasibility of our approach, we implemented the precompiled planning problem from Figure 2 modeling the Earth observation scenario from Figure 1 in PDDL and solved it with TFD/M. Figure 7 shows a visualization of the

[1]It is worth to mention that the commanded maneuver in B-frame could be feasible anyhow due to the conservatism induced in Equation 6. This is part of future optimization.

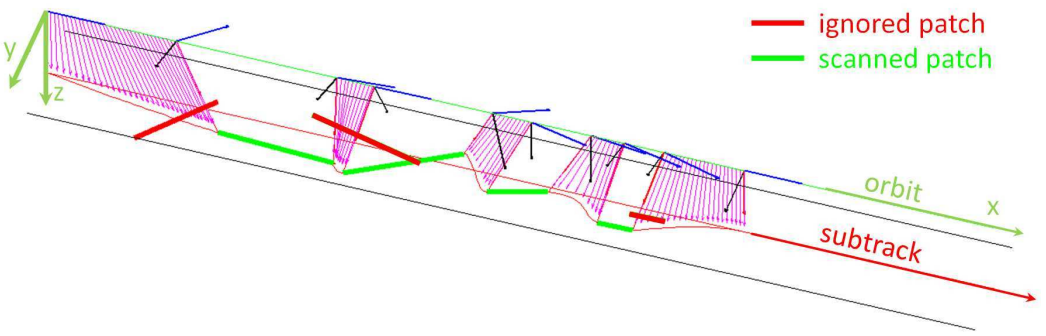


Figure 7: Earth observation scenario with subtrack of the satellite and observation sites to be scanned. The magenta colored arrows depict the line of sight of the instrument during the slew maneuver.

intermediate states of the satellite extracted from the resulting plan. The satellite's attitude is depicted by the current body fixed frame in blue black and magenta.

The satellite slews towards the first patch to scan it at its earliest possible state. The start of the scan maneuver of all other patches is constrained by the attitude of the satellite after scanning the previous patch, so all other maneuvers have to be calculated by interval nesting.

The resulting plan happens to be optimal for the tested planning instance given our assumptions and the satellite parameters used. The leftmost patch cannot be scanned because the satellite's state to start the scan maneuver is infeasible. In additional experiments we investigated the influence of increasing the maximal angular rates of the satellite in the planning problem. In this case the slew maneuver from the last patch to the ignored rightmost patch becomes feasible. With more angular scope also the first patch is in range and TFD/M finds a plan scanning six of the patches.

Although our approach is promising and seems to work well in practice, optimality cannot be guaranteed, even regarding the inaccuracies of the model such as plain Earth surface, circular orbits, etc. The interval nesting approach is only iterated to a certain depth so that each scan action is started at a minimally later orbit position than theoretically possible. Now it is easy to construct a problem that will not be solved optimally by our approach by adding a new patch that is reachable by a slew maneuver from the orbit position after scanning the previous patch from the theoretical earliest orbit position but not from the orbit position found by interval nesting. Completeness can be achieved with the trivial plan, but in scenarios where all patches could be scanned it is not guaranteed that TFD/M will unnecessarily ignore some observation sites with the analogous argument as for optimality.

Conclusion and Future Work

We have presented an agile Earth observation task and an automated planning system capable of solving it. Preliminary experiments show the feasibility of our approach. We will continue our research in more complex problems.

The automated planning system TFD/M can be successfully applied to our Earth observation scenario. Nevertheless we believe that better planning systems could be developed for larger Earth observation tasks. As modeled, the Earth observation problem does not involve temporal concurrency, and the planning problem is even serialized artificially with the help of Boolean "idle" variables. While numerical variables are required for the Earth observation scenario, the temporal aspect of TFD is not. Many other real world problems do not require temporal concurrency but have to deal with complex numerical calculations in the same way as TFD/M handles semantic attachments. A serial numerical planner would hav to solve a problem with a smaller branching factor of sequential actions instead of time stamped states which offers the potential for better heuristics. These heuristics could either be more informed or faster to compute which increases the performance of a planning system and therefore the industrial applicability.

The planning problem can be formalized with the semantic attachments extension of the planning language PDDL. However, it is strongly connected with the syntax in the modules, and the interface between modules (implemented in the programming language C++) and planning language has to be maintained by hand. An object oriented planning language would be better suited to setup the problem. A promising approach that improves the communication between modules and planning problem is the Object Oriented Planning Language OPL (Hertle 2011).

Satellites in an Earth observation mission have only limited storage capacities and have to dump collected data to a ground station from time to time. We intend to model this data handling in future implementations. Further improvements include the handling of the satellite's orbit which was modeled to be circular, while it is somewhat elliptical in reality. Considering the deviations from a circular orbit does not change the general structure of our model and is beneficial to better approximate the satellite's real behavior.

Acknowledgments

This work was supported by the German Aerospace Center (DLR) as part of the Kontiplan project (50 RA 1221).

References

Dornhege, C.; Eyerich, P.; Keller, T.; Trüg, S.; Brenner, M.; and Nebel, B. 2009. Semantic Attachments for Domain-independent Planning Systems. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS 09)*.

Eyerich, P.; Mattmüller, R.; and Röger, G. 2009. Using the Context-enhanced Additive Heuristic for Temporal and Numeric Planning. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS 09)*, 130–137. AAAI Press.

Fox, M.; Long, D.; and Magazzeni, D. 2011. Automatic Construction of Efficient Multiple Battery Usage Policies. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS 2011)*.

Ghallab, M.; Isi, C. K.; Penberthy, S.; Smith, D. E.; Sun, Y.; and Weld, D. 1998. PDDL - The Planning Domain Definition Language. Technical report, CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control.

Helmert, M. 2006. The Fast Downward Planning System. *Journal of Artificial Intelligence Research* 26:191–246.

Hertle, A. 2011. Design and Implementation of an Object-Oriented Planning Language. Master's thesis, Albert-Ludwigs-University, Freiburg.

Keyder, E., and Geffner, H. 2009. Soft Goals can be Compiled Away. *Journal of Artificial Intelligence Research (JAIR)* 36:547–556.

Löhr, J.; Eyerich, P.; Keller, T.; and Nebel, B. 2012. A Planning Based Framework for Controlling Hybrid Systems. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS 2012)*.

Penna, G. D.; Intrigila, B.; Magazzeni, D.; and Mercorio, F. 2010. Planning for autonomous planetary vehicles. In *27th Congress of the International Council of the Aeronautical Sciences (ICAS 2010)*, 131–136.

Operational Planning of Thermal Generators with Factored Markov Decision Process Models

Daniel Nikolaev Nikovski
nikovski@merl.com
Mitsubishi Electric Research Laboratories,
201 Broadway, Cambridge, MA 02139, USA

Abstract

We describe a method for creating conditional plans for controllable thermal power generators operating together with uncontrollable renewable power generators, under significant uncertainty in demand and output. The resulting stochastic sequential decision problem has mixed discrete and continuous state variables and dynamics, and we propose a discretization method for the continuous part of the model that unifies all variables into a large discrete Markov decision process model. Although this model is way too large to be solved directly, its state transition probabilities can be factored efficiently, and a reduction of all continuous variables to one net demand variable makes it tractable by dynamic programming over a suitably constructed AND/OR tree. The proposed algorithm outperformed existing non-stochastic solvers on several problem instances, resulting in both lower risks and operational costs.

Introduction

The operational planning of thermal generators is a difficult sequential optimization problem that electrical power utilities must solve continuously to ensure that they meet power demand with maximal reliability and at a minimal cost. Fossil-burned thermal generators (using coal, natural gas, or oil) consume vast amounts of expensive fuel and contribute significantly to global warming, so minimizing the amount of consumed fuel is of primary importance in the electrical power industry.

Given a set of generators with their cost structure and fuel consumption rates, the objective of optimal operational planning is to find the best sequence of commands to turn individual generators on or off, and the optimal amount of power produced by each of them over an extended period of time, subject to the operational constraints that these generators might have. Typical planning periods range between one day or one week, and the state of the generators can typically be changes once every hour. The predicted demand over the entire planning horizon is also assumed to be known, either exactly, or with some quantifiable uncertainty.

There are several reasons why this problem is very computationally challenging. The first reason lies in the temporal constraints on the operational durations of individual generators that arise from their mechanical construction and requirements for reliable operation. It is generally not desirable (or frequently even possible) to turn the burners of the generators on and off at arbitrary moments, because frequent switching would cause damage due to excessive thermal expansion and contraction. For this reason, once a generator is turned on or off, it must remain in that state for several hours, and conversely, if it has been turned off, it must be kept off for several hours. In other words, when a command is given to turn a generator on or off, it is committed to that state for multiple decision periods, and that is why this problem is also known as the *unit commitment* problem. Due to these temporal constraints, the planning problem must be solved over the entire planning horizon, making it a sequential decision problem. Although the states of the generators are Boolean variables (on or off), the state variables of the sequential decision problem must include information about how many decision periods the generator has been on or off, and that increases the cardinality of the state space enormously.

The second reason for the high computational complexity of this problem is its mixed continuous/discrete nature. Some of the decision variables are discrete (e.g. the commitment status of generation units), and others are continuous (e.g. the amount of power produced by each unit). Moreover, the dynamics that govern the evolution of the system are also mixed: the total demand for electricity is a continuous scalar variable, whereas the main components (generation units) switch between discrete modes (on or off). This significantly limits the number of solution methods that can be applied to this problem, because there are relatively few planning and optimization methods that can solve mixed continuous/discrete problems efficiently.

The third reason this problem is very difficult is that at least part of the system dynamics are random, for most practical situations. In all cases, for any future moment during the planning period, the total demand for power will not be a completely known, deterministic value, but a random variable predicted from the information available at the time of planning. The prediction error can often be quantified (typical values are around 2% of total demand), and although most currently deployed planning systems have chosen to ignore this uncertainty or deal with it in a heuristic manner, such uncertainty information can arguably be used to improve the performance of the planning algorithm. Moreover, the increased penetration of renewable power sources such as photovoltaic panels and wind turbines, whose output

depends strongly on uncontrollable atmospheric conditions such as solar radiation and wind speed, has effectively introduced much higher levels of uncertainty in the net demand for power to the controllable thermal generators. For example, if 20% of the power generator by a utility is supplied by wind turbines, in case the wind dies down suddenly, the net demand to the thermal generators might increase suddenly by 20%. The operational plan must allow for such contingencies, if forced outages are to be avoided. As a result, ignoring uncertainty in the system is becoming increasingly impossible for electrical power utilities. And, in addition, other sources of uncertainty are possible faults in the generators, which are naturally random events, but can be characterized probabilistically.

Due to its primary economic significance, the operational planning problem for thermal generators has been addressed by a very large number of solution methods, including ones based on dynamic programming, Lagrangian relaxation, interior point methods, and mixed integer programming, as well as heuristic methods such as genetic algorithms, simulated annealing, evolutionary programming, differential evolution, particle swarm optimization, Hopfield neural networks, etc. (Wood and Wollenberg 1996; Xia and Elaiw 2010). Formulations as a model predictive control or optimal control problem are possible, too (Xia, Zhang, and Elaiw 2011). Dynamic programming methods can leverage successfully the sequential nature of the decision process in order to compute suitable plans efficiently, but suffer from the well known curse of dimensionality due to the large size of the state space of the problem. Mixed integer programming methods can handle successfully the mixed continuous/discrete nature of the planning task, but again do not scale up very well because of the sheer combinatorial complexity of the discrete optimization part. Lagrangian relaxation could also be a very effective solution to the mixed continuous/discrete optimization problem, and has been shown to perform well on large problems. Global optimization methods such as genetic algorithms and simulated annealing can be very effective on problems with disjoint feasibility regions, but cannot guarantee that global optima would always be reached, in general.

However, the majority of these methods either ignore uncertainty completely at the planning state, or deal with it heuristically, or consider only a small number of possible future realizations of the uncertain variables, and usually compute a fixed operational plan for the entire period that is executed sequentially. As is well known in AI, such plans can only succeed if the problem domain is static, completely observable, deterministic, and the action descriptions available to the planner are correct and complete. One common heuristic is to include a safety margin of extra capacity (for example, 3%) to be committed for production. This results in operating more and/or larger units than are necessary to meet expected demand. This approach is largely heuristic, and is not likely to work in the future, when renewable energy sources become even more widespread. And, in general, whereas there might be some value in algorithms that can find fixed plans that are maximally robust with respect to future uncertain outcomes, a much more natural approach would be to use algorithms that can compute conditional plans that can select actions depending on future states (also known as contingency plans in AI, feedback controllers in control theory, and decision policies in operations research). This paper describes one such approach based on factored Markov decision processes (fMDP), where continuous dynamics are discretized by means of a barycentric approximation and added to the discrete dynamics, the state of the resulting completely discrete fMDP is pruned by means of problem domain knowledge, and the optimal decision policy is found by means of dynamic programming over AND/OR trees.

Formulation of the Planning Problem

Formally, the operational planning problem for generators can be described as follows. Given N available controllable generator units, and a planning horizon of length T units of suitable duration, for example one hour, the overall goal is to minimize the total operating cost for these units, subject to operating constraints and at an acceptable risk of a forced outage. The demands D_t, $1 \leq t \leq T$, over the entire planning horizon are random variables coming from a stochastic process with known structure and parameters. There are also K uncontrollable generators, and we assume that the realizations y_t^k of their random output amounts Y_t^k, $1 \leq t \leq T$, $1 \leq k \leq K$, also come from known stochastic processes. At all times, the sum of the supply from all generators, controllable and uncontrollable, must match the total demand at that time.

In order to formulate a sequential decision problem, we introduce the decision variables $u_t^i \in \{0, 1\}$ for all time periods t, $1 \leq t \leq T$, and controllable units i, $1 \leq k \leq K$, which represent the intended commitment status of the generators during the next operational period. Similarly, we introduce the state variables $x_t^i \in \{-l, -l+1, \ldots, -1, 1, \ldots, L-1, L\}$, where l is the minimum allowed time for keeping a generator off, and L is the minimum allowed time for keeping a generator on. Negative values correspond to off condition, and positive values correspond to on condition.

For the state variables of the controllable part of the process, if we have an existing commitment status u_{t-1}^i for generator i, operation time x_{t-1}^i, and new commitment status u_t^i, the new operational time x_t^i can calculated by Equation (1) where T_i^{cl} is the "cold start" time of unit i, l_i is the minimum down time of unit i, and L_i is the minimum up time of unit i (Li, Johnson, and Svoboda 1997).

$$x_t^i = \begin{cases} 1 & \text{if } -T_t^{cl} \le x_{t-1}^i \le -l_i \text{ and } u_t^i = 1 \text{ (start up)} \\ x_{t-1}^i + 1 & \text{if } 1 \le x_{t-1}^i \le L_i - 1 \text{ (up and must stay up)} \\ L_i & \text{if } x_{t-1}^i = L_i \text{ and } u_t^i = 1 \text{ (up and available to shut down)} \\ -1 & \text{if } x_{t-1}^i = L_i \text{ and } u_t^i = 0 \text{ (shutting down)} \\ x_{t-1}^i - 1 & \text{if } -l + 1 \le x_{t-1}^i \le -1 \text{ (down and must stay down) or } -T_i^{cl} + 1 \le x_{t-1}^i \le -l_i \text{ and } u_t^i = 0 \text{ (down and available to start up)} \\ -T_i^{cl} & \text{if } x_{t-1}^i = -T_i^{cl} \text{ and } u_t^i = 0 \end{cases} \quad (1)$$

Additional constraints, such as maximal up/down times, can be accommodated by suitable modifications to Eq. (1).

For the demand variable, we assume that we have a stochastic dynamic model that specifies the probability $Pr(D_t = d_t | D_{t-1} = d_{t-1}, D_{t-2} = d_{t-2}, \ldots, D_0 = d_0)$ that value (power demand) d_t will be observed at time t if a series of demands $d_0, d_1, \ldots, d_{t-2}, d_{t-1}$ has been observed until then. Similarly, for each uncontrollable generator k we assume that we can estimate the probability $Pr(Y_t^k = y_t^k | Y_{t-1}^k = y_{t-1}^k, Y_{t-2}^k = y_{t-2}^k, \ldots, Y_0^k = y_0^k)$ that value (power output) y_t^k will be observed at time t if a series of outputs $y_0^k, y_1^k, \ldots, y_{t-2}^k, y_{t-1}^k$ has been observed until then. Various predictive models can be used, such as auto-regressive (AR), neural nets, support vector machines, etc., that map past observations onto future values.

The planner must observe several constraints in minimizing the total cost. The load balance constraint states that the total generation must be equal to the demand d_t at any time step. If p_t^i is the generation of unit i at hour t, then

$$\sum_{i=1}^{N} p_t^i u_t^i + \sum_{k=1}^{K} y_t^k - d_t = 0, \text{ for } \quad t = 1, 2, \ldots, T. \quad (2)$$

The objective function is presented in Equation 3, where $\mathbf{E}_{u_0, x_0, y_0, d_0}$ denotes the expectation operator with regard to the initial configuration u_0, operational time x_0, the initial demand d_0, and the initial output y_0. For notational simplicity, the decision variables at time t are represented as the vector $u_t \doteq [u_t^1, u_t^2, \ldots, u_t^N]$, the state variables are denoted by the vector $x_t \doteq [x_t^1, x_t^2, \ldots, x_t^N]$, and the realizations of all uncontrollable generators are denoted as $y_t \doteq [y_t^1, y_t^2, \ldots, y_t^K]$.

$$\begin{aligned} J^* &= \min_{u_1, u_2, \ldots, u_T} \mathbf{E}_{u_0, x_0, y_t, d_t} \{ \textstyle\sum_{t=0}^{T-1} [\sum_{i=1}^{N} f_i(x_t^i, u_t^i, y_t, d_t) \\ &+ \textstyle\sum_{i=1}^{N} h_i(x_t^i, u_t^i, u_{t+1}^i) + g_t(u_t, y_t, d_t)] \} \end{aligned} \quad (3)$$

Here $f_i(x_t^i, u_t^i, y_t, d_t)$ denotes the operating cost of operating unit i in configuration u_t^i and state x_t^i for one time step in order to meet demand d_t when the uncontrollable generators output electricity amount y_t. The function $h_i(x_t^i, u_t^i, u_{t+1}^i)$ denotes the cost of switching to configuration u_{t+1}^i at the end of the step. The third cost component, $g_t(u_t, y_t, d_t)$, denotes the equivalent cost of the risk of not being able to meet demand d_t under output of uncontrollable generators y_t with the chosen configuration of all units u_t. This cost is proportional to the probability that the total capacity of the committed units in u_t plus what the uncontrollable generators produce (y_t) is less than the demand d_t:

$$g_t(u_t, y_t, d_t) = \alpha Pr(\sum_{i=1}^{N} u_t^i cap^i + \sum_{k=1}^{K} y_t^k < d_t),$$

where cap^i is the maximal generation capacity of unit i. A suitably chosen proportionality coefficient α specifies the relative preference between minimizing operating cost and risk of failure to meet demand. By adding the operating cost and risk compensation cost together, the objective function represents a trade-off between fuel costs and risk.

At any given time, if we can find the optimal sequence $u_1, u_2, \ldots, u_T$ that minimizes the cost in Equation 3 by whatever computational means, we will have an operational plan that can be executed over the entire planning horizon. However, as argued above, such an open-loop, unconditional plan is not tailored to the concrete situation that will be encountered in the future. An alternative approach is to recognize that the uncertainty in power demand and generator supply makes the decision problem a stochastic one, and its optimal solution is not an unconditional plan (sequence of commitment decisions), but an entire decision policy. A conditional operational planner could compute conditional plans that are robust to future variations of supply and demand, and could provide a safety margin implicitly, by planning for all possible contingencies. One significant difficulty associated with this approach has been how to represent all such possible contingencies, and how to plan for them. One proposal organizes all future possible realizations of the system (called scenarios) as a tree of scenario bundles (Takriti, Birge, and Long 1996). However, this model for representing stochasticity is limited to only the few scenarios included in it, whereas in a practical system the future evolution can be realized in an infinite number of ways. Our work aims to expand this approach by improving the probabilistic modeling of system evolution.

We propose a method for finding the optimal conditional operational plan of a set of power generators under stochastic demand for electrical power and stochastic output of some generators. Unlike traditional operational plans, which are fixed in advance, a conditional operational plan depends on the future state of the observable random variables (demand and output), and can result in different actual sequences of decisions depending on the observed outcomes for these variables. The planner explicitly balances the operational cost of electricity generation with the risk of not being able to meet future electricity demand. We represent the stochastic dynamics of the components of the system as a factored Markov decision process (MDP) model, and propose efficient approximate algorithms for computing suitable conditional operational plans.

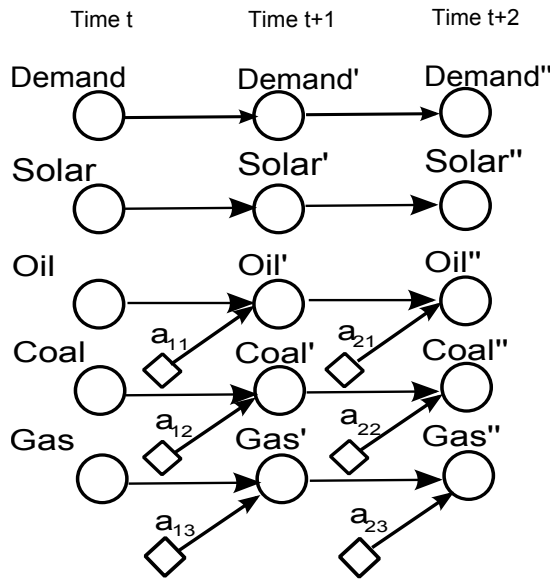


Figure 1: DBN for a power generation problem with three controllable and one uncontrollable power generators.

Factored Markov Decision Processes for Conditional Operational Planning

We propose to represent a power generation system consisting of multiple generators of the type described above by means of a *factored Markov decision process* (fMDP), and find the optimal conditional operational plan by means of approximate dynamic programming (Boutilier, Dearden, and Goldszmidt 2000). The fMDP is usually expressed graphically as a *dynamic Bayesian network* (DBN). A DBN consists of circles that represent random variables, diamonds that represent decision variables, and directed edges connecting the circles and diamonds that represent the statistical dependence between the corresponding variables. When dealing with a time-dependent system, each time period (e.g. one hour) is represented by its own set of random variables. Three time slices of the DBN for an example stochastic unit commitment problem with four generators, one of which uncontrollable (solar), are shown in Fig. 1.

In Fig. 1, one random, continuous, and uncontrollable variable represents the power demand. Another random, continuous, and uncontrollable variable represents the output of a photovoltaic generator. (In this case, these two model components are first-order Markovian, that is, the next state depends only on the current state, for example by means of an AT(1) model. However, this is not a fundamental limitation: for higher-order models, edges from previous time slices can be added, too.) In addition, three conventional controllable generators are shown, too; their discrete variables x_t^i take on $l + L$ possible different values, and represent the operational time of the respective generator. Three decision variables (shown as diamonds) represent the individual decisions $a_{ti} = u_t^i$ to turn on/off the corresponding generators, and thus commit them for power production. These models components are necessarily first-order Markovian, but do not need to be deterministic — certain probability of failure to change the state of a generator as desired could be modeled in them. The probabilistic evolution of the system is described by local conditional probability tables for each variable, where the conditional dependence is defined only on the parents of that variable in the graph of the DBN. Thus, the DBN serves as a compact representation of a large Markov decision process whose state space is exponentially large in the number of states of the individual variables over which it is factored.

In order to specify a factored MDP, the state, action, and transition model for each individual variable must be defined, along with the reward/cost structure. This is done differently for the thermal generators which are naturally represented by means of discrete variables, and for the demand and uncontrollable generators which are naturally represented by means of continuous variables. For the fMDP part corresponding to thermal generators, the definitions of state and action variables coincide with those in the original sequential decision problem described in Section . For the continuous variables, we use a discretization method based on barycentric coordinates that we have already applied to other sequential decision problems such as train run-curve optimization and set-point scheduling for air conditioners (Nikovski and Esenther 2011; Nikovski et al. 2012; Nikovski, Xu, and Nonaka 2013).

The main idea of the method is to replace the continuous state variables with a discrete set of states in a way that approximates well the original continuous dynamics. Let the dynamics of a continuous component of the model be represented by the function $z_{t+1} = f_z(z_t, a_t)$, where z_t is a vector variable that could include one or more of the demand d_t, the output of uncontrollable generators y_t^k, or some of their time-lagged values d_{t-1}, y_{t-1}^k, etc. Let the dimensionality of this vector be b. The objective of the conversion method is to represent the dynamics of the continuous system $z_{t+1} = f_z(z_t, a_t)$ by a conditional probability transfer function $Pr(s_{t+1} = s^{(j)} | s_t = s^{(i)}, a_t = a^{(k)})$, defined over suitably chosen set S of N discrete states $s^{(i)}$, $1 \leq k \leq K$. The algorithm selects N states $s^{(1)}$, $s^{(2)}$, ..., $s^{(N)}$ such that each corresponds to a state $z^{(i)} \in R^b$, and their Delaunay triangulation is computed (Fig. 2), (Preparata and Shamos 1990). Then, each available action $a^{(l)}$ is executed in each of them in turn, according the continuous dynamics function $z' = f_z(z^{(i)}, a^{(l)})$, and the barycentric coordinates $p_1, p_2, \ldots, p_{b+1}$ of the end state z' are computed with respect to the simplex that encloses it. These barycentric coordinates are then used as transition probabilities of the discrete MDP. The detailed computational procedure, along with discussion of its computational complexity, is available in (Nikovski and Esenther 2011).

Conceptually, we can think of this algorithm as a way of converting the system dynamics represented by the function f_z to an equivalent probabilistic representation involving only a small set of points $s^{(i)}$embedded into the original continuous state space of the system. If the system starts in one of these few points, the successor state z', in general, will

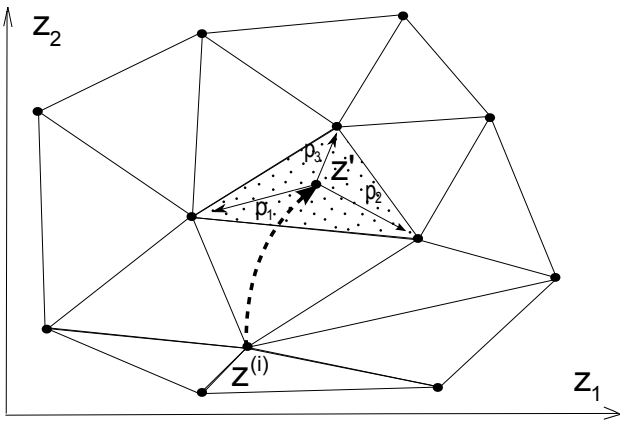


Figure 2: A Delaunay triangulation on a set of vertices sampled from the embedding two dimensional space. The dashed line shows the transition from some starting state $z^{(i)}$ under action a resulting in end state $z' = f(z^{(i)}, a)$. The simplex (here, triangle) containing the end state z' is shown with a dotted background, and the barycentric coordinates p_1, p_2, and p_3 of z' are computed with respect to the vertices of that simplex. These coordinates are also the transition probabilities from $z^{(i)}$ under action a to the states corresponding to these vertices in the resulting MDP.

not coincide with another one of these points. However, we can identify the $b + 1$ points that define a simplex that completely encloses the successor state z', and can think that the system has transitioned not to point z' itself, but to the vertices of this simplex with various probabilities, instead. The probabilities are equal to the convex decomposition of point z' with respect to the vertices of the simplex, also known as the barycentric coordinates of that point within the simplex. The similarities between convex combinations (barycentric coordinates) and probability mass functions required by the MDP formalism make this conversion possible.

This procedure is applied in turn for every group of variables in the DBN that have temporal dependence. For the demand variable D, we could assume that the next demand D_{t+1} depends only on the current demand D_t (Markovian property of the underlying stochastic process) with transition probability $Pr(D_{t+1} = d_{t+1}|D_t = d_t)$, and if a higher-order model is necessary, time-lagged values of demand D_{t-1}, D_{t-2}, etc. could be included. For the uncontrollable generators, we make similar assumptions that Y^k_{t+1} depends only on Y^k_t, with probability $Pr(Y^k_{t+1} = y^k_{t+1}|Y^k_t = y^k_t)$. These transition probabilities can be estimated either from statistical data, or by means of discretizing a suitable continuous stochastic Markov process, such as the auto-regressive process of order 1 (AR(1) process).

Once the transition probabilities for all variables in the DBN have been determined, joint transition probability for the entire system $Pr(u_{t+1}, x_{t+1}, y_{t+1}, d_{t+1}|u_t, x_t, y_t, d_t)$ can be computed from the transition probabilities of the individual random variables, as is customary for Bayesian networks. It can be observed that although the MDP has a very large joint state space, its transition structure is very sparse.

The next step is to determine the transition cost, which, unlike transition probabilities that can be specified separately for each individual variable, must be specified for the entire MDP. Given a joint MDP state (u_t, x_t, y_t, d_t) and an action u_{t+1}, the immediate one-step cost $c(u_t, x_t, u_{t+1}, y_t, d_t)$ is computed as

$$\begin{array}{rcl} c(u_t, x_t, u_{t+1}, y_t, d_t) & = & \sum_{i=1}^{N} f_i(x^i_t, u^i_t, y_t, d_t) \\ +\sum_{i=1}^{N} h_i(x^i_t, u^i_t, u^i_{t+1}) & + & g_t(u_t, y_t, d_t) \end{array} \quad (4)$$

where the switching costs $h_i(x^i_t, u^i_t, u^i_{t+1})$ and risk cost $g_t(u_t, y_t, d_t)$ are computed as described above, and the fuel costs $f_i(x^i_t, u^i_t, y_t, d_t)$ are computed by solving the following economic dispatch problem: minimize $\sum_i F_i(p^i_t)$ subject to the generation limits for all generators and the load balance constraint for this particular realization of the uncontrollable variables y_t and demand d_t:

$$\sum_{i=1}^{N} u^i_t p^i_t + \sum_{k=1}^{K} y^k_t - d_t = 0$$

where $F_i(p^i_t)$ is the cost of producing p^i_t units of electricity by generator i; typically, this function is quadratic in p^i_t, and the economic dispatch problem can be solved by means of quadratic programming. The objective of economic dispatch is to find the optimal generation amounts p^i_t of the committed units so that the cost of generation is minimized for a specific realization of the random variables. After the optimal generation amounts $[p^1_t, p^2_t, \ldots, p^N_t]$ are found, the individual generation costs can be calculated as $f_i(x^i_t, u^i_t, y_t, d_t) = F_i(p^i_t)$, $1 \le i \le N$.

Given such an MDP, we can define its cost-to-go functions J_t for each step t and each joint state of the MDP. For the terminal step T, when no further decisions will be made, $J_T(u_T, x_T, y_T, d_T) = 0$.

For all other steps, the cost-to-go function $J_t(u_t, x_t, y_t, d_t)$ is defined iteratively by means of a Bellman equation, as follows (Puterman 1994):

$$\begin{array}{l} J_t(u_t, x_t, y_t, d_t) = \min_{u_{t+1}}\{c(u_t, x_t, u_{t+1}, y_t, d_t) \\ +\sum_{d_{t+1}, y_{t+1}} Pr(d_{t+1}, y_{t+1}|d_t, y_t) J_{t+1}(u_{t+1}, x_{t+1}, y_{t+1}, d_{t+1})\} \end{array} \quad (5)$$

Note that the transition probabilities $Pr(d_{t+1}, y_{t+1}|d_t, y_t)$ are factored conveniently, due to the conditional independence relations in the DBN of the MDP. The cost-to-go function $J_0(u_0, x_0, y_0, d_0)$ of the initial state of the generators and demand would then correspond to the minimal operating cost under the optimal policy for the entire planning problem.

In principle, this cost can be found by computing the costs-to-go of all states in the MDP. However, when some of the variables are continuous, the cost-to-go (value function) of the MDP cannot be computed and represented efficiently. The discretization method described above addresses precisely this problem, by replacing the continuous variables D and Y_k with sets of discrete states S, making the entire

MDP discrete, and standard MDP solution methods such as dynamic programming, value iteration, and policy iteration can be applied (Puterman 1994). A solution method based on dynamic programming over AND/OR trees is described in the next section.

Furthermore, if these costs are computed and stored, the optimal decision $u_{t+1} = \pi_t(u_t, x_t, y_t, d_t)$ for time step t and state (u_t, x_t, y_t, d_t) can be identified as the one that minimizes the right-hand side of the Bellman equation 5:

$$\pi_t(u_t, x_t, y_t, d_t) = argmin_{u_{t+1}}\{c(u_t, x_t, u_{t+1}, y_t, d_t) + \sum_{d_{t+1}, y_{t+1}} Pr(d_{t+1}, y_{t+1}|d_t, y_t) J_{t+1}(u_{t+1}, x_{t+1}, y_{t+1}, d_{t+1})\} \quad (6)$$

This policy is conditioned upon the current realizations of the random variables y_t and d_t, so it represents a conditional planner. By observing the outcomes y_t and d_t for each consecutive time step, different actual operating schedules will be obtained.

Solving fMDP Models with Aggregated Net Demand

The objective of solving the stochastic unit commitment problem represented by the fMDP is to find the optimal policy that maps the states of the fMDP onto the decision variables that signify which generators will be turned on/off in the next period, where optimality is defined in terms of jointly minimizing production cost and risk of failure. The straightforward method of solving fMDPs is to expand the factored state and solve the resulting flat MDP by means of dynamic programming, applying equation 5 repeatedly, starting from the terminal step and proceeding backwards to the first step (Puterman 1994). However, for most practical problems, e.g. when $L = l = 5$, the number of generators $N = 20$, the number of one-hour time periods $T = 24$, the expanded MDP will have $|X| = T(L+l)^N = 24 \cdot 10^{20}$ distinct states for the controllable generators only, and would be impossible to solve.

One practical simplification of the problem is to aggregate the output of the uncontrollable generators Y_t into the demand variable, by subtracting these outputs from the total demand D_t to arrive at the net demand D'_t. If all uncontrollable random variables are Gaussian processes, then D'_t is a Gaussian process, too, with expected value (mean) $\bar{D}'_t$ and variance σ_t for each time period t. Henceforth, we will assume that D_t denotes the net demand. For planning purposes, the net demand D_t can be computed by subtracting the expected values $\bar{Y}_t$ at the time of planning ($t = 0$). When executing the policy, the actually observed realizations y_t at time t can be used to estimate the distribution of the random variable D_{t+1}, so that the estimates of the transition probabilities $Pr(d_{t+1}|d_t, y_t)$ will in fact be based on y_t, when determining the optimal configuration u_{t+1} by means of Equation 6.

Another computational simplification of the problem is to reduce the size of the MDP in a reasonable manner. Intuitively, if forecasts for the values of the continuous random variables D_t and Y_t are known in advance, and the assumption that these are Gaussian processes holds true, most of the configurations of the generators u_t at time t would be irrelevant to satisfying demand at that time. Some of them will have capacities too low to meet demand, and others will use unnecessarily many generators to meet demand economically. By considering only configurations u_t of the controllable part of the MDP whose maximal committed capacity (MCC) is close to the expected net demand $\bar{D}_t$, we can drastically reduce the size of the space of the MDP.

A practical way of identifying such suitable configurations is to run a fast deterministic algorithm for unit commitment for several possible values of target reserve β such that the target demand is $(1 + \beta)\bar{D}$. Suitable schedules S_β are identified for each β, and the generator configurations u_t present in S_β are included in the reduced state space of an approximate solver, which essentially switches between individual segments from multiple schedules S_β, depending on the time evolution of power demand and uncontrollable generators.

Hence, the fundamental idea of the solution algorithm is to identify suitable configurations for representative demands, and then use them to produce schedules for any possible realization of demand. We use an AND/OR tree (Martelli and Montanari 1973) to represent all selected configurations of the generators and possible realizations of future demand. The AND/OR tree is then used for planning for any demand instances.

Generating Candidate Schedules

This step identifies a finite set of representative commitment schedules so that they can be reused in the remaining steps. To solicit schedules, we first select a set of demand samples in hope that they are representative ones. For each slected demand, a deterministic UC problem associated with the demand is solved to obtain its schedule. We start identifying demand samples by finding the overall "upper" and "lower" demands of interest. Let the mean of the demand D be $\bar{D} = [\bar{D}_1, \bar{D}_1, \ldots, \bar{D}_T]$. Starting from a large positive number β and decreasing it gradually, we find a demand $(1 + \beta)\bar{D}$ whose UC is feasible. This demand is the upper demand. Similar procedure can find the lower demand. The two demands determine a demand interval. The schedule generation procedure performs search in the interval and finds demands and their solution schedules.

Building the AND/OR tree

An AND/OR tree has two types of nodes — AND nodes and OR nodes. An AND/OR tree is a tree where (1) its root is an AND node, (2) it has alternating levels of AND and OR nodes, and (3) its terminal nodes are AND nodes (Martelli and Montanari 1973). An AND/OR tree is shown in Fig. where the AND/OR nodes are respectively in rectangular/circular shapes. Note that in this case the outputs of the uncontrollable generators Y_t have been aggregated into the net demand variable D_t, and are not included in the AND/OR tree.

An AND node for the UC problem is associated with a system state (u_t, x_t, d_t) at time step t, whereas an OR node is associated with the action u_t at that time. The root node corresponds to the initial state of the UC system. The values

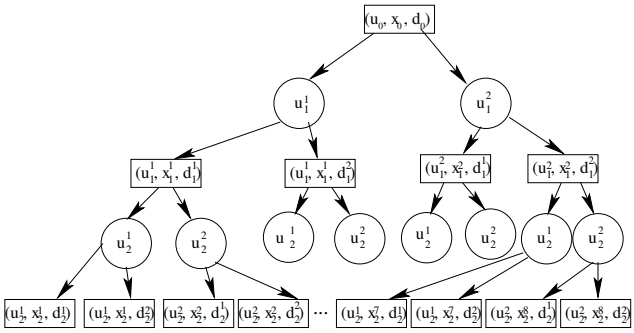


Figure 3: An AND/OR tree example

of the nodes are evaluated bottom-up. For an OR node u_{t+1}, if its parent AND node is (u_t, x_t, d_t) and its children (AND) nodes are $\{(u_{t+1}, x_{t+1}, d_{t+1})|d_{t+1}\}$, then the value of the OR node is evaluated as

$$\begin{aligned} V_t(u_{t+1}|u_t, x_t, d_t) &= c(u_t, x_t, u_{t+1}, d_t) \\ &+ \textstyle\sum_{d_{t+1}} p(d_{t+1}|d_t) V_{t+1}(u_{t+1}, x_{t+1}, d_{t+1}) \end{aligned} \tag{7}$$

Note that the notation $V_t(u_{t+1}|u_t, x_t, d_t)$ means that the value of OR node is conditional on its parent AND node. For an AND node (u_t, x_t, d_t), its value $V_t(u_t, x_t, d_t)$ is evaluated as follows:

$$V_t(u_t, x_t, d_t) = \begin{cases} c(u_T, x_T, u_T, d_T), & \text{if } t = T \\ \min_{u_{t+1}} V_t(u_{t+1}|u_t, x_t, d_t) & \text{otherwise} \end{cases} \tag{8}$$

Note that the minimization in $\min_{u_t}$ is over all children OR nodes u_{t+1}, and that no configuration switching cost is incurred at the last step, since the continuation of the schedule at that time is yet unknown.

Evaluating MDP Policies

Once a policy has been computed and stored in the AND/OR tree, we adopt a sampling approach to evaluate its operational cost and risk under future random demand D. For this purpose, we draw a suitable number of samples $d = [d_1, d_2, \ldots, d_T]$ from the demand variable D (e.g., 1000 samples). For each sample, we start from the root of the tree and execute the actions specified by the tree. Such an execution results in a path in the tree. Specifically, an execution path is a sequence of system states and actions $\{(u_0, x_0, d_0), u_1, (u_1, x_1, d_1), \ldots, u_T, (u_T, x_T, d_T)\}$ that are prescribed by the initial system state, the AND/OR tree, and the demand realization $d = [d_1, d_2, \ldots, d_T]$. The cost of a path can be accessed by solving the economic dispatch problem for each step, given the prescribed configurations u_t, while its risk can be calculated using the committed capacity u_t and the realization of demand d_t. The overall risks and costs are the average across the paths associated with the demand samples. These costs and risks show how the risks can be compromised by the additionally paid cost.

Experimental Results

We experimented with the proposed method on a test problem adopted from (Li, Johnson, and Svoboda 1997), extended with the introduction of uncertainty in the demand. The standard deviation of demand was assumed to be 2% of expected demand: $\sigma_t = 0.02\bar{D}_t$. No uncontrollable generators were used, so the net demand is equal to the total demand. The approximate algorithm from the previous section was implemented and compared against two existing algorithms: one of them was based on a priority list ((Wood and Wollenberg 1996)), and the other one was the decommitment algorithm proposed in (Li, Johnson, and Svoboda 1997). Our results showed that the approximate solution method provides a good balance between generation cost and risk of failure to meet demand. We performed experiments on two UC examples: one with 4 units, and another one with 20 units. We were able to calculate the truly optimal MDP solution for the 4-unit UC example, so we were able to investigate the accuracy of our approximation scheme on that problem, too. The experiments were performed on a computer with Intel Core 2 Duo E6600 CPU (2.40GHz). The algorithm was implemented in MATLAB.

Experimental Conditions

The generation cost of a committed unit i at time t is computed as a quadratic function of the produced amount of power by the unit: $f_i(x_t^i, u_t^i, d_t) = c_0^i + c_1^i p_i^t + c_2^i (p_t^i)^2$. The unit switching and start-up cost is expressed as $h(x_t^i, u_t^i, u_{t+1}^i) = tcst_i + bcst_i(1 - \exp(-\gamma x_t^i))$, if $u_t^i = 0$ and $u_{t+1}^i = 1$, and zero otherwise. In the start-up cost, the fixed component $tcst_i$ represents the cost of starting generator i, while the second term $bcst_i$ represents the cost of starting the boiler and varies exponentially with the length of the time that the unit has been off.

Under a Gaussian assumption for demand ($D_t \sim N(\bar{D}_t, \sigma_t^2)$), the risk compensation cost $g_t(u_t, d_t)$ is given by

$$\alpha' \cdot C_{FSO} \cdot \int_{\sum_i u_t^i cap_i}^{\infty} \frac{1}{\sqrt{2\pi\sigma_t^2}} exp(-\frac{(D - \bar{D}_t)^2}{2\sigma_t^2}) \cdot dD$$

where α' is the proportionality constant, C_{FSO} is the full system operating costs (the cost of the system in which all units are turned on and generate according to their maximum capacity), and the integral is the failure probability (risk). Failure happens when the actual demand D is greater than the Maximum Committed Capacity (MCC) $\sum_i cap_i u_t^i$ of all operating units. By increasing the constant α, the weight of the risk component in the objective function is increased, thus favoring configurations with higher MCC, at the expense of a higher operational cost for running such configurations.

A 4-unit example

The decision horizon of the 4-unit UC problem was 24 hours. The coefficients $tcst_i$ and $bcst_i$ of the start-up costs for the four units were [200,2000;500,20000;100,700;44,100]. The

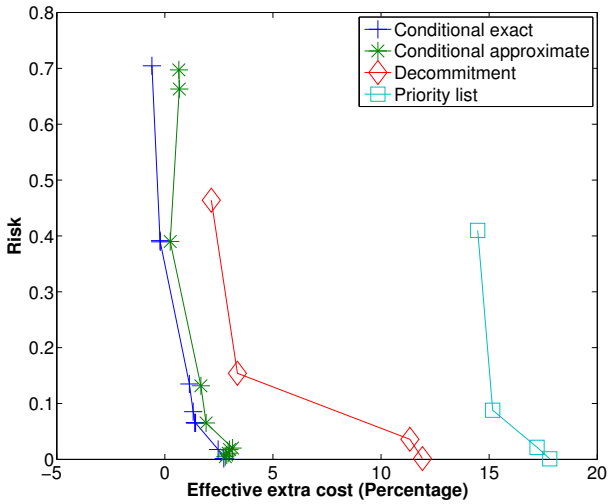


Figure 4: Performances of the algorithms on a 4-unit problem

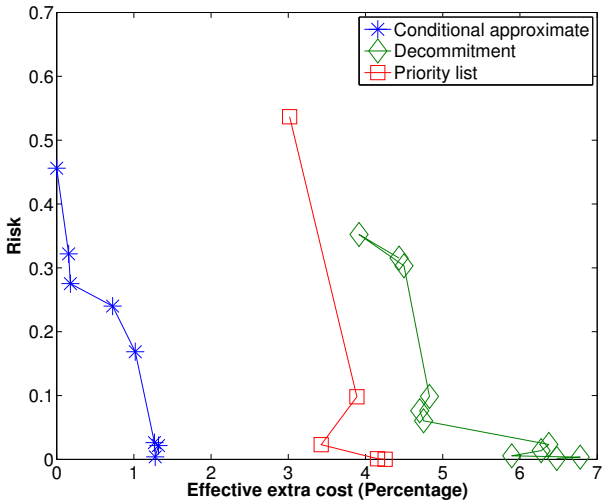


Figure 5: Performances of the algorithms on a 20-unit problem

fuel cost coefficients $[c_0, c_1, c_2]$ for the four units were [0.00211,16.51,02.7; 0.00063,21.05,1313.6; 0.00712,22.26,371.0; 0.00413,25.92,660.8], in chosen cost units. The minimum up and minimum down times were $[3, 3, 2, 2]$ and $[4, 4, 3, 3]$. The minimum and maximum capacities were $[10, 10, 10, 10]$ and $[100, 90, 80, 60]$, here and henceforward, in chosen power units. The expected demand vector was $\bar{D}$ =[105,85,65,140,100,105,125,145,165,185,205,245,265,285, 200,140,100,105,125,145,165,185,205,225]. The initial operational times were $x_0 = [5, -5, 5, -5]$.

The risk versus cost curves for various methods are presented in Fig. 4. "Conditional exact" refers to the algorithm that solves the MDP exactly, i.e., all Bellman backups (Equation 6) were performed. "Conditional approximate" refers to the algorithm proposed in the previous section. In the figure, the horizontal axis is the percentage of the extra operational cost with respect to a reference operational cost, taken to be the lowest experimentally obtained operational cost for any scheduler on this problem. For this problem instance, it can be seen that the solution of the proposed algorithm is very close to optimality (the conditional exact solution), and the algorithm outperforms significantly both the priority list and the decommitment algorithms in balancing operational costs and risks. For the lowest levels of risk, which are probably close to the desired cost/risk trade-off point of an actual generation system, the loss of optimality is less than 1%, whereas the gain in costs with respect to deterministic schedulers is greater than 9%.

20-unit example

In this experiment we used all 20 generators described in (Li, Johnson, and Svoboda 1997). The expected demand vector was [2133.3, 2133.3, 2066.7, 2066.7, 2133.3, 2133.3, 2266.7, 2400.0, 2400.0, 2400.0, 2333.3, 2200.0, 2133.3, 2133.3 ,2200.0, 2266.7, 2400.0, 2400.0, 2400.0, 2400.0, 2333.3, 2200.0, 2200.0, 2066.7]. It was no longer possible to find the truly optimal conditional schedules, but it is possible to compare the performance of the conditional approximate, priority list, and decommitment algorithms (Fig. 5). Again, the results show that the proposed novel algorithm uniformly achieved a much better risk/cost balance than the priority list and the decommitment approaches, with operational cost savings around 4% for the lowest levels of risk.

Conclusion and Future Work

We have described a general method for representing the mixed continuous/discrete dynamics of power generation systems under multiple sources of uncertainty such as variable power demand and intermittent renewable energy sources, and have introduced a class of conditional operational plans where the unit commitment decisions are conditioned upon the state of observable random variables. The proposed factored Markov decision process models represented in the form of dynamic Bayesian networks are compact and are also easy to specify, maintain, and extend with new power sources. We have also proposed one concrete algorithm for finding such conditional operational schedules for power generation that depend on a single random variable — the net demand that aggregates in itself all sources of randomness. The algorithm focuses on small subsets of all possible configurations of generators in order to compute the schedule efficiently. Experimental results suggest that the resulting conditional plans are close to the truly optimal ones, and provide a much better trade-off between generation cost and risk of failure to meet demand than two known non-stochastic unit commitment algorithms that compute fixed schedules.

In the proposed solution algorithm, we use AND/OR trees to represent, find, and evaluate the optimal conditional plan.

However, this algorithm is by no means the only possible way to solve stochastic generation problems represented by means of fMDPs and DBNs. In future work, we plan to investigate other solution methods based on approximate dynamic programming that could result in much better computational complexity. Furthermore, the current solution aggregates the variability of all stochastic variables into the net demand to the controllable power generators, for the sake of computational efficiency. This simplifies the planning problem, because the branching in the AND-OR tree is based only on that single variable. However, even higher efficiency might be possible if the conditional schedule is conditioned on the values of each individual stochastic component. This would significantly increase the complexity of the planning process, and would depend critically on finding more computationally efficient solution methods for the undrelying fMDP models.

For example, the method proposed in (Feng et al. 2004) represents the value function of the dynamic programming problem over continuous domains by adaptively discretizing such continuous variables. This approach might result in more accurate and compact representations than are possible with our method, where the tesselation of the continuous domains is performed apriori, before value functions are evaluated. Adaptive discretization is indeed compatible with our discretization scheme, too, for example by sub-dividing a simplex where the value function varies a lot (measured on its vertices), into multiple smaller simplices. The application of symbolic dynamic programming (SDP, (Sanner, Delgado, and de Barros 2011)) to the factored MDP-based formulation of the operational planning problem might be possible, too.

The formulation of the fMDP described in the paper assumes that all generators assume their intended configuration u_i^t without fail. This allows us to use the decision variables u_i^t as components of the state of the system, thus simplifying the planning process. If the possibility of equipment failure must be taken into account, the actual configuration U_i^t of the generators should be included as a random state variable in the DBN, and its probabilistic dependence on the intended configuration u_i^t can be modeled according to the failure probabilities of individual generators. Such an extension is completely compatible with the proposed modeling formalism of factored Markov decision processes.

References

Boutilier, C.; Dearden, R.; and Goldszmidt, M. 2000. Stochastic dynamic programming with factored representations. *Artificial Intelligence* 49–107.

Feng, Z.; Dearden, R.; Meuleau, N.; and Washington, R. 2004. Dynamic programming for structured continuous Markov decision problems. In *Proceedings of the 20th conference on Uncertainty in Artificial Intelligence*, UAI '04, 154–161. Arlington, Virginia, United States: AUAI Press.

Li, C.; Johnson, R. B.; and Svoboda, A. J. 1997. A new unit commitment method. *IEEE Transactions on Power Systems* 113–119.

Martelli, A., and Montanari, U. 1973. Additive AND/OR graphs. In *Proceedings of the Third International Joint Conference on Artificial intelligence*, 1–11.

Nikovski, D., and Esenther, A. 2011. Construction of embedded Markov decision processes for optimal control of non-linear systems with continuous state spaces. In *IEEE Conference on Decision and Control and European Control Conference*, 7944–7949.

Nikovski, D.; Lidicky, B.; Zhang, W.; Kataoka, K.; and Yoshimoto, K. 2012. Markov decision processes for train run curve optimization. In *Electrical Systems for Aircraft, Railway and Ship Propulsion (ESARS), 2012*, 1–6.

Nikovski, D.; Xu, J.; and Nonaka, M. 2013. A method for computing optimal set-point schedules for HVAC systems. In *Proceedings of the 11th REHVA World Congress CLIMA 2013*.

Preparata, F. P., and Shamos, M. I. 1990. *Computational Geometry*. Heidelberg: Springer Verlag.

Puterman, M. L. 1994. *Markov Decision Processes—Discrete Stochastic Dynamic Programming*. New York, NY: John Wiley & Sons, Inc.

Sanner, S.; Delgado, K. V.; and de Barros, L. N. 2011. Symbolic dynamic programming for discrete and continuous state MDPs. In *Proceedings of the 27th conference on Uncertainty in Artificial Intelligence*, UAI '11, 643–652. Arlington, Virginia, United States: AUAI Press.

Takriti, S.; Birge, J. R.; and Long, E. 1996. A stochastic model of the unit commitment problem. *IEEE Transactions on Power Systems* 1497–1508.

Wood, A. J., and Wollenberg, B. F. 1996. *Power Generation, Operation, and Control*. New York: Wiley-Interscience.

Xia, X., and Elaiw, A. M. 2010. Optimal dynamic economic dispatch of generation: a review. *Electric Power Systems Research* 975–986.

Xia, X.; Zhang, J.; and Elaiw, A. 2011. An application of model predictive control to the dynamic economic dispatch of power generation. *Control Engineering Practice* 19(6):638–648.

Flexible Execution of Partial Order Plans With Temporal Constraints*

Christian Muise[1], J. Christopher Beck[2], and **Sheila A. McIlraith[1]**

[1]Dept. of Computer Science
University of Toronto
{cjmuise,sheila}@cs.toronto.edu

[2]Dept. of Mechanical & Industrial Engineering
University of Toronto
jcb@mie.utoronto.ca

Abstract

We propose a unified approach to plan execution and schedule dispatching that converts a plan, which has been augmented with temporal constraints, into a policy for dispatching. Our approach generalizes the original plan and temporal constraints so that the executor need only consider the subset of state that is relevant to successful execution of valid plan fragments. We can accommodate a variety of calamitous and serendipitous changes to the state of the world by supporting the seamless re-execution or omission of plan fragments, without the need for costly replanning. Our methodology for plan generalization and online dispatching is a novel combination of plan execution and schedule dispatching techniques. We demonstrate the effectiveness of our method through a prototype implementation and a series of experiments.

1 Introduction

Plans and schedules often go awry because of unanticipated changes in the world. In such cases, it is up to the execution monitoring system (EM) to determine what to do. Typically, an EM represents the temporal plan it is executing as a partial-order plan (POP) with an associated simple temporal network (STN) [Dechter *et al.*, 1991] that captures the temporal constraints between actions [Younes and Simmons, 2003; Coles *et al.*, 2010]. The EM executes the POP's actions one after another until the goal is reached or a discrepancy is detected. Often, the EM is forced to resolve discrepancies through costly replanning, rescheduling, or plan repair (e.g., the IxTeT-eXeC system [Lemai and Ingrand, 2003]).

The focus of this paper is on maximizing the robustness of plan execution by minimizing the need for replanning. We propose an execution module, TPOPEXEC, which is comprised of two components: 1) COMPILER, an offline preprocessor that takes as input a POP and a set of temporal constraints, and produces a generalized representation; and 2) EXECUTOR, an online component that soundly selects a temporally consistent, *valid* plan fragment from the generalized plan. TPOPEXEC does no replanning or repair. Rather, it can serve as a component of a larger execution engine to reduce, but not eliminate, the need for replanning.

TPOPEXEC reacts to the state of the world, proposing the next action of one of a large number of valid plan fragments whose starting state satisfies the necessary conditions for plan validity. This enables TPOPEXEC to seamlessly elect to execute parts of a plan multiple times and/or to omit actions that are no longer necessary for achieving the goal.

Such flexibility can introduce ambiguity in the interpretation of temporal constraints. For example, if you must start eating 3 to 10 minutes after heating your dinner, and eating gets delayed causing you to re-heat, then what temporal relationship(s), if any, should exist between the first heating and the eating? As such ambiguities are not addressed by STNs, we introduce a specification language for temporal constraints that avoids the execution-time ambiguities and further supports the specification of constraints between state conditions and actions. We formally define the semantics of the temporal constraints and prove the correctness of TPOPEXEC. Compared to conventional EM systems, our approach has the potential to avoid replanning exponentially more often (in the size of the state). Experiments with a simulated uncertain environment show TPOPEXEC achieving the goal in 92% of the trials while the standard STN dispatching technique only has a success rate of roughly 30%.

TPOPEXEC leverages existing work from both partial-order plan execution and temporal reasoning. While many of the core algorithms are based on existing techniques, our main contribution stems from the dynamic creation of temporal subproblems that need to be solved during execution. Our approach is noteworthy for its novelty and broad applicability while making an important step towards integrating plan execution and schedule dispatching – tasks that are traditionally addressed independently [Smith *et al.*, 2000].

2 Preliminaries

STRIPS Following [Ghallab *et al.*, 2004], a *STRIPS Planning Problem* is a tuple $\Pi = \langle F, O, I, G \rangle$ where F is a set of fluent symbols, O is a set of action operators, and I and G are sets of fluents, corresponding to the initial state and goal condition. Every action $a \in O$ is defined by three sets of fluents PRE, ADD, and DEL, corresponding to the preconditions,

*This paper also appears in the Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013).

add effects, and delete effects. An action a is *executable* in state s iff $PRE(a) \subseteq s$. An executable sequence of actions is a *plan*. A plan that commences with I and terminates in G is a *valid plan* for G. Given a plan $a_0, \ldots, a_n$, the sequence of actions $a_i, \ldots, a_n$, where $i \geq 0$, is a *plan suffix*.

Partial-order Plans A partial-order plan (POP) is a tuple $P = \langle \mathcal{A}, \mathcal{O} \rangle$ where $\mathcal{A}$ is the set of actions in the plan and $\mathcal{O}$ is a set of orderings between the actions in $\mathcal{A}$ (e.g., $(a_1 \prec a_2) \in \mathcal{O}$) [Weld, 1994]. For this work, we do not require a set of causal links to be defined. A total ordering of the actions in $\mathcal{A}$ that respects $\mathcal{O}$ is a *linearization* of P. A POP provides a compact representation for multiple linearizations, and is considered *valid* iff every linearization is a valid plan. We further assume that the set of ordering constraints $\mathcal{O}$ in a valid POP is transitively closed. We typically add two special actions to the POP, a_I and a_G, that encode the initial and goal states through their add effects and preconditions. A *POP suffix* of a given POP $\langle \mathcal{A}, \mathcal{O} \rangle$ is any POP $\langle \mathcal{A}', \mathcal{O}' \rangle$ where (1) $\mathcal{A}' \subseteq \mathcal{A}$, (2) $\mathcal{O}' = \{(a_1 \prec a_2) \mid (a_1 \prec a_2) \in \mathcal{O}$ and $a_1, a_2 \in \mathcal{A}'\}$, and (3) $\forall a_1 \in \mathcal{A}', ((a_1 \prec a_2) \in \mathcal{O}) \rightarrow (a_2 \in \mathcal{A}')$. That is, every ordering originating from an action in the suffix implies the corresponding action is also in the suffix. While the same ground action may appear more than once in $\mathcal{A}$, we assume that every element of $\mathcal{A}$ is uniquely identifiable.

Durative Actions Following Fox and Long (2003), any durative action appears in the plan as a pair of instantaneous *start* and *end* actions that must alternate (a durative action must end before it can be started again). We further augment the pair of actions with a suitable temporal constraint to enforce the duration (cf. Section 3.1). We handle domains where durative actions must overlap, typically referred to as *required concurrency* [Fox *et al.*, 2004; Cushing *et al.*, 2007], but we do not handle situations where a single durative action must overlap with itself during execution [Coles *et al.*, 2008]: when a *start* action occurs, the corresponding *end* action must occur before the *start* can be executed again. Our work is focused on STRIPS planning problems and a set of temporal constraints inspired by those found in PDDL3.0. STRIPS cannot require that two instantaneous actions "execute in parallel" (e.g., [Boutilier and Brafman, 2001]), so this situation will not arise. In the future, we hope to expand to other aspects of PDDL including conditional effects, numeric state fluents, and continuous change.

3 Temporal Constraints and Traces

The starting point for our work is a *Temporally Constrained POP* (TPOP), $\langle \langle \mathcal{A}, \mathcal{O} \rangle, \mathcal{C} \rangle$, which comprises a valid POP $\langle \mathcal{A}, \mathcal{O} \rangle$, and a set of temporal constraints $\mathcal{C}$. Here, we do not concern ourselves with where the POP comes from, but options include using a partial-order planner (e.g., VHPOP [Younes and Simmons, 2003]), relaxing a sequential plan (e.g., [Muise *et al.*, 2012]), or having a POP specified by the user. Temporal constraints originate with the user, with the exception of those generated in the transformation of durative actions to pairs of instantaneous actions. This decoupling enables a POP to be re-used in multiple scenarios by simply varying the temporal constraints. In this section, we propose a syntax and semantics for the temporal constraints in $\mathcal{C}$.

Consider the TPOPEXEC for a mobile-phone based cognitive assistant (CA) that oversees a user's daily activities. The CA knows his/her plan for the day including activities such as laundry, transportation, dinner, and a movie. It is updated about the state of the world by the user, RSS feeds, etc. CA "executes" a plan by reminding the user to perform actions. As the state is updated, CA's EXECUTOR revises its reminders. If the user's date is delayed, they may need to rebook dinner. If the user's friend gives them $50, they can skip going to the bank.

As noted in Section 1, TPOPEXEC is able to seamlessly re-execute or omit portions of a plan. This can create ambiguity in the interpretation of standard STNs, as illustrated by the "heating & eating" example. It is also the case that many temporal constraints are more compellingly expressed as constraints between state properties and actions (e.g., "Be at the movie at least 15 minutes before it starts."). These two desiderata serve as motivation for our new language.

3.1 Temporal Constraints

Inspired by PDDL3.0 and linear temporal logic [Gerevini *et al.*, 2009; Pnueli, 1977], our language introduces four temporal modal operators ranging over the actions of our TPOP, the fluents in our planning domain, and time (the positive reals). Our language supports the specification of a set of constraints, but no connectives. During execution, actions in our TPOP may be repeated or skipped, requiring a formalism strictly more expressive than STNs: in an STN, there is no accommodation for unplanned re-execution or omission of actions, nor is there a facility to express temporal constraints with respect to the state of the world [Dechter *et al.*, 1991].

Definition 1. Temporal Constraint Types

- **(latest-before** $b\ a\ l\ u$**)**: A *past constraint* between actions a and b over bounds $l, u \in \mathbb{R}_{\geq 0} \cup \{\infty\}$ stipulates that if b occurs, then a must have occurred previously and the most recent occurrence of a is between l and u time units.
- **(earliest-after** $a\ b\ l\ u$**)**: A *future constraint* between actions a and b over bounds $l, u \in \mathbb{R}_{\geq 0} \cup \{\infty\}$ stipulates that if a occurs, then b must occur in the future and the next occurrence is between l and u time units.
- **(holds-before** $a\ f\ l\ u$**)**: A *past fluent constraint* between an action a and fluent f over bounds $l, u \in \mathbb{R}_{\geq 0} \cup \{\infty\}$ stipulates that if a occurs, then f must have held between l and u time units in the past.
- **(holds-after** $a\ f\ l\ u$**)**: A *future fluent constraint* between an action a and fluent f over bounds $l, u \in \mathbb{R}_{\geq 0} \cup \{\infty\}$ stipulates that if a occurs, then f must hold between l and u time units in the future.

The notation mirrors the PDDL3.0 preference syntax: e.g., **(latest-before** $b\ a\ l\ u$**)** should be read as "the latest occurrence of action a before an occurrence of b is between l and u time units". For the CA example, consider the temporal constraint, **(latest-before** $exercise\ eat\_meal$ 30 240**)**: exercise must be more than a half hour and no more than four hours

after eating. If, after exercising, the user decides to exercise again (due to an exogenous change in the world), the timing of the second exercise action must be consistent with the constraint and the timing of the most recent meal. If the constraint had been **(earliest-after** eat_meal $exercise$ 30 240**)**, there would be no constraint between the meal and the second occurrence of exercise: that constraint is only relevant to the *earliest* occurrence of exercise after a meal.

If $a_{\vdash}$ and $a_{\dashv}$ denote the instantaneous actions corresponding to the start and end of a durative action a, we augment the domain with **(latest-before** $a_{\dashv}$ $a_{\vdash}$ l u**)** where l and u are lower and upper bounds on the duration of a.

3.2 Semantics

We define the semantics of our temporal constraints with respect to the execution trace of the plan – a history of action-state pairs, indexed by time and represented as a *timed word* [Alur and Dill, 1994].

Definition 2. Trace

Given a TPOP $\langle\langle \mathcal{A}, \mathcal{O}\rangle, \mathcal{C}\rangle$ and planning problem $\langle F, O, I, G\rangle$, we define a *trace*, $\mathcal{T}$, to be a finite timed word, $(\sigma_0, t_0), \cdots, (\sigma_n, t_n)$, where $\sigma_i \in \Sigma$, the alphabet Σ ranges over $\mathcal{A} \times \mathcal{S}$, and the time values, $t_i \in \mathbb{R}_{\geq 0}$, are strictly increasing. $\mathcal{T}$ is *executable* iff for every $((a_i, s_i), t_i)$ in $\mathcal{T}$, a_i is executable in s_i. $\mathcal{T}$ is *static* iff for every $((a_i, s_i), t_i)$ in $\mathcal{T}$, if $i > 0$ then s_i is the result of executing a_{i-1} in state s_{i-1}. We signify the concatenation of traces $\mathcal{T}$ and $\mathcal{T}'$ as $\mathcal{T} \cdot \mathcal{T}'$.

We assume that the state of the world is fully observable. If a fluent changes unexpectedly (e.g., through an exogenous event), a tuple in the trace reflects this change. A single tuple $((a_i, s_i), t_i) \in \mathcal{T}$ is an *occurrence* , and we refer to the actual trace of performed actions as an *execution trace*. An execution trace is *valid* if it is executable, satisfies every temporal constraint, and the final action is a_G (i.e., the goal is achieved). Finally, we say that an execution trace is a *valid partial trace* if it can be extended to be a valid trace.

Figure 1 defines the semantics of our temporal modal operators with respect to a trace. We use the abbreviation **(time-diff** i j l u**)** $\stackrel{\text{def}}{=} l \leq t_j - t_i \leq u$ to indicate that the time between indices i and j is bounded between l and u; **(occ** a i**)** to denote that action a occurred at t_i in the trace. For both variants of a future constraint, $\mathcal{T}$ may be a valid partial trace but not a valid trace because the constraint is not yet satisfied – e.g., for **earliest-after**, a appears in $\mathcal{T}$, but b has not occurred since then. Such constraints are *unresolved*.

4 Generalizing and Executing TPOPs

Typical EMs execute actions in a plan in the order prescribed, until the goal is reached or a discrepancy is detected. At that point, they trigger replanning or rescheduling [Lemai and Ingrand, 2003; Conrad and Williams, 2011; Levine, 2012]. The dispatching of STNs operates in a similar fashion [Dechter *et al.*, 1991]. In contrast, TPOPEXEC executes the first action of the cheapest valid plan fragment whose starting state satisfies the necessary conditions for plan validity and satisfies the temporal constraints. Such robustness is not found in existing methods without explicit replanning or rescheduling.

Our approach is to provide a flexible representation that generalizes a plan to capture, for each step of the plan, the necessary subset of state required to ensure the plan's validity. In the case of a TPOP, which compactly encodes k linearizations (sequential plans of length n), this generalization produces up to kn sequential plans of lengths ranging from 1 to n, each leading to the goal but starting in different states. The generalized TPOP is represented as a policy, and it allows for the choice of any one of the (up to) kn sequential plans, filtering out those that do not respect the temporal constraints. Our policy enables TPOPEXEC to choose between the execution of different linearizations depending on the state of the world. In doing so, it can accommodate a number of unanticipated changes, either calamitous or serendipitous.

TPOPEXEC is comprised of an offline preprocessing phase (COMPILER) and an online computation phase (EXECUTOR). COMPILER systematically computes every possible temporally consistent partial plan that corresponds to a suffix of the input TPOP. EXECUTOR retrieves a temporally consistent partial plan that it can use to achieve the goal. To simplify the exposition, we present our approach for a subclass of TPOPs where the constraints, $\mathcal{C}$, are restricted to involve only actions (i.e., the **latest-before** and **earliest-after** constraints): referred to as an ATPOP. In Section 4.3, we show how to express an arbitrary TPOP as an ATPOP. We assume that the ATPOP is provided to TPOPEXEC. Potential sources of an ATPOP include manually hand-coding one, annotating a standard POP with temporal constraints, or computing one with a dedicated planner. For this work, however, we focus on executing an ATPOP rather than its synthesis.

4.1 COMPILER: Offline Generalization

Given an ATPOP, execution trace, and state of the world, TPOPEXEC needs to determine if any fragment of the ATPOP can achieve the goal and satisfy all of the temporal constraints while taking the trace so far into account. We compile the causal and temporal conditions required for every partial plan into a policy that indicates if we can still reach the goal, and if so, what action to execute next and when.

The key component of the policy representation is a *partial plan context*. Given an ATPOP, a partial plan context captures a subset of the original ATPOP actions and orderings, a candidate action, a, and a set of sufficient conditions, ψ, both for the execution of a and to guarantee that some linearization of the ATPOP suffix starting from a will lead to the goal, ignoring for the moment the temporal constraints.

Definition 3. Partial Plan Context

Given a problem $\langle F, O, I, G\rangle$ and ATPOP $\langle\langle \mathcal{A}, \mathcal{O}\rangle, \mathcal{C}\rangle$, we define a *partial plan context* as a tuple $\langle \mathcal{A}_{\dashv}, \mathcal{O}_{\dashv}, \psi, a\rangle$, where:

1. $\mathcal{A}_{\dashv} \subseteq \mathcal{A}$ is the set of actions to be executed.
2. $\mathcal{O}_{\dashv}$ is a set of ordering constraints over $\mathcal{A}_{\dashv}$.
3. $\psi \subseteq F$ is a set of fluents sufficient for executing $\mathcal{A}_{\dashv}$.
4. $a \in \mathcal{A}_{\dashv}$ and $\nexists a' \in \mathcal{A}_{\dashv}\ s.t.\ (a' \prec a) \in \mathcal{O}_{\dashv}$

$((a_0, s_0), t_0), \cdots, ((a_n, s_n), t_n) \models$ **(latest-before** $b\ a\ l\ u$**)**
iff $\forall j : 1 \leq j \leq n$ if **(occ** $b\ j$**)** then $\exists i : 0 \leq i < j,$ **(occ** $a\ i$**)** $\wedge$ **(time-diff** $i\ j\ l\ u$**)** $\wedge\ \forall k : i < k < j,\ a_k \neq a$

$((a_0, s_0), t_0), \cdots, ((a_n, s_n), t_n) \models$ **(earliest-after** $a\ b\ l\ u$**)**
iff $\forall i : 0 \leq i \leq n-1$ if **(occ** $a\ i$**)** then $\exists j : i < j \leq n,$ **(occ** $b\ j$**)** $\wedge$ **(time-diff** $i\ j\ l\ u$**)** $\wedge\ \forall k : i < k < j,\ a_k \neq b$

$((a_0, s_0), t_0), \cdots, ((a_n, s_n), t_n) \models$ **(holds-before** $a\ f\ l\ u$**)**
iff $\forall j : 1 \leq j \leq n$ if **(occ** $a\ j$**)** then $\exists i : 0 \leq i < j,\ (f \in s_i) \wedge [\exists t^* : (t_i \leq t^* \lneq t_{i+1}) \wedge (l \leq t_j - t^* \leq u)]$

$((a_0, s_0), t_0), \cdots, ((a_n, s_n), t_n) \models$ **(holds-after** $a\ f\ l\ u$**)**
iff $\forall i : 0 \leq i \leq n-1$ if **(occ** $a\ i$**)** then $\exists j : i < j \leq n,\ (f \in s_j) \wedge [\exists t^* : (t_j \leq t^* \lneq t_{j+1}) \wedge (l \leq t^* - t_i \leq u)]$

Figure 1: Semantics of the temporal modal operators with respect to a trace. $l, u \in \mathbb{R}_{\geq 0} \cup \{\infty\}$, $l \leq u$, and a, b are actions.

Context viability captures the notion that there exists a linearization of the partial plan context's POP that is valid and satisfies every constraint. Formally, given a planning problem Π, ATPOP $\langle\langle \mathcal{A}, \mathcal{O}\rangle, \mathcal{C}\rangle$, valid partial trace $\mathcal{T}$, and current state of the world s, a partial plan context $\langle \mathcal{A}_{\dashv}, \mathcal{O}_{\dashv}, \psi, a\rangle$ is (1) *causally viable* wrt. Π and s iff the POP $\langle \mathcal{A}_{\dashv}, \mathcal{O}_{\dashv}\rangle$ has a linearization starting with a that is a valid plan for the planning problem with s as the initial state, (2) *temporally viable* wrt. $\mathcal{C}$ and $\mathcal{T}$ iff there exists a trace $\mathcal{T}'$ where the actions in $\mathcal{T}'$ correspond to a linearization of $\langle \mathcal{A}_{\dashv}, \mathcal{O}_{\dashv}\rangle$ and $\mathcal{T} \cdot \mathcal{T}'$ satisfies every temporal constraint in $\mathcal{C}$, and (3) simply *viable* wrt. Π, s, $\mathcal{C}$, and $\mathcal{T}$ iff $\langle \mathcal{A}_{\dashv}, \mathcal{O}_{\dashv}\rangle$ has a linearization making the context both causally and temporally viable.

Establishing Causal Viability To generate every causally viable context, we appeal to the approach of Muise et al. (2011) which transforms a POP into a policy. As part of their process, they produce a sequence of condition-action pairs where the condition holds in a state iff some linearization of the POP has a suffix that can reach the goal starting with the action. Space prohibits us from a full exposition, but we modify their algorithm in two ways: (1) rather than simply record the condition ψ and candidate action a, we also record the set of actions and ordering constraints to build a partial plan context, and (2) additional ordering constraints are computed to ensure that, when establishing temporal viability, we reason about the correct linearization. Both modifications are primarily for bookkeeping and the soundness of the subsequent steps. Neither modification has a significant impact on the algorithm's performance. The following proposition follows from the proof of correctness of Muise et al.'s causal viability algorithm (Muise et al. 2011, Theorem 2).

Proposition 1. *Every partial plan context, $\langle \mathcal{A}_{\dashv}, \mathcal{O}_{\dashv}, \psi, a\rangle$, that we produce is causally viable wrt. Π and s iff $\psi \subseteq s$.*

Establishing Temporal Viability Given the temporal constraints, for each partial plan context, COMPILER determines temporal viability by proving consistency of a carefully constructed *context-specific STN* (CSTN).

An STN consists of a set of events and a set of simple temporal constraints. We use X_a to signify an event for action a, and make the distinction between an action a and an event X_a corresponding to an execution of a. A simple temporal constraint restricts the time between two events to be between a pair of bounds: $[l, u]_{X_{a_1}, X_{a_2}} \stackrel{\text{def}}{=} l \leq t(X_{a_2}) - t(X_{a_1}) \leq u$ where $t(\cdot)$ is a mapping of events to time-points. A CSTN contains events corresponding to the scope of the set of simple temporal constraints relevant to the context. The set of relevant simple temporal constraints, with respect to the ATPOP $\langle\langle \mathcal{A}, \mathcal{O}\rangle, \mathcal{C}\rangle$ and the context $\langle \mathcal{A}_{\dashv}, \mathcal{O}_{\dashv}, \psi, a\rangle$, consists of:

1. Temporal constraints on the unexecuted actions in $\mathcal{A}_{\dashv}$:
$$\{[\epsilon, \infty]_{X_{a_1}, X_{a_2}} \mid (a_1 \prec a_2) \in \mathcal{O}_{\dashv}\}$$
2. Past temporal constraints ending in $\mathcal{A}_{\dashv}$:
$$\{[l, u]_{X_{a_1}, X_{a_2}} \mid \textbf{(latest-before } a_2\ a_1\ l\ u\textbf{)} \in \mathcal{C}, a_2 \in \mathcal{A}_{\dashv}\}$$
3. Future temporal constraints involving only $\mathcal{A}_{\dashv}$:
$$\{[l, u]_{X_{a_1}, X_{a_2}} \mid \textbf{(earliest-after } a_1\ a_2\ l\ u\textbf{)} \in \mathcal{C}, a_1, a_2 \in \mathcal{A}_{\dashv}\}$$

COMPILER stores only those contexts that have a temporally consistent CSTN [Muscettola *et al.*, 1998]. Such a CSTN may or may not lead to a temporally viable partial plan context depending on the actual timing of *occurrences*. To enable a quick, online re-calculation of temporal viability, COMPILER stores the temporal windows between event X_a and events in the CSTN that correspond to actions *outside* of $\mathcal{A}_{\dashv}$. We ignore future constraints with a single action outside of $\mathcal{A}_{\dashv}$, because without knowing if the first action appears in the execution trace, the CSTN should not include it.

4.2 EXECUTOR: Online Execution

Given the state of the world and execution trace, EXECUTOR follows a four step process: (1) retrieve the set of causally viable partial plan contexts, (2) sort the contexts in ascending distance-to-goal, (3) identify the first context that is temporally viable, and (4) return the leading action and its temporal window. To determine temporal viability, given an execution trace and the stored temporal windows for events that have occurred, EXECUTOR uses the following two-step process:

1. If there are unresolved future constraints in the trace, rebuild the CSTN and recheck its consistency.
2. Simulate the execution of past events in the CSTN.

Resolving Future Temporal Constraints For every unsatisfied future temporal constraint **(earliest-after** $a_1\ a_2\ l\ u$**)**,

we have a set of *occurrences* that are the cause for the constraint remaining unsatisfied: the *occurrences* containing a_1 that have happened *after* the most recent *occurrence* containing a_2. If X_{a_1} does not already exist in the CSTN, then EXECUTOR adds event, X_{a_1}, corresponding to the *latest occurrence of* a_1 and includes the simple temporal constraint $[l, u]_{X_{a_1}, X_{a_2}}$. If there is more than one *occurrence* that serves as a reason for the unsatisfied constraint, EXECUTOR adds another event, X'_{a_1}, to the CSTN corresponding to the *earliest occurrence* containing a_1 (with the constraint $[l, u]_{X'_{a_1}, X_{a_2}}$). The remaining *occurrences* containing a_1 can be ignored as they cannot further constrain the CSTN. EXECUTOR then rechecks for consistency to ensure temporal viability.

Simulating Previous Events Using the standard dispatching algorithm for an STN [Muscettola *et al.*, 1998], EXECUTOR tests if a schedule exists for the actions in $A_{\dashv}$ that adheres to all of the temporal constraints and the execution trace. For every event X_a in the CSTN where $a \notin A_{\dashv}$, we have a corresponding latest *occurrence* $((a, s_i), t_i) \in \mathcal{T}$ (start actions for active future temporal constraints also have an associated *occurrence*). EXECUTOR follows the order found in $\mathcal{T}$ to dispatch each event at the time already established, propagating the start times. If EXECUTOR must dispatch an event at a time outside of its temporal bounds, then the network is inconsistent (cf. Theorem 2 of Muscettola et al. (1998)). If the temporal windows remain non-empty, then the process ends with a temporal window for the event corresponding to the candidate action, a. This provides EXECUTOR both with a certificate that the CSTN is consistent, and indicates what should be done: execute a within its temporal window.

Theorem 1. *Given an ATPOP $\langle\langle \mathcal{A}, \mathcal{O}\rangle, \mathcal{C}\rangle$, valid partial trace $\mathcal{T}$, and partial plan context $\langle \mathcal{A}_{\dashv}, \mathcal{O}_{\dashv}, \psi, a\rangle$, the context is temporally viable iff the context's CSTN is consistent and can be dispatched following the above two steps.*

Proof sketch. For the context to be temporally viable, $\langle \mathcal{A}_{\dashv}, \mathcal{O}_{\dashv}\rangle$ must have a linearization that corresponds to some trace $\mathcal{T}'$ such that $\mathcal{T} \cdot \mathcal{T}'$ satisfies every constraint in $\mathcal{C}$. The first set of constraints included in the CSTN ensures that any schedule follows a linearization of $\langle \mathcal{A}_{\dashv}, \mathcal{O}_{\dashv}\rangle$. The CSTN is consistent and dispatchable iff there is a schedule of the actions in $\mathcal{A}_{\dashv}$ that satisfies every constraint. We thus have a candidate for $\mathcal{T}'$ iff the context is temporally viable. □

Combining Proposition 1 and Theorem 1, we can now ascertain how TPOPEXEC leverages a partial plan context:

Theorem 2. *For a given planning problem Π, ATPOP $\langle\langle \mathcal{A}, \mathcal{O}\rangle, \mathcal{C}\rangle$, execution trace $\mathcal{T}$, state of the world s, and partial plan context $\langle \mathcal{A}_{\dashv}, \mathcal{O}_{\dashv}, \psi, a\rangle$, the partial plan context is viable iff (1) $\psi \subseteq s$, (2) the context's CSTN is consistent, and (3) the context's CSTN can be dispatched.*

4.3 Discussion

We have built computational machinery to enable TPOPEXEC to select a next action and the timing of its execution. Following Theorem 2, as long as a suffix of some linearization of the POP can achieve the goal while satisfying all temporal constraints, TPOPEXEC will eventually achieve the goal. Because determining temporal viability requires some amount of reasoning online, EXECUTOR filters first based on causal viability, and then discards the contexts which are not temporally viable. To choose amongst temporally viable contexts, EXECUTOR prefers the context with the best plan quality based on action cost, breaking ties by the minimum temporal distance between the current time and the goal.

The complexity for computing the causally viable contexts online is at worst linear in the number of contexts, but in practice is much smaller – typically linear in the size of the relevant portion of the current state. The complexity of determining temporal viability is at worst polynomial in the size of the CSTN. However, we have identified many heuristic checks that successfully determine, in the overwhelming majority of situations, whether or not the CSTN is temporally consistent. Naively stored, the number of contexts and temporal networks may pose a problem. However, by leveraging the commonality between the contexts and their temporal networks, we were able to drastically reduce the overall storage compared to [Muise *et al.*, 2011] to store both the state and temporal information.

Optimizations We augmented these methods with a number of critical optimizations. Among the most important are the following: (1) When constructing a CSTN, COMPILER keeps only those events in the scope of any temporal constraint in the CSTN while retaining the transitive ordering from all actions in $\mathcal{A}_{\dashv}$ (not every action in $\mathcal{A}_{\dashv}$ must be a part of a temporal constraint). This reduces the size and complexity of the STN. (2) Rather than always doing a full consistency check for testing temporal viability in the presence of open future temporal constraints, EXECUTOR evaluates a number of necessary or sufficient conditions first. EXECUTOR uses a full consistency check only when more efficient checks fail. Space precludes us from detailing the techniques here, but one example is that if no future temporal constraint tightens a lower or upper bound on an unexecuted action, then the previously compiled temporal windows remain valid: if the CSTN was found to be consistent in the offline phase, then it remains consistent as long as the temporal windows are not tightened.

Reformulation to ATPOP The approach described applies only to ATPOPs. We reformulate TPOPs involving fluent temporal constraints into ATPOPs, enabling the elegant application of our approach to arbitrary TPOPs. The reformulation is sound, but incomplete with respect to the **holds-before** constraint. For completeness, we must expand the temporal reasoning to handle disjunctive constraints (i.e., having a **sometime-before** constraint between actions).

We reformulate our fluent temporal constraints to action constraints. As such, for each fluent participating in a temporal constraint, we introduce an auxiliary action a_f with the precondition of $PRE(a_f) = \{f\}$ and no add or delete effects. These actions are used to record the observation of fluents. We then replace the fluent temporal modal operators with suitable counterparts: **(holds-before** $a\ f\ l\ u$**)** (resp. **(holds-after** $a\ f\ l\ u$**)**) is replaced by **(latest-before** $a\ a_f\ l\ u$**)** (resp. **(earliest-after** $a\ a_f\ l\ u$**)**). This modification permits TPOPEXEC to observe necessary facts at a time required. Finally, we require a unique auxiliary action for each fluent constraint, as sharing the auxiliary actions is unsound.

5 Experimental Evaluation

The core contribution of this work is the ability to execute a plan in a world that can change in unpredicted ways, while reasoning about ongoing causal and temporal viability. We accomplish this while avoiding unnecessary replanning, rescheduling, or plan repair. In building on the work of Muise et al. (2011), TPOPEXEC is able to continue executing a POP in exponentially many more states than traditional approaches that execute actions according to a prescribed ordering. Nevertheless, the work of Muise et al. (2011) cannot reason about temporal constraints. Many execution monitoring systems suffer a similar fate. Standard approaches to schedule dispatching, such as Muscettola et al. (1998), are blind to causal viability and the conditions for the executability of actions. Such approaches will only succeed on problems that do not experience obstructive change.

Our evaluation focuses on TPOPEXEC's robustness and ability to avoid replanning. We also evaluate the general properties of TPOPEXEC's behaviour. The first experiment demonstrates the increased capabilities of our approach over restricted forms of our method that improve on existing execution strategies (i.e., STN dispatching), and the second experiment examines the amount of replanning TPOPEXEC avoids. TPOPEXEC is written in Python, and we conducted the experiments on a Linux desktop with a 3.0GHz processor.

IPC benchmarks lack a combination of causal requirements and complex temporal constraints. As such, we tested our implementation on an expanded version of the CA domain that serves to challenge the causal and temporal reasoning, and is representative of what we might find in the real world. In total, there are 19 actions in the plan (11 durative), 8 past temporal constraints, and 4 future temporal constraints. The ATPOP has 18 ordering constraints which result in a total of 49,140 linearizations. The types of un-modelled dynamics include children becoming hungry, laundry being soiled, etc. In addition to being modelled after real-world temporal requirements, the constraints were designed to pose a challenge for TPOPEXEC. For example, often in the CA domain there is ample opportunity for a context to be causally viable but not temporally viable – the actions in the context can achieve the goal, but not without violating some temporal constraint. Such situations challenge the temporal reasoning aspects of TPOPEXEC to find the most appropriate context.

Rate of Success We simulate our CA agent in a world where fluents change unexpectedly in both positive and negative ways (i.e., adding and deleting fluents from the state). TPOPEXEC fails when EXECUTOR determines the goal is no longer causally and temporally achievable. The level of variability in the world is set using parameter α: $\alpha = 0$ corresponds to no changes whatsoever and $\alpha = 1$ corresponds to significant unpredictable change (at least one fluent changes after every action with 99.998% probability). For 20 different values of α, we ran 1000 trials for each approach. The proportion of successful trials is referred to as the *success rate*.

Ignoring the causal requirements and simply dispatching the ATPOP one action after another mirrors STN dispatching, which we argued above would be unsuccessful in most instances. Nonetheless, we test this approach to verify our intuition and evaluate the level of environmental variability that an STN dispatching algorithm can handle. We also present an *Opportunistic* version of TPOPEXEC that we restricted to execute an action at most once. It can, however, skip actions if positive changes allow. Figure 2 shows the success rate for all three approaches for a given value of α.

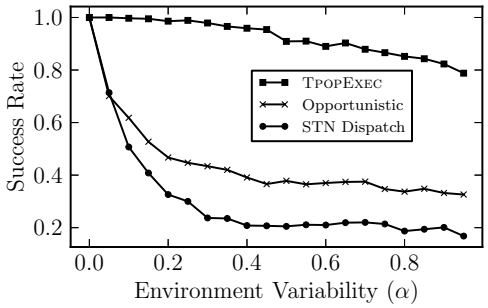


Figure 2: The success rate of the three approaches over a range of environment dynamics, both good and bad.

TPOPEXEC consistently outperforms both the ablated version and STN dispatching by a substantial margin – successfully executing the plan in more than 80% of the instances for almost all values of α and in total succeeding in over 92% of the simulations compared to just over 30% for the STN approach. Having the opportunity to re-execute plan fragments that are required again, while adhering to the imposed temporal constraints, provides us with a distinct advantage. The ablated version, while heavily restricted, still outperformed the STN dispatching for the majority of α-values, solving roughly twice as many instances.

Replan Avoidance We evaluate how often TPOPEXEC avoids replanning during execution in the CA domain. If other systems that replan online are able to replan quickly enough, then their execution behaviour would match ours. Every replan, however, requires solving a PSPACE problem which is what we avoid. Similar to the previous experiment, we evaluate with respect to a range of environment dynamics (the α parameter). However, to properly gauge the need for replanning, we only allow for negative changes to the world: fluents are randomly made false. We count the number of times during execution that TPOPEXEC would be forced to replan if it had not generalized the ATPOP, and we consider only those runs where TPOPEXEC reaches the goal. Figure 3 shows the mean replan rate for a given α-value, normalized by a theoretical maximum number of replans.

We find that the number of replans avoided increases linearly with the increase variability. Due to the temporal constraints on the length of the day, and the length of some durative actions, there is a theoretical limit of roughly 20 replans required for the dynamics we introduce. In situations where TPOPEXEC must operate over a larger time frame, we would expect the potential for replan avoidance to grow.

System Behaviour Profiling EXECUTOR, we found that 35% (resp. 60%) of the time was spent determining if contexts were causally (resp. temporally) viable. The remaining time was used for bookkeeping and data-structure updates for

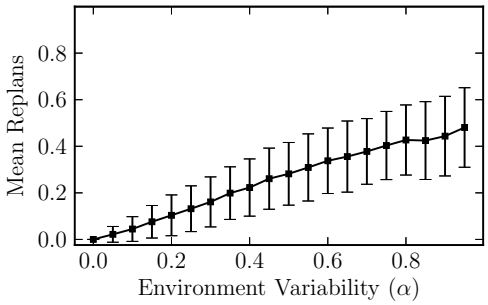


Figure 3: The number of replans that would be required during execution over a range of destructive environment dynamics (mean and standard deviation).

the simulation. COMPILER spent the vast majority of time checking the consistency for the CSTN of each of the 306 partial plan contexts. There is substantial commonality between CSTNs of similar contexts, and a potential optimization is to reuse the computation results. An average of 19 temporal windows were required for every CSTN. However, the memory bottleneck is the data-structure used for computing the causally viable contexts. Using a custom representation, we were able to reduce the memory requirements for our causal information by a factor of four compared to Muise et al. (2011), while our total footprint (causal plus temporal information) used about half the memory of just storing the causal information with the previous representation.

6 Related Work

Most approaches to executing plans with complex temporal constraints assume that an action in a plan will be executed once: an action may appear multiple times in a plan, but each plan appearance corresponds to exactly one action occurrence. IxTeT-eXeC executes actions from a temporally restricted POP and monitors the sufficient conditions for continued causal and temporal viability, replanning when they fail to hold [Lemai and Ingrand, 2003]. In a similar vein, the Pike system executes a temporally restricted POP while continuously monitoring a weaker set of conditions for temporal viability [Levine, 2012]. TPOPEXEC can be seen as an improved executor that tries to avoid plan repair and replanning, and we hope to incorporate our method into a larger system.

The Drake system focuses primarily on executing a plan with complex temporal constraints [Conrad and Williams, 2011]. While it can choose not to execute an action if non-execution is explicitly included as part of a complex temporal constraint, Drake does not represent or reason about causal validity. One avenue we hope to pursue is using the Drake temporal reasoning in place of our CSTNs. Doing so would allow us to handle more expressive constraints.

There is a large body of research on plan execution monitoring (e.g., [Pettersson, 2005; Fritz and McIlraith, 2007; Doherty *et al.*, 2009]). Some systems, such as the work of Doherty et al. 2009, monitor temporal constraints, but many do not. They typically focus on the feasibility of just one partial, sequential plan and resort to replanning when any condition is violated. There have been a number of temporal logics introduced for monitoring plans and schedules (e.g., [Koymans, 1990; Kvarnström *et al.*, 2008]). The most related to our work is TLTL [Bauer *et al.*, 2007]: it uses timed words at the core of its specification and provides a syntax capable of expressing the temporal constraints available to an ATPOP. They do not, however, include a mechanism for deciding what to do next. It is of interest to consider how we might expand our constraint specification language to handle all of TLTL.

7 Summary and Discussion

We presented TPOPEXEC, a system for generalizing and robustly executing a plan that is augmented with temporal constraints. In the face of unexpected changes in the world, TPOPEXEC can select from a large number of valid plan fragments that are consistent with the temporal constraints, repeating parts of a plan or omitting actions, as necessary. This is all done without the need to replan. To accommodate this flexibility, we introduced temporal constraints over actions and fluents, formalizing their semantics with respect to the execution trace. During execution, TPOPEXEC identifies the partial plans, computed offline, that can achieve the goal while satisfying all of the temporal constraints. To choose an action for execution, TPOPEXEC selects one at the start of the best quality partial plan identified as being viable.

We demonstrated our methodology through a prototype implementation and a series of experiments to test the robustness and flexibility of TPOPEXEC. In a simulated uncertain environment for a real-world inspired domain, TPOPEXEC achieved the goal in 92% of the trials while the standard STN dispatching technique succeeded 30% of the time.

We aim to address two fundamental limitations with our work: 1) temporal reasoning and schedule dispatching techniques typically do not consider the state of the world, and 2) execution monitoring schemes for planning problems that allow multiple action occurrences typically do not allow for temporal constraints to be defined. The temporal constraints that we introduce are an essential ingredient for the synthesis of plan execution and schedule dispatching techniques when the environment can change in unexpected ways. They also elucidate the need for referring to both state and actions as integral parts of a temporal constraint.

There may exist a tradeoff between the time saved by avoiding replanning and the quality of a new plan that could be found. In this work, we assume that replanning should be avoided if at all possible, but we hope to consider this tradeoff future work. As the contributions of TPOPEXEC can be seen as complementary to many existing execution monitoring systems (e.g., IxTeT-eXeC or Kirk [Lemai and Ingrand, 2003; Kim *et al.*, 2001]), we hope to incorporate our techniques into a larger system for wider application.

Acknowledgements

We would like to thank the anonymous reviewers whose valuable feedback helped improve the final paper. The authors gratefully acknowledge funding from the Ontario Ministry of Innovation and the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

[Alur and Dill, 1994] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[Bauer *et al.*, 2007] A. Bauer, M. Leucker, and C. Schallhart. Runtime verification for LTL and TLTL. *Transactions on Software Engineering and Methodology*, pages 1–68, 2007.

[Boutilier and Brafman, 2001] C. Boutilier and R. I. Brafman. Partial-order planning with concurrent interacting actions. *Journal of Artificial Intelligence Research*, 14:105–136, 2001.

[Coles *et al.*, 2008] A. Coles, M. Fox, D. Long, and A. Smith. Planning with problems requiring temporal coordination. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence*, pages 892–897, 2008.

[Coles *et al.*, 2010] A. J. Coles, A. I. Coles, M. Fox, and D. Long. Forward-chaining partial-order planning. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling*, pages 42–49, 2010.

[Conrad and Williams, 2011] P. R. Conrad and B. C. Williams. Drake: An Efficient Executive for Temporal Plans with Choice. *Journal of Artificial Intelligence Research*, 42:607–659, 2011.

[Cushing *et al.*, 2007] W. Cushing, S. Kambhampati, Mausam, and Weld D. S. When is temporal planning really temporal. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, pages 1852–1859, 2007.

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

[Doherty *et al.*, 2009] P. Doherty, J. Kvarnström, and F. Heintz. A temporal logic-based planning and execution monitoring framework for unmanned aircraft systems. *Autonomous Agents and Multi-Agent Systems*, 19(3):332–377, 2009.

[Fox and Long, 2003] M. Fox and D. Long. PDDL2.1: An extension to pddl for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20:61–124, 2003.

[Fox *et al.*, 2004] M. Fox, D. Long, and K. Halsey. An investigation into the expressive power of PDDL2.1. In *Proceedings of the 16th European Conference of Artificial Intelligence*, 2004.

[Fritz and McIlraith, 2007] C. Fritz and S. A. McIlraith. Monitoring plan optimality during execution. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling*, pages 144–151, 2007.

[Gerevini *et al.*, 2009] A. Gerevini, P. Haslum, D. Long, A. Saetti, and Y. Dimopoulos. Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. *Artificial Intelligence*, 173(5-6):619–668, 2009.

[Ghallab *et al.*, 2004] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: Theory & Practice*. Morgan Kaufmann, 2004.

[Kim *et al.*, 2001] P. Kim, B. C. Williams, and M. Abramson. Executing reactive, model-based programs through graph-based temporal planning. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, pages 487–493, 2001.

[Koymans, 1990] R. Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.

[Kvarnström *et al.*, 2008] J. Kvarnström, P. Doherty, and F. Heintz. A temporal logic-based planning and execution monitoring system. In *Proceedings of the 18th International Conference on Automated Planning and Scheduling*, pages 332–377, 2008.

[Lemai and Ingrand, 2003] S. Lemai and F. Ingrand. Interleaving temporal planning and execution: IxTeT-eXeC. In *Proceedings of the ICAPS Workshop on Plan Execution*, 2003.

[Levine, 2012] S. J. Levine. Monitoring the execution of temporal plans for robotic systems. *Master's Thesis*, 2012.

[Muise *et al.*, 2011] C. Muise, S. A. McIlraith, and J. C. Beck. Monitoring the execution of partial-order plans via regression. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, pages 1975–1982, 2011.

[Muise *et al.*, 2012] C. Muise, S. A. McIlraith, and J. C. Beck. Optimally relaxing partial-order plans with MaxSAT. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling*, pages 358–362, 2012.

[Muscettola *et al.*, 1998] N. Muscettola, P. H. Morris, and I. Tsamardinos. Reformulating temporal plans for efficient execution. In *Proceedings of the 6th International Conference on Principles of Knowledge Representation and Reasoning*, pages 444–452, 1998.

[Pettersson, 2005] O. Pettersson. Execution monitoring in robotics: A survey. *Robotics and Autonomous Systems*, 53(2):73–88, 2005.

[Pnueli, 1977] A. Pnueli. The temporal logic of programs. In *Proceedings of the Eighteenth IEEE Symposium Foundations of Computer Science*, pages 46–57, 1977.

[Smith *et al.*, 2000] D. E. Smith, J. Frank, and A. K. Jónsson. Bridging the gap between planning and scheduling. *Knowledge Engineering Review*, 15(1):47–83, 2000.

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

[Younes and Simmons, 2003] H. L. S. Younes and R. G. Simmons. VHPOP: Versatile heuristic partial order planner. *Journal of Artificial Intelligence Research*, 20:405–430, 2003.

PDDL+ Planning with Events and Linear Processes

Amanda Coles and Andrew Coles
Department of Informatics,
King's College London, WC2R 2LS UK
email: firstname.lastname@kcl.ac.uk

Abstract

In this paper we present a scalable fully-automated PDDL planner capable of reasoning with PDDL+ events and linear processes. Processes and events model (respectively) continuous and discrete exogenous activity in the environment, occurring when certain conditions hold. We discuss the significant research challenges posed in creating a forward-chaining planner that can reason with these, and present novel state-progression and consistency enforcing techniques that allow us to meet these challenges. Finally we present results showing that our new planner, using PDDL+ domain models, is able to solve realistic expressive problems more efficiently than the current state-of-the-art alternative: a compiled PDDL 2.1 representation with continuous numeric effects.

1 Introduction

Classical planning has traditionally been concerned with reasoning about a static world in which the effects of actions occur instantaneously. The reality of the world in which plans must be executed is, however, often different to this: numeric quantities change over time and exogenous happenings occur, both in response to, and independently of, the actions carried out in the plan. For example, at sunrise the battery charge of a space vehicle begins to increase continuously over time, this increase does not depend upon the vehicle taking any specific action, it happens automatically.

Even in the absence of exogeny, scalable automated planning in the presence of continuous numeric change has only recently become a possibility, due to advances in classical and temporal planning. While there was some early work on planning with such models, notably the planners Zeno (Penberthy and Weld 1994) and OPTOP (McDermott 2003b), the challenge of efficiently computing effective heuristics severely restricted scalability. Following the introduction of continuous numeric change into version 2.1 of the standard planning domain modelling language, PDDL, (Fox and Long 2003) a number of modern planners began to address the challenge of reasoning with continuous numeric change.

The planner COLIN (Coles et al. 2012) performs forward-chaining search and uses a mixed integer program (MIP) to ensure that the constraints arising due to the interaction of continuous numeric variables are met. POPF (Coles et al. 2010) extends COLIN to reason with partially ordered plans, and forms the basis for this work. Kongming (Li and Williams 2011) uses a planning graph based structure to build plans, making use of a proprietary language to specify continuous dynamics. It also uses a MIP to manage temporal and numeric constraints, but is less expressive than COLIN in the sense that it does not allow two actions to simultaneously change the value of a variable.

To date there are only two planners that are capable of reasoning with discrete and continuous change caused by both actions and exogenous happenings as described in PDDL+ (Fox and Long 2006). TM-LPSAT (Shin and Davis 2005) is a fully automated planner that can solve PDDL+ planning problems with linear continuous change. It uses a SAT-based compilation, giving a discrete set of time points; and, like COLIN, uses an LP solver to manage numeric constraints. Its approach shows promise, but empirically, suffers from scalability issues. UPMurphi (Penna et al. 2009) takes a model-checking approach but relies on a hand-crafted discretisation of time to reason with continuous change. The use of a discretisation allows it to handle non-linear continuous change, the only planner to do so, but of course requires human expertise. The main challenge for UPMurphi is scalability as it it has no heuristic for guidance.

In this paper we present a scalable forward-chaining planner capable of reasoning with linear continuous change and exogenous happenings. By building on state-of-the-art approaches to planning with continuous numeric change, we avoid the need to discretise time, with the consequence of improved scalability. Avoiding discretisation introduces new challenges in ensuring that exogenous happenings occur immediately when their conditions hold and that their conditions are avoided if they are not desired. We discuss how we overcome these challenges and empirically demonstrate the scalability of our planner on PDDL+ problems.

2 Problem Definition

The logical basis for temporal planning, as modelled in PDDL 2.1 (Fox and Long 2003), is a collection of propositions P, and a vector of numeric variables $\mathbf{v}$. These are manipulated and referred to by actions. The executability of actions is determined by their preconditions. A single *condition* is either a single proposition $p \in P$, or a numeric constraint over $\mathbf{v}$. We assume all such constraints are linear, and hence can be represented in the form:

$$\mathbf{w}.\mathbf{v}\{>, \geq, <, \leq, =\}c$$

($\mathbf{w}$ is a vector of constants and c is a constant). A *precondition* is a conjunction of zero or more conditions.

Each durative action A has three sets of preconditions: $\text{pre}_{\vdash}A$, $\text{pre}_{\leftrightarrow}A$, $\text{pre}_{\dashv}A$. These represent the conditions that must hold at its start, throughout its execution, and at the end of the action, respectively. Instantaneous effects can then be bound to the start or end of the action. $\text{eff}^{+}_{\vdash}A$ and $\text{eff}^{-}_{\vdash}A$ denote the propositions added and deleted at the start of A, and $\text{eff}^{num}_{\vdash}A$ denotes any numeric effects. Similarly, $\text{eff}^{+}_{\dashv}A$, $\text{eff}^{-}_{\dashv}$ and $\text{eff}^{num}_{\dashv}$ record effects at the end. We assume all such effects are linear, i.e. are of the form:

$$v\{\text{+=, -=, =}\}\mathbf{w}.\mathbf{v} + c \qquad \text{where } v \in \mathbf{v}$$

Semantically, the values of these instantaneous effects become available small amount of time, ϵ, after they occur.

Each action additionally has a conjunction of *continuous* numeric effects $\text{eff}_{\leftrightarrow}$, of the form dv/dt=$c, c \in \Re$, that occur while it is executing[1]. Finally, the action has a duration constraint: a conjunction of (assumedly linear) numeric constraints applied to a special variable dur_A corresponding to the duration of A. As a special case, *instantaneous* actions have duration ϵ, and have only one set of preconditions pre A and effects $\text{eff}^{+}A$ and $\text{eff}^{-}A$. For use in reasoning a durative action A can be split into two instantaneous *snap* actions, $A_{\vdash}$ and $A_{\dashv}$, representing the start and end of the action respectively, and a set of constraints (invariant and duration constraints and continuous numeric effects). Action $A_{\vdash}$ has precondition $\text{pre}_{\vdash}A$ and effects $\text{eff}^{+}_{\vdash}A \cup \text{eff}^{-}_{\vdash}A \cup \text{eff}^{num}_{\vdash}A$. Likewise, $A_{\dashv}$ is the analogous action for the end of A.

A PDDL+ planning problem augments a PDDL planning problem with processes and events. Like actions, these have preconditions, and effects. As an analogue, events are akin to instantaneous actions: if an event's preconditions pre A are satisfied, it occurs, yielding the event's instantaneous effects. Similarly, processes are akin to durative actions, with $\text{pre}_{\leftrightarrow}A$ corresponding to the process' precondition, and $\text{eff}_{\leftrightarrow}$ containing its continuous numeric effects. Then, while $\text{pre}_{\leftrightarrow}A$ is satisfied, the continuous numeric change occurs. Thus, the critical distinction between processes and events, and actions, is that a process/event will *automatically* occur as soon as its precondition is satisfied, modelling exogenous activity in the environment; whereas an action will only happen if chosen to execute in the plan.

PDDL+ has a number of problematic features that make the plan validation problem is intractable, even when the language is restricted to linear continuous change. In particular, in theory, events can infinitely cascade, repeatedly firing and self-supporting. Or, having reached the goals, it is challenging to determine whether they persist from then onwards, given future processes and events that may occur. To address the former of these issues, we make the restriction proposed by Fox and Long (2006) that events must delete one of their own preconditions. For the latter, we require that, if persistence is desired, the goal specified is sufficient to ensure the desired goals persist. Note that a goal required to be true beyond a specified fixed time, but not necessarily persist, can be modelled by using a process to count time and adding $time > time\_required$ to the goal.

[1]We allow c to be derived from mathematical operations on constant-valued state variables.

3 Running Example

We introduce a simple small example problem based on the use of a mobile phone (cellular phone). The scenario is as follows: a person initially in the countryside with his phone switched off must go to the city and make a call from there using his mobile phone (i.e. the goal is *called*[2]). The domain has three durative actions:

- **travel**: dur = 15; $\text{pre}_{\vdash}$ = {at country}; $\text{eff}^{-}_{\vdash}$ = {at country}; $\text{eff}^{+}_{\dashv}$ = {at city}; $\text{eff}_{\leftrightarrow}$ = $\{d(signal)/dt = 0.5\}$
- **turn_on**: dur > 0; $\text{pre}_{\vdash}$ = {$\neg$on}; $\text{eff}^{+}_{\vdash}$ = {on}; $\text{eff}^{-}_{\dashv}$ = {on}; $\text{eff}_{\leftrightarrow}$ = $d(battery)/dt$ = -1
- **call**: dur=1; $\text{pre}_{\vdash}$={at city $\wedge$ battery > 1}; $\text{eff}^{+}_{\dashv}$={called};

There is also a process, which models the transfer of data over the network at a fixed rate, if certain conditions are met:

- **transfer**: $\text{pre}_{\leftrightarrow} = \{\text{on} \wedge \text{battery} > 10 \wedge \text{signal} > 5\}$; $\text{eff}_{\leftrightarrow}$ = $d(data)/dt$ = 1

Finally, an event models a low-battery warning:

- **warning**: pre = {$\neg$warned$\wedge$battery< 8}; eff = {warned}

4 PDDL+ versus PDDL 2.1

In this section we further explore the relationship between PDDL+ processes and events and their PDDL 2.1 counterparts: durative-actions and (instantaneous) actions.

First, we observe that, at any time, each process p_i (with precondition C_i and effects $\text{eff}_{\leftrightarrow}p_i$) is either executing, or not; i.e. either C_i or $\neg C_i$. We might therefore consider there to be two durative-actions for p_i, rather than one:

- run_p_i, with $\text{pre}_{\leftrightarrow}$run_$p_i$=$C_i$, and $\text{eff}_{\leftrightarrow}$run_$p_i$=$\text{eff}_{\leftrightarrow}p_i$;
- not-run_p_i, with $\text{pre}_{\leftrightarrow}$not-run_$p_i$=$\neg C$, and no effects;
- in both cases, the duration of the action is in $[\epsilon, \infty]$.

If we could somehow then ensure that we only ever apply run_$p_{i\dashv}$ (the end of run_p_i) if we simultaneously apply not-run_$p_{i\vdash}$ – and vice-versa – then the behaviour of the process has been simulated with actions. Whenever the truth value of C_i changes (which may be many times) we simply switch which one of these two actions is executing. Ensuring this switch happens simultaneously is crucial: if time was allowed to pass between e.g. not-run_$p_{i\dashv}$ and run_$p_{i\vdash}$ then there would be a period during which C_i might be true, but the effect of p_i is not being captured by any executing action.

We also observe, that each event e_j with precondition C_j and effects $\text{eff}\,e_j$, at any point it is either happening at that time, instantaneously; or its conditions are false. Precisely:

- As events occur as soon as their preconditions are satisfied, there is a period prior to e_j during which $\neg C_j$ holds;
- When the event occurs, C_j is true – and, as noted earlier, events must delete one of their own preconditions;
- Thus, begins again a period in which $\neg C_j$ holds.

This could be captured with the use of synchronisation actions, similar to the mechanism postulated for processes. A non-temporal action e_j, with the preconditions and effects of e_j represents the event itself. Then, only one durative action is needed – not-do_e_j, with $\text{pre}_{\leftrightarrow}$not-do_$e_j$=$\neg C_j$. This

[2]Persistence is guaranteed: no action or event deletes this fact.

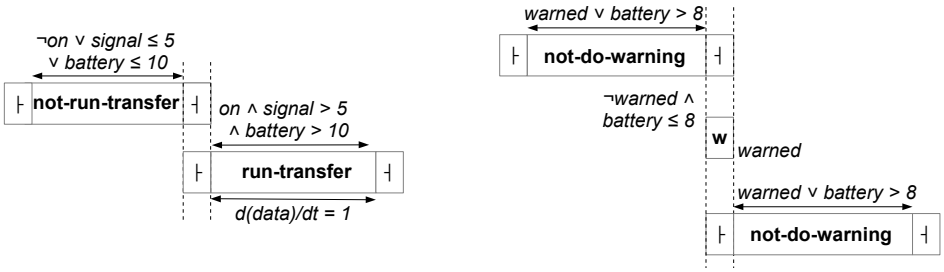


Figure 1: Representing Processes (left) and Events (right). Dotted lines denote synchronised actions.

captures the C_j intervals, with not-do_e_j ending and immediately re-starting at exactly the time e_j occurs.

Returning to our running example, the left of Figure 1 shows how synchronised actions can represent the 'transfer' process transitioning from not-running to running. To be not running, one of the terms in its precondition must be false; to be running, they must all be true. Synchronisation (dotted lines) then ensures this transition occurs at the right time. The right of Figure 1 shows the 'warning' event, abbreviated to 'w'. (The fact 'warned' ensures the event only fires once.) As can be seen, w is synchronised with the 'not-do-warning' durative-actions, again ensuring it occurs at the right time: the first point at which its precondition was met, and no later.

4.1 Achieving Synchronisation in PDDL 2.1

If we wish to reason with processes and events using a PDDL 2.1 planner we must use a compilation to enforce correct synchronisation. In fact, there are three requirements:

1. Synchronisation (as in Figure 1);
2. Ensuring that not-run_p_i (or run_p_i) and not-do_e_j are started *immediately*, at the start of the plan;
3. Allowing processes/event actions to finish in goal states.

We can achieve **synchronisation (1)**, through the use of *clip* actions (Fox, Long, and Halsey 2004):

Action 4.1 — clip_f
A (tight) clip for fact f, and auxiliary facts $\{fe_0, \ldots, fe_n\}$, is a durative action A, with duration 2ϵ, where:

- $\text{pre}_{\vdash} A = \neg f$, $\text{eff}^+_{\vdash} A = f$;
- $\text{pre}_{\dashv} A = \{fe_0, \ldots, fe_n\}$, $\text{eff}^-_{\dashv} A = \neg f, \{\neg fe_0, \ldots, \neg fe_n\}$.

f is thus only available, instantaneously, at time ϵ after starting clip_f; before immediately being deleted. If two snap-actions with a condition on f are placed inside the clip, they must therefore be synchronised: there is only one point during the clip at which their condition is met. The other aspect of a clip is its auxiliary facts, which must be true before it ends. If there are n+1 snap-actions that must occur during the clip, then each of these is given a distinct fe fact as an effect. Thus, not only must the snap-actions occur at the same time; but also, no necessary snap-action can be omitted.

To synchronise the actions from Section 4 in PDDL2.1, we use a clip for each process or event. For each process p_i:

- Create clip_f_i, with auxiliary facts $f_i e_0$, $f_i e_1$;
- Add f_i to $pre_{\vdash}$ and $pre_{\dashv}$ of run_p_i and not-run_p_i;
- Add $f_i e_0$ to $eff^+_{\dashv}$ of run_p_i and not-run_p_i.
- Add $f_i e_1$ to $eff^+_{\vdash}$ of run_p_i and not-run_p_i;

Similarly, for event e_j:

- Create clip_f_j, with auxiliary facts $f_j e_0$, $f_j e_1$, $f_j e_2$;
- Add f_i to $pre\ e_j$, and to $pre_{\vdash}$ and $pre_{\dashv}$ of not-do_e_j;
- Add $f_j e_0$ to $eff^+_{\dashv}$ not-do_e_j;
- Add $f_j e_1$ to $eff^+_{\vdash}$ not-do_e_j;
- Add $f_i e_2$ to $eff^+\ e_j$.

In both of these cases, the clip meets the objectives of Figure 1: by the use of 2 (resp. 3) auxiliary facts, the correct actions must occur within the clip; and due to the tight availability of f_i (f_j), the actions must be synchronised. Note that search using this compilation can be made slightly more efficient through the use of two clip actions for each process: one forcing a change from run-p_i to not-run-p_i and the other vice-versa. (The clip, as presented here, permits clipping run-p_i to run-p_i, and not-run-e_j to itself, which is pointless.)

The issue with the compilation as it stands is that a clip cannot end, unless an already-executing 'run', 'not-run' or 'not-do' action ends inside it: each of these adds the zero'th auxiliary fact of the clip, which is an end-condition of its execution. This is desirable in the general case, but in the initial state no such actions are executing. This brings us to the second of our requirements here, *viz.* **starting the actions immediately (2)**. For this, we use:

- A Timed Initial Literal (TIL) (Hoffmann and Edelkamp 2005) go, which is true initially and deleted at time ϵ, creating a small window at the start of the plan in which go is available. (NB go is not added by any action/event/TIL.)
- A TIL $exec$, appearing at time ϵ, and added as an 'at start' condition of every non-clip action in the plan.

We create a single 'go' clip allowing all process/event tracking actions to begin. We collate all the clip facts into a set F. We define the set FE_1 as the set of all '1' auxiliary facts (i.e. each $f_i e_1$ or $f_j e_1$). The go clip is defined thus:

Action 4.2 — go-clip
A 'go' clip for clip facts F, and 1-auxiliary fact set FE_1, is a durative action A, with duration 2ϵ where:

- $\text{pre}_{\vdash} A = go \wedge \forall f \in F\ \ \neg f$, $\text{eff}^+_{\vdash} A = F$;
- $\text{pre}_{\dashv} A = FE_1$, $\text{eff}^-_{\dashv} A = \forall f \in F\ \ \neg f$.

The condition go ensures the go-clip can only occur at time zero; and $exec$ ensures it precedes all other actions.

The final piece of the puzzle is to allow the run/not-run/do actions to **terminate once the goals have been met (3)** –

the semantics of PDDL 2.1 require there to be no executing actions in goal states. For this, we use a final, modified clip action – a 'goal-clip'. FE_0 is analogous to FE_1 – but for '0' facts – and $FE_{>0}$ contains all auxiliary facts not in FE_0.

Action 4.3 — goal-clip
A 'goal' clip for clip facts F, auxiliary fact sets FE_0 and $FE_{>0}$, in a problem with goal G, is a durative action A, with duration 2ϵ, where:

- $\text{pre}_{\vdash} A = \forall f \in F \ \neg f$, $\text{eff}^{+}_{\vdash} A = F$, $\text{eff}^{-}_{\vdash} A = exec$;
- $\text{pre}_{\dashv} A = G \ \wedge FE_0 \ \wedge \forall fe \in FE_{>0} \ \neg fe$, $\text{eff}^{+}_{\dashv} A = goal\_reached$, $\text{eff}^{-}_{\dashv} A = \forall f \in F \ \neg f$.

As a final note we observe that the negation of conjunctive conditions on processes/events results in disjunctive invariants on the not-run/not-do actions. In the absence of a planner supporting these, it is possible to create several not-run actions, each with an invariant comprising a single condition from the disjunction. Clips can then be used to switch between not-run actions to change which condition in the disjunct is satisfied. This has implications when using the efficient clip model (distinct clips for switching from run to not-run, and vice versa) – we must also allow different not-run actions to be clipped to each other. This does not, however, negate the benefits of the efficient clip model.

While this compilation to PDDL 2.1 is possible, it is clearly a very unnatural model, and still requires a highly expressive planner. Indeed several authors have argued that the model adopted in PDDL+ is much more natural than the previous model (McDermott 2003a; Boddy 2003). Further, it is likely to make search computationally inefficient: not only is the planner forced to reason about the exogenous actions within the environment as if they were real planning actions, many extra 'book-keeping' actions are added to the domain. If there are n processes and m events then $3n + 2m + 2$ actions are added to the planning problem, of which $(m + n)$ are applicable in each state. This massively the branching factor and the length of solution plans. Permutations of such actions can also cause significant problems in temporal planning (Tierney et al. 2012). It therefore seems that the native handling of processes and events is, in theory, far more efficient – and this forms the focus of the rest of the paper. We will, of course, return to this point in our evaluation.

5 Forward Chaining Partial-Order Planning

In this work, we build upon the planner POPF (Coles et al. 2010). POPF uses an adaptation of a forward-chaining planning approach where, rather than placing a total-order on plan steps, the plan steps are partially ordered: ordering constraints are inserted on an as-needed basis. To support this partial-ordering, additional information is stored in states, associated with the facts and variable values. For facts p:

- $F^{+}(p)$ ($F^{-}(p)$) records the index of the step that last added (deleted) p;
- $FP^{+}(p)$, a set of pairs, each $\langle i, d\rangle$, notes steps with a precondition p: i is the plan step index, and $d \in \{0, \epsilon\}$. If d=0, p can be deleted at or after step i; if d=ϵ, p can be deleted from ϵ after i.
- $FP^{-}(p)$, similarly, records negative preconditions on p.

For the vector of state variables $\mathbf{v}$, the state records lower- and upper-bound vectors, V^{min} and V^{max}. These reflect the fact that in the presence of continuous numeric change, a variable's value depends on the time; and hence, having applied some actions, a range of variable values are possible. With each $v \in \mathbf{v}$ the state also notes:

- $V^{\mathit{eff}}(v)$, the index of the most recent step to affect v;
- $VP(v)$, a set of plan step indices, of steps that have referred to v since the last effect on v. A step depends on v if either: it has a precondition on v; an effect whose outcome depends on v; or is the start of an action with a duration depending on v.
- $VI(v)$, a set of plan step indices, of the start of actions that are currently executing but have not yet finished; and have an over all condition on v.

The actions applied during search are snap-actions, corresponding to either instantaneous actions; the start of a durative action; or ending a (currently executing) durative action. When a snap-action is applied, the temporal constraints recorded are derived from the annotations: the new plan step is ordered after each fact or variable the action refers in its conditions, effects, and duration. Similarly, to avoid interference with existing plan steps, if the action deletes (adds) p it is ordered after each $FP^{+}(p)$ (resp. $FP^{-}(p)$); or if it affects v (either instantaneously, or by starting/ending a continuous numeric effect on v) it is ordered after each $VP(v)$, and after each $VI(v)$. Finally, in addition to these, constraints are added to respect the duration constraints on actions.

In the absence of continuous (or duration-dependent) numeric effects, the temporal-constraint consistency in POPF can be checked with a simple temporal network. However, with linear continuous numeric effects, a MIP solver is required for the resulting temporal–numeric constraints.

For each step i, $t(i)$ records its time-stamp, and the temporal constraints are encoded directly over these variables. Additionally, for each i, the variables $v_i \in V_i$ record the values of each of $\mathbf{v}$ prior to i for variables referred to in the preconditions/effects/duration constraints of step i). Likewise, $v'_i \in V'_i$ record the variable values immediately following i. The numeric preconditions and effects of actions are then added as constraints on these:

- preconditions at i form constraints over V_i;
- the invariants of i form constraints over V'_i if i is a start snap-action; or over V_i if it is an end snap-action.
- instantaneous numeric effects form constraints relating V_i to V'_i; for instance, $v'_i = v_i + x_i$ records that at step i, v is increased by the value of variable x.

In addition to the constraints for i itself, if i has an effect on v, it will be ordered after each $VI(v)$: the steps that have invariants on v. These invariants need to be enforced at step i: although they do not belong to i itself, they belong to currently executing actions, and we need to ensure i does not adversely interfere with these. Thus, v_i and v'_i are constrained to obey these invariants. This may necessitate the addition of extra ordering constraints, if an invariant to be enforced refers to a variable not otherwise relevant to the action. Hereon, if we state that an invariant must be *enforced* at step i, we mean that v_i and v'_i must be constrained

step	variables	constraint
turn_on$_\vdash$	t_0	≥ 0
	$battery_0$	$= 30$
	$battery'_0$	$= battery_0 \wedge > 0$
travel$_\vdash$	t_1	≥ 0
	$signal_1$	$= 0$
	$signal'_1$	$= signal_1$
travel$_\dashv$	t_2	$= t_0 + 15$
	$signal_2$	$= signal'_1 + 0.5 * (t_2 - t_1)$
	$signal'_2$	$= signal_2$
now	$battery_{now}$	$= battery'_0 - 1 * (t_{battery-now} - t_0)$
	$t_{battery-now}$	$> t_0$
	$signal_{now}$	$= signal'_2$
	$t_{signal-now}$	$> t_2$

Table 1: Example POPF MIP

to obey the invariant, and any extra ordering constraints must be added. For conjunctive invariants this neatly exploits the monotonicity of the continuous numeric change supported: for some interval in which the invariant on v must hold, it suffices to check the condition at the start and end of consecutive intervals, bounded by either the action to which the invariant belongs, or between successive effects on v. If, for example, $v \geq 5$ at the start and end of an interval, then under monotonic change it must have been true throughout.

To capture the interaction between time and variable values, the final consideration is the continuous numeric change that occurs over time. During MIP construction, at each point referring to v, the sum of the gradient effects δv acting on a variable v are noted. As the continuous numeric change is linear, and any changes to δv are totally ordered, at each point this is a constant value, known at the time the MIP is built. With $\delta v'_i$ denoting the gradient active after step i (assuming i refers to v), the value of v at a future step j is:

$$v_j = v'_i + \delta v'_i(t(j) - t(i)) \quad (1)$$

To illustrate the MIP built we return to our running example; since POPF does not handle processes/events we remove transfer and warning to demonstrate POPF's MIP. Table 1 shows the MIP that would be built in the state following the addition of travel$_\vdash$, turn_on$_\vdash$ and travel$_\dashv$ to the plan. Notice that each step only has MIP variables representing variables in its conditions/effects, or if an invariant is enforced; and it is only ordered w.r.t. other steps that affect or condition on the same variables/propositions. Because this ordering is guaranteed it is possible to compute $\delta' v_i$ at each t_i that has an effect on v, by working through the plan. In this example, $\delta signal'_0$=0.5, $\delta battery'_1$=-1 and $\delta signal'_2$=0.

The *now* steps are additionally added to the MIP, to allow the computation of the upper and lower bounds on state variables: for each numeric problem variable we ask the MIP solver to maximise and then to minimise its corresponding MIP variable and use these as the bounds (for brevity, we omit these from future MIPs in the paper). A solution to the MIP represents a valid setting of the timestamps of plan actions $t_0 ... t_n$ that respects all of the temporal and numeric constraints. If no such solution exists the plan to reach that state cannot be scheduled and the state can be pruned.

6 Search with Processes and Events

In Section 4 we detailed how synchronised actions can in principle be used for processes and events; and then in Section 4.1 detailed how this forms the basis of a compilation to PDDL 2.1 The drawback of this compilation is the search-space blow-up it entails. In this section we present an alternative approach: modifying a forward-chaining planning approach, to eliminate many of the artificial planning decisions entailed by the compilation.

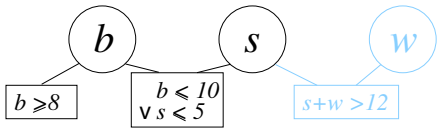


Figure 2: Condition–Variable Dependency Graph

6.1 Managing Invariants of Processes/Events

The run, not-run and not-do actions introduced by processes and events are, for many purposes, normal durative actions, with invariants – the only difference is the necessary synchronisation constraints. Thus, first, we need to consider how to decide when such invariants need to be enforced during planning. The basic approach in POPF was described in Section 5 – we build on this here.

The subset of invariants chosen by POPF to be enforced at a given step is sound if all invariants are conjuncts of single-variable constraints, e.g. $(battery > 10) \wedge (signal > 5)$. In other words, there is a direct relationship between the variables an action affects, and the constraints that need to be enforced. However, there are two cases where POPF, as it stands, cannot handle numeric invariants. These limitations only arise if variables referred to in the invariant have been subject to continuous numeric change, but in the context of processes, this is almost a certainty. POPF cannot handle:

- Invariants that are multi-variate e.g. $signal + wifi > 12$;
- Disjunctive invariants e.g. $(signal \leq 5) \vee (battery \leq 10)$.

The latter of these is particularly problematic with processes and events: even if a process/event has a condition that is a conjunct of terms, taking the negation of this, to mark the intervals during which the process/event is not occurring, leads to a disjunction (c.f. De Morgan's laws).

To handle such invariants, we require a more general solution to numeric invariants in POPF. Fundamental to our approach is a condition–variable dependency graph, built from the invariants of the currently executing actions. An example of such a graph is shown in Figure 2 – this is based on our running example, with an additional action whose invariant is $signal + wifi > 12$, the variable names are abbreviated to b,s, and w. The vertices are:

- One variable vertex for each numeric variable in the problem (in our example, b, s, w);
- $n+1$ constraint vertices for each invariant $C = (c_0 \wedge \ldots \wedge c_n)$, one for each term $c_i \in C$;
- One constraint vertex for each invariant $C = (c_0 \vee \ldots \vee c_n)$, containing the entire constraint C.

An edge is added between a constraint vertex and a variable vertex if the constraint refers to that variable. With this graph, we then have a straightforward way of ascertaining the *indirect* relationships between variables, that arise due to

disjunctive and multi-variate invariants. Simply: if a snap-action has an effect on a variable v, then any condition (invariant) that can be reached in the graph from v needs to be enforced at the point when the snap-action is applied.

We return to our running example to illustrate why this mechanism is necessary. Suppose we are in a state where the currently executing actions are turn_on, travel, not-run_transfer and not-do_warning. The condition–variable dependency graph comprises the dark (black) portion on the left of Figure 2. As turn_on and travel both have continuous numeric effects, the values of $battery$ and $signal$ are not fixed – they depend on the timestamps given to the actions in the plan, their ranges, as evaluated by the MIP, are $battery \in [0, 30]$ and $signal \in [0, 7.5]$. Suppose the action travel$_\dashv$ is then to be applied – which refers to the variable $signal$ in its effects. With the prior mechanism of POPF:

- the condition that would be enforced is $(battery \leq 10) \vee (signal \leq 5)$ – as it refers to $signal$ (we omit '$\ldots \vee \neg on$' from this discussion since on is known to be true);
- as the constraint is disjunctive, it could be satisfied by assuming for instance, $battery$=5: a value that lies within its range in the current state.

However, from the graph we see that if restrictions are made on the value of b, this may impact other conditions; in fact, assuming b=5 is incompatible with the invariant $b \geq 8$. This would, however, be captured by the new mechanism: upon referring to s, all reachable conditions are enforced, including those on b, due to the disjunctive constraint.

As an illustration of why the new mechanism is needed for multi-variate conditions too: we have the additional invariant shown on the right of Figure 2. Suppose an action is being applied that assigns w=5. This necessitates enforcing the invariant s+$w > 12$. With the prior mechanism of POPF, we could assume $s > 7.1$, which is within its range in the current state. But, from the graph we can see that constraints on s also need to be enforced. Notably, $s \leq 5$ can no longer be true if $s > 7.1$; and hence $b \leq 10$ has to be true; which, in turn, may impact whether $b \geq 8$ can be true (indeed had the condition on warning been, for example, $b \geq 12$ search would need to backtrack at this point). Thus, even though the action only affected w, the multi-variate and disjunctive invariants lead to indirect relationships with s and b, too.

6.2 Ordering Implications

A side effect of enforcing extra invariants is the addition of extra ordering constraints when adding actions to the plan. In our running example, when the action turn_on$_\vdash$ has been applied, and we consider applying travel$_\vdash$, the disjunctive invariant condition $\neg on \vee battery \leq 10 \vee signal \leq 5$ must be enforced. This leads to additional ordering constraints: using the POPF state-update rules, to establish the value of $battery$ for the purposes of enforcing this invariant, travel$_\vdash$ will be ordered after turn_on$_\vdash$. This would not have been the case had the disjunctive invariant not been enforced, as travel$_\vdash$ does not otherwise refer to $battery$. Note that completeness is not compromised as the state arising from applying these actions in the opposite order still appears as a distinct state in the search space. The practical effect of the ordering constraints is to impose a total order on actions affecting any variable in a set of connected variables in the condition-variable dependency graph. In the running example that means any action affecting b, s or w will be ordered with respect to any other action affecting b, s or w.

This has useful implications on our obligations to enforce disjunctive invariants. As a result of these orderings, we guarantee that we only need to maintain an invariant C=$(c_0 \vee ... \vee c_n)$ between plan steps at which C is enforced, and between which no step exists that could affect of the set of variables V_C (those referred to by any $c_f \in C$).

To understand this, suppose this invariant C became active at step i. Any action a_k with an effect on any $v \in V_C$ is totally ordered after the previous such action (c.f constraints introduced from condition-variable dependency graph). Let us name this totally ordered collection of actions A_C=$[a_0, \ldots, a_m]$, where $t(a_k) < t(a_{k+1})$, and $t(i) < t(a_0)$. At each a_k, C is enforced (when a_k is added to the plan). Therefore, when adding a new action a to the plan, we need only record the obligation to enforce the invariant at $t(a_m)$ (if A_C is not empty), and at $t(a)$: where $t(a_m)$ and $t(a)$ are adjacently ordered steps. Further, we know that no other action affecting any $v \in V_C$ occurs between $t(a_m)$ and $t(a)$: if such an action was added to the plan before a_m it would be ordered in A_C before a_m; and if it is later added, after a, it will be ordered in A_C after a.

6.3 Maintaining Disjunctive Invariants

In Section 5 we observed that in order to enforce a conjunctive invariant it is sufficient to *enforce* the invariant at the start and end of the interval over which it is required. This is not, however, sufficient for disjunctive invariants.

Consider, for example, meeting the invariant $(battery \leq 10) \vee (signal \leq 5)$ (the condition of not-run_transfer) during the interval between the actions travel$_\vdash$ and travel$_\dashv$, hereon step i and step j. This scenario is shown in Table 2. At this point in the plan, $battery$ is decreasing and $signal$ is increasing. If we simply insist that the disjunction is true at each end, we can rely on $(signal \leq 5)$ at the start and $(battery \leq 10)$ at the end but in fact both constraints could be false at some time during the interval: signal could become too large before battery becomes sufficiently small. Conversely, if we were to insist that either one of the conditions hold at both i *and* at j, we would preclude the possibility that for the first part of the interval we can rely on $(signal \leq 5)$; and then later, but before step j, rely on $(battery \leq 10)$. That is, we must allow changing of which condition we rely on part way through the interval.

Allowing for a potentially infinite number of such changes would be infeasible. Fortunately, in the general case for a disjunction C of $|C|$ numeric terms $c_1 ... c_{|C|}$ we need only include $|C|$-1 possible changing points. This result arises from the monotonicity of continuous numeric change: if we rely on a condition c_i until it becomes false, we will never be able to later rely on c_i as it cannot become true again. In our example, when $(signal \leq 5)$ becomes false, it cannot become true again until some later action affects $signal$ or the gradient on $signal$. As we saw in the previous section, there is a guarantee that between two adjacently ordered plan

step	variable	constraints
turn_on$_{\vdash}$	t_0	≥ 0
	$battery_0$	= 30
	$battery'_0$	= $battery_0 \wedge \geq 8$
travel$_{\vdash}$	t_1	$> t_0$
	$signal_1$	= 0
	$signal'_1$	= $signal_1$
	$battery_1$	$= battery'_0 - 1 * (t_1 - t_0) \wedge \geq 8$
	$battery'_1$	= $battery_1 \wedge \geq 8$
	$battery_1 \leq 10 \vee signal_1 \leq 5$	
	$battery'_1 \leq 10 \vee signal'_1 \leq 5$	
travel$_{\dashv}$	t_2	= t_1 + 15
	$signal_2$	= $signal'_1$ + 0.5*(t_2-t_1)
	$signal'_2$	= $signal_2$
	$battery_2$	$= battery'_0 - 1 * (t_2 - t_0) \wedge \geq 8$
	$battery'_2$	= $battery_2 \wedge \geq 8$
	$battery_2 \leq 10 \vee signal_2 \leq 5$	
	$battery'_2 \leq 10 \vee signal'_2 \leq 5$	
$\psi_{0-transfer}$	$t_{\psi 0}$	$\geq t_1 \wedge \leq t_2$
	$battery_{\psi 0}$	= $battery'_0$ - 1*($t_{\psi 0} - t_0$) $\wedge \geq 8$
	$signal_{\psi 0}$	= $signal'_1$ + 0.5*($t_{\psi 0}$-t_1)
	($battery'_1 \leq 10 \wedge battery_{\psi 0} \leq 10 \vee$ $signal'_1 \leq 5 \wedge signal_{\psi 0} \leq 5$) $\wedge$ ($battery_{\psi 0} \leq 10 \wedge battery_2 \leq 10 \vee$ $signal_{\psi 0} \leq 5 \wedge signal_2 \leq 5$)	

Table 2: Example MIP featuring a Disjunctive Invariant

steps at which a disjunctive invariant is enforced, no actions affecting the variables referred to in that invariant are applied. Therefore, if we select a true condition to rely on, and maintain that for as long as possible before switching to another condition, we need only $|C|$-1 changing points for each adjacently ordered pair of plan steps.

To maintain a disjunctive invariant in the interval between two adjacently ordered plan steps i,j at which it was enforced, we add changing points, each ψ_m, as totally ordered time-points in the MIP such that:

$$t_i \leq t(\psi_0) \ldots \leq t(\psi|C|\text{-}1) \leq t(j)$$

Between each adjacent pair of steps $[y, z]$ in this total order, we insist that there is a condition $c_i \in C$ which is true at y *and* true at z. In doing so, at least one $c \in C$ is true at all points over the interval $[t(i), t(j))]$. Table 2 shows the changing point $\psi_{0-transfer}$ and its associated constraints that enforce the satisfaction of the disjunction $(battery \leq 10) \vee (signal \leq 5)$ between travel$_{\vdash}$ and travel$_{\dashv}$. Notice it is possible to either rely on one condition for the whole interval if desired; or to switch conditions at $\psi_{0-transfer}$.

6.4 Synchronising Process/Event Actions

As discussed in Section 4, the action analogues used for processes and events must be synchronised. By modifying the planner, we can achieve this quite readily. Suppose not-run_p_i is executing, and the decision is made to apply run_$p_{i\vdash}$, as step k of a plan. We can treat this as a special case, insisting that, at step k, we simultaneously apply not-run_$p_{i\dashv}$. The constraints on the resulting plan step are:

- Due to not-run_$p_{i\dashv}$, V_k (the values of the variables immediately at step k) must satisfy the invariants of not-run_p_i;
- Due to run_$p_{i\vdash}$, V'_k (the values of the variables immediately after step k) must satisfy the invariants of run_p_i.

This slightly abuses the PDDL 2.1 semantics: strictly, the invariants of not-run_p_i only need to hold to the end of the half-closed interval ending at step k, i.e. up to, but excluding, the values of the variables at V_k. This is not an issue *per se* when dealing with processes, but is an issue with events. Suppose not-do_e_j is executing, with invariants $\neg C_j$ – the negation of the condition C_j on event e_j. Then, suppose $e_{j\dashv}$ is applied as step k – we would want to synchronise this with ending and restarting not-do, as shown in Figure 1. But:

1. Due to not-do_$e_{j\dashv}$, we would say that V_k must satisfy $\neg C$: the invariants of not-do_e_j;
2. Due to e_j itself, V_k must also satisfy C, i.e. *pre* e_j;
3. Due to not-do_$e_{j\vdash}$, V'_k must then satisfy $\neg C$.

The first two of these are mutually exclusive: they require V_k to satisfy $C \wedge \neg C$. Thus, we tweak the constraints, creating variables ϵ prior to step k – denoted $V_k^{-\epsilon}$ – and apply the constraints of not-do_$e_{j\dashv}$ to $V_k^{-\epsilon}$ rather than V_k.

The constraints to give the values of $V_k^{-\epsilon}$ can be calculated *almost* in the same way as those for V_k, with a slight modification to calculate the values of variables ϵ in the past rather than now. With a small substitution into Equation 1 we get:

$$v_k^{-\epsilon} = v'_i + \delta v'_i((t(k) - \epsilon) - t(i)) \quad (2)$$

...where step i was the last step to have an effect on v. This is correct unless $t(k)$=$t(i)$+ϵ, in which case:

$$v_k^{-\epsilon} = v_i \quad (3)$$

Or, in other words, $v_k^{-\epsilon}$ takes the value of v immediately before the last effect on v (i.e. step i). To capture this choice over $v_k^{-\epsilon}$, each invariant of not-do_$e_{j\dashv}$ referring to v is replaced with a disjunction of two conditions:

- The invariant with $v_k^{-\epsilon}$ from Equation 2 $\wedge\, t(k) > t(i)$+ϵ;
- The invariant with $v_k^{-\epsilon}$ from Equation 3 $\wedge\, t(k)$=$t(i)$+ϵ;

Invariants referring to n variables are replaced with a disjoint of 2^n options: one for each combination of choosing some $v_k^{-\epsilon}$, and enforcing the appropriate temporal side-constraint.

So far, the focus has been on the general case: choosing to apply an event or to switch the running state of a process. In some cases such choices are inevitable, and for efficiency, we can exploit this. For instance, suppose run_p_i is executing, and has an invariant $battery \geq 8$, and an action assigns $battery$=0 – that invariant is immediately broken. In this case, we need not branch over what to apply next in the state reached: the only sensible thing to do is to force the (synchronised) application of run_$p_{i\dashv}$ and not-run_$p_{i\vdash}$, to change the process from running to not-running, or vice versa as appropriate. Similarly, if the invariant was attached to not-do_e_j, and is now broken, we can force the application of not-do_$e_{i\dashv}$, e_j, not-do_$e_{j\vdash}$. This eliminates the increase in branching factor due to process/event steps in such cases.

As a final note, we need to consider what happens at the start and end of the plan. In the compilation (Section 4.1), this was achieved with clips. Here, it is far easier:

- In the initial state I, for each p_i, if its condition C_i is satisfied, apply run_$p_{i\vdash}$; otherwise, apply not-run_$p_{i\vdash}$. In both cases, fix the time of this step to zero.
- In the initial state I, for each e_j, its condition C_j is false (c.f. PDDL semantics). Thus, apply not-do_$e_{j\vdash}$, at t=0.
- A state satisfies goals G, if an action with precondition G could be applied, and if there are no currently executing actions other than run, not-run or do actions.

Domain	Ver	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
#S/#G		1/3	1/5	2/5	2/8	3/8	3/7	4/9	4/11	4/12	4/13	4/14	4/15	4/16	4/17	5/3	5/6	5/9	6/4	7/4	8/4
Satellite	P/E	0.30	1.42	1.39	2.89	8.54	3.40	247.58	62.85	34.91	34.66	37.00	41.82	42.89	74.83	36.71	471.23	42.93	25.64	27.27	364.96
Satellite	comp	-	9.33	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Transformer	P/E	0.01	0.02	0.29	0.30	1.19	0.29	4.67	0.27	13.27	1.5	43.30	5.91	598.96	17.91	-	61.68	-	395.59	-	1465.89
Transformer	comp	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
LSFRP	P/E	0.03	0.04	2.33	5.32	5.58	2.88	5.58	5.58	5.25	5.74	5.52	x	x	x	x	x	x	x	x	x
LSFRP	comp	0.29	0.34	-	-	-	-	-	-	-	-	-	x	x	x	x	x	x	x	x	x

Table 3: Results of running the planner on PDDL+ Domains. P/E denotes 'Processes and Events'. comp denotes 'Compiled'. '-' indicates that the problem was unsolved, 'x' marks problems that do not exist.

Note that the initial state checks do not require recourse to the MIP, as variables (and hence the truth values of conditions) hold definite values, namely those of the initial state: there is initially no active continuous numeric change.

7 Evaluation

In this section we empirically demonstrate the performance of our implemented planner on PDDL+ domains. In domains without processes and events our planner will perform exactly as the planner POPF, runner up in the IPC2011 temporal track. Thus, we refer the reader to the published results for that planner (Coles et al. 2010) and the results of IPC2011 (Jimenez and Linares-Lopez 2011) for details of its performance on these domains, and comparisons to other existing planners. Unfortunately we are unable to compare the performance of our planner on PDDL+ domains with that of any other existing planner as TM-LPSAT, the only other fully-automated PDDL planner to support these features, is not available in a runnable form. As a guide to the reader, however, the limited published results for TM-LPSAT on available benchmarks (Shin 2004) report that the best configuration solves IPC2000 Driverlog Numeric Problems 2,3 and 4 in 149.82, 29.28 and 139.97 seconds respectively; whereas our planner solves these instances in 0.16, 0.01 and 0.05 seconds (albeit on slightly different hardware).

As a baseline for comparison we therefore use POPF, reasoning with the 'efficient' version of the clip compilation described in Section 4.1. POPF (and its siblings) are the only currently available systems for solving such problems, even when compiled. As there are no available standard PDDL+ problem sets we have created three domains and problem sets of our own based on those in the existing literature. Table 3 shows the time taken to solve these problems with processes and events (P/E) and using the compilation (comp).

The first of these is the cooled satellite domain described in (Coles et al. 2012). Originally based on the IPC2000 satellite domain, the cooled version allows active cooling of imaging sensors to be used to reduce the exposure time needed; at the expense of increased power demands. In our version of this domain, sunrise/sunset are processes, with preconditions on the time elapsed thus far, and that increase and decrease the (solar) power available to satellites. The results for this domain show that the compilation scales very poorly, indeed the planner using this solves only 1 problem. The #S/#G row shows the number of satellites and goals in each of the problem files: the P/E configuration scales well and to much larger problems than the compilation.

Our second domain is the transformer domain described in (Bell et al. 2009). This domain lends itself naturally to processes and events: the voltage on the circuit changes throughout the day due to customer demand. Our encoding uses processes based on the current time to update the voltage linearly over each half-hour period, using a piecewise-linear demand curve: an improvement on the original discrete model. The goal in this problem is to keep the voltage within a specified range, until a specified time, by switching transformers and capacitors in response to demand. We model the voltage going out of range using an event (one for each of bound) with the precondition $voltage > max$ (or $< min$), that deletes a fact required by the goal. From the table we see that the compilation leads to poor performance – no problems are solved – whilst the P/E configuration performs well. For guidance, within the problem set, even-numbered problems model winter demand, whilst odd problems model summer demand. Also, problem n+2 has one additional half-hour period of demand change compared to problem n. Performance on the winter configuration does not scale quite as far as summer, as more control actions are needed in winter to keep the voltage in range.

Finally, we consider the LSFRP domain (Tierney et al. 2012), based on the movement of ocean liners from one shipping service to another, around the world. The key aspect of this domain with respect to processes is the 'hotel-cost' of a vessel, paid from when it leaves one service until it reaches another. There may be several actions between these two points and the ship might have to wait to join its destination service at an appropriate point in the timetable. Further, if 'sail-on-service' actions are used, available on certain routes at certain times, hotel cost is not payable for the duration of these actions. The most natural model of this cost is as a process, which starts when the ship leaves its initial service; and stops when it either joins its destination service, or while sailing-on-service. Whilst the compilation successfully solves the 2 smallest problems in this domain it quickly becomes unable to scale. The P/E configuration solves all 11 problems in the suite – a set of real-world-sized problems developed working with industry – in under 6 seconds, although it is not attempting to optimise quality.

In conclusion, we have shown that direct handling of processes and events can lead to a significant advantage in terms of scalability in solving problems in which exogenous happenings are involved. Since the compilation solved so few problems it is difficult to make conclusions about solution quality, but on the 3 problems that were mutually solved, 2 had identical quality solutions, and in the other, the P/E configuration found the better solution. In future work we intend to consider optimising the quality of plans in the presence of processes and events and to extend our reasoning to consider a wider class of continuous functions.

References

Bell, K. R. W.; Coles, A. J.; Coles, A. I.; Fox, M.; and Long, D. 2009. The role of AI planning as a decision support tool in power substation management. *AI Communications* 22(1):37–57.

Boddy, M. S. 2003. Imperfect match: PDDL 2.1 and real applications. *Journal of Artificial Intelligence Research* 20:61–124.

Coles, A. J.; Coles, A. I.; Fox, M.; and Long, D. 2010. Forward-Chaining Partial-Order Planning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*.

Coles, A. J.; Coles, A. I.; Fox, M.; and Long, D. 2012. COLIN: Planning with Continuous Linear Numeric Change. *Journal of Artificial Intelligence Research* 44:1–96.

Fox, M., and Long, D. 2003. PDDL2.1: An extension of PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research* 20:61–124.

Fox, M., and Long, D. 2006. Modelling mixed discrete continuous domains for planning. *Journal of Artificial Intelligence Research* 27:235–297.

Fox, M.; Long, D.; and Halsey, K. 2004. An Investigation into the Expressive Power of PDDL2.1. In *Proceedings of the European Conference of Artificial Intelligence (ECAI)*.

Hoffmann, J., and Edelkamp, S. 2005. The Deterministic Part of IPC-4: An Overview. *Journal of Artificial Intelligence Research* 24:519–579.

Jimenez, S., and Linares-Lopez, C. 2011. IPC-2011 results. `http://www.plg.inf.uc3m.es/ipc2011-deterministic/Results`.

Li, H., and Williams, B. 2011. Hybrid planning with temporally extended goals for sustainable ocean observing. In *Proceedings of AAAI*.

McDermott, D. 2003a. PDDL2.1-The Art of the Possible? Commentary on Fox and Long. *Journal of Artificial Intelligence Research* 20:61–124.

McDermott, D. 2003b. Reasoning about Autonomous Processes in an Estimated Regression Planner. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*.

Penberthy, S., and Weld, D. 1994. Temporal Planning with Continuous Change. In *Proceedings of AAAI*.

Penna, G. D.; Intrigila, B.; Magazzeni, D.; and Mercorio, F. 2009. UPMurphi: a tool for universal planning on PDDL+ problems. In *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS)*.

Shin, J., and Davis, E. 2005. Processes and Continuous Change in a SAT-based Planner. *Artificial Intelligence* 166:194–253.

Shin, J. 2004. *TM-LPSAT: Encoding Temporal Metric Planning in Continuous Time*. Ph.D. Dissertation, New York University.

Tierney, K.; Coles, A. J.; Coles, A. I.; Kroer, C.; Britt, A.; and Jensen., R. M. 2012. Automated planning for liner shipping fleet repositioning. In *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS)*.