The 26th International Conference on Automated Planning and Scheduling

Proceedings of the 10th

Scheduling and Planning Applications woRKshop (SPARK)

Edited by:

Sara Bernardini, Steve Chien, Shirin Sohrabi, Simon Parkinson

London, UK, 13-14/06/2016

Organising Committee

Sara Bernardini, Royal Holloway University of London, UK.

Steve Chien, NASA Jet Propulsion Laboratory, USA

Shirin Sohrabi, IBM, USA

Simon Parkinson, University of Huddersfield, UK

Program Committee

Bram Ridder, King's College London

Riccardo De Benedictis, CNR - National Research Council o Italy

Riccardo Rasconi, ISTC-CNR

Mark Johnston, JPL/California Inst. of Technology

Minh Do, NASA Ames Research Center

Simone Fratini, European Space Agency - ESA/ESOC

Adrien Maillard, ONERA

Angelo Oddi, ISTC-CNR, Italian National Research Council

Tiago Stegun Vaquero, MIT and Caltech

Alexandre Albore, Onera & INRA

Kartik Talamadupula, IBM Research, USA

Christophe Guettier, SAGEM

Ramiro Varela, University of Oviedo

Patrik Haslum, ANU

Nicola Policella, ESA/ESOC

Bryan O'gorman, NASA Ames Research Center

Foreword

Application domains that entail planning and scheduling (P&S) problems present a set of compelling challenges to the AI planning and scheduling community that from modeling to technological to institutional issues. New real-world domains and problems are becoming more and more frequently affordable challenges for AI. The international Scheduling and Planning Applications woRKshop (SPARK) was established to foster the practical application of advances made in the AI P&S community. Building on antecedent events, SPARK'16 is the tenth edition of a workshop series designed to provide a stable, long-term forum where researchers and practitioners can discuss the applications of planning and scheduling techniques to real-world problems. The series webpage is at http://decsai.ugr.es/~lcv/SPARK/ We are once more very pleased to continue the tradition of representing more applied aspects of the planning and scheduling community and to perhaps present a pipeline that will enable increased representation of applied papers in the main ICAPS conference. We thank the Program Committee for their commitment in reviewing. We thank the ICAPS'16 workshop and publication chairs for their support.

The SPARK’16 Organizers

Table of Contents

Planning Autonomous Underwater Reconnaissance Operations*

Sara Bernardini
Department of Computer Science
Royal Holloway, University of London
Egham, Surrey, UK, TW20 0EX
sara.bernardini@rhul.ac.uk

Maria Fox and **Derek Long** and **Bram Ridder**
Department of Informatics
King's College London
London, UK, WC2R 2LS
firstname.lastname@kcl.ac.uk

Abstract

In this paper, we focus on the task of reconnaissance in an underwater setting. The objective is to efficiently search a large area in order to identify safe passages for ships. A path is considered safe if there are no rocky regions within it as these can conceal hazards such as mines. Our approach is to decompose the area into regions which can then be allocated for seach to individual vehicles. An initial, coarse, lawnmower survey gives some indication of the make-up of the area to be searched, and the next step is to use this survey to hypothesise about the structure of the unseen areas. It is hypothesised that any as-yet unseen areas are rocky and the task of the AUVs is to disprove this hypothesis by planning efficient paths to investigate these unseen areas. We have developed a plan-based approach that searches a large area very efficiently by reasoning about the structure of the seabed as the planned search progresses. To demonstrate the potential of our technique, we have developed a simulation based on ROS and integrated our system into the commercial "SeeByte Neptune" simulator.

1 Underwater Reconnaissance Problem

The current state of the art is that underwater operations are typically scripted or remotely operated. The successful proprietary SeeByte Neptune system (SeeByte 2013), for example, offers a way to support human planning of underwater missions at a level of abstraction, but remains demanding of human planners and offers limited opportunities for dynamic replanning on-board as missions unfold. There is a limited application of automated planning in multi-domain unmanned platforms, see for example (Cashmore et al. 2013) and (Faria et al. 2014). The first paper concerns the use of autonomous AUVs to perform inspection and maintenance tasks in underwater installations, while the second paper explores the role of planning technologies in underwater missions and considers broader-based missions with mixed platforms including UAVs and AUVs.

In this paper, we describe our approach to use generic planning technology to underpin fully autonomous reconnaissance missions in an underwater setting. In particular, we consider an underwater area A, which is initially completely unknown and where mines might be present. The seabed in this area can be: flat (F), rippled (R), mixed (M) and complex (C). We assume that the AUVs available for the reconnaissance missions are capable of detecting these features with some probability error, which is given for each feature. When a region in A is directly observed as flat is classified as F1, if a region is adjacent to an F1 region is classified as F2. The same holds for the other types of seabed. The probability of spotting a mine is different depending on the seabed quality and these probabilities are given. Flat regions offer the best visibility. Let us know call *corridor* a passage in the area A from two given points. The goal of a reconnaissance mission is to find a *safe* corridor, i.e. a passage with the lowest probability of having mines in it (see Figure 1). The ideal outcome would be to find a corridor where the seabed is flat and no mines are spotted. Mission time needs to be minimised so, although multiple observations of the same locations are possible, the information that is acquired in doing that must be balanced against the time used for those additional observations.

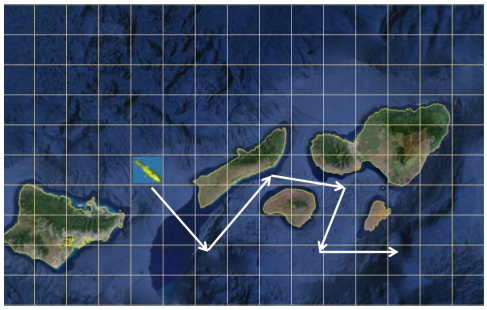


Figure 1: Underwater Reconnaissance Problem: finding a safe passage from two given locations.

Currently, in real applications (SeeByte 2013), the problem is solved as follows. First, the AUV proceeds along one diagonal d_1 in the quadrangular area A and, in doing so, classifies n quads that can be identified within A and whose diagonals overlap d_1. The classes are: F1, F2, R1, R2, M1, M2, C1 and C2. Then, the AUV does the same along one side of A and along the other diagonal d_2 (see Figure 2). This procedure is repeated until the entire area A has been classified. After classification, a corridor is identified. This method works, but is inefficient since it requires that the en-

*The authors gratefully acknowledge support from ASUR, MOD Contract Number: DSTLX-1000079686.

tire area is classified in order to find a corridor, although this is not a requirement of the reconnaissance problem itself.

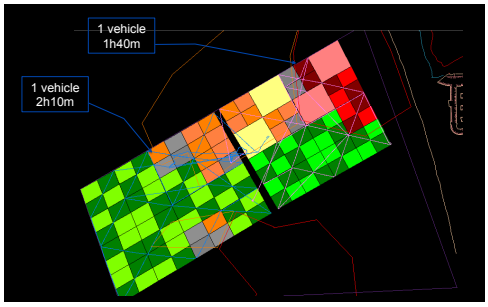


Figure 2: Classification of the underwater area to find a safe corridor. This solution is currently adopted in real-world operations, for example by SeeByte Inc.

We propose a new approach to underwater reconnaissance that is based on the use of *automated planning* techniques. Our approach allows one to minimise the resources used by the AUV during the inspection tasks since it does not require a complete classification of the seabed in the area of operation in order to find a passage. At the same time, planning tools provide solutions that maximise the likelihood of discovering safe passages.

As a first step towards solving the general underwater reconnaissance problem, we have focused on one particular instance, which has been devised in consultation with our industrial collaborators at SeeByte Inc. We consider an underwater quadrangular area Q (around 2 Km x 2 Km), which is initially completely unknown. When inspected by the AUV, regions in this area can be classified as flat (F) or non-flat. We call complex (C) any region that is non-flat. We call *safe* corridor (or passage) in the area Q, a sequence of segments that connects the left side to the right side, where each segment traverses flat regions only.

1.1 Search as a Planning Problem

The approach we adopt in tackling underwater reconnaissance is based on our previous work on search-and-track (i.e. the problem of finding and following a mobile target) using an unmanned aerial vehicle (UAV) (Bernardini et al. 2013; Bernardini, Fox, and Long 2014; 2015; Bernardini et al. 2016). That work exploits generic technology: modelling tools, a temporal planner and an execution architecture, which have been redeployed in the underwater environment. There is no domain-specific behaviour in either application, but the work presented here for the reconnaissance problem changes the focus of search from a single mobile target to finding a structured static area on the seabed. This change has led to interesting further developments in our ideas, although still using the same underlying paradigm.

The paradigm we use in all of these application areas relies on a *hypothesise-and-test* approach. In search, the hard decisions have to be made when considering what to do when the target of search is *not* found: once the target is found, then behaviour changes (to a following behaviour in search-and-track missions, or to an exploitation phase when searching for safe passage for ships). This leads to the observation that one can usefully plan those hard decisions in advance, to optimise search, based on the *hypothesis* that each search attempt will fail to find the target. That is, by *anticipating* failure in each search, we can focus on the decisions about what to do next in order to continue the search. Therefore, we plan searches and hypothesise that they will fail to discover the target of search, making it useful to plan to continue to search further.

Although we use a *deterministic* planner to perform the planning, we do not ignore probability. We assume that the actions of the searching agent are predictable, but that the outcome of search is not. However, as noted above, a successful search will cause a change in behaviour, so we can plan purely on the assumption that each search will fail. In that case, we can use this anticipated failure to determine how the probability mass describing the expected outcome of searches moves around (exploiting correlations in the properties of locations). This is reflected in changing rewards for different locations and allows the planner to trade between resource efficiency and expected benefits in its direction of search.

1.2 Assumptions

In this work we make several assumptions. Some of these are simplifying assumptions but others reflect our limited knowledge in some specialised areas related to the domain. In general, these assumptions are not fundamental to the way we solve the problem, because we can extend and modify our models to reflect a refined understanding of the domain without changing the underlying technology. This is a key benefit in the use of our generic technology.

We have ignored the problem of localisation accuracy in the AUVs, which simplifies the problem of navigation when there are multiple vehicles in the area. We have ignored the issue of use of sonar in locations where there might be wildlife that prevents us from using sonar safely. We have also ignored the question of the accuracy of the identification of subsea terrain structures based on sonar data. We have also ignored the potential uncertainty in navigation caused by currents.

2 Three Phase Approach

We adopt a three phase approach to tackle the underwater reconnaissance problem. First, the vehicle performs a coarse survey of the area, which is simply a big *lawnmower*(that is, the vehicle passes backward and forward over the search area) search pattern to acquire initial, coarse-grained information about the area of operation. This pattern is not a classic lawnmower pattern (which comprises tightly adjacent paths to cover the entire region), but a simple crossing path to give an initial view of the space (see Figure 4). Then, a cycle of *planning* and *replanning* rounds generate plans for the AUV to efficiently explore the area of operation in more depth. During this cycle, the vehicle discards as many paths that contain complex surfaces as possible and accumulates information about flat areas. When it has acquired sufficient

confidence that several flat regions can be linked together in a clear path, the AUV switches to the last phase, in which it goes along this hypothetical path to *confirm* that it is a safe passage. If this confirmation phase is successful, the problem is solved; if not, the planning and replanning cycle starts again until a clear path is found. The three phase approach is illustrated in Figure 3.

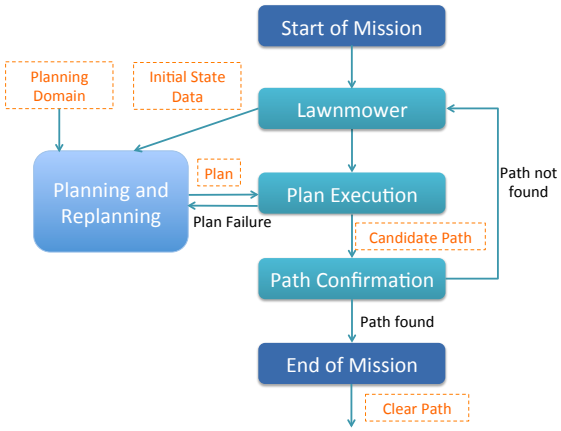


Figure 3: Three Phase Approach to the Underwater Reconnaissance Problem

3 Phase 1: Lawnmower

During the first phase, the AUV executes the initial survey pattern with three vertical legs and two horizontal legs over the area of operation Q (see Figure 4). In order to structure and store the information regarding the area of operation, we lay a grid over the area Q, where the cells have sides of 100 meters (see Figure 4). Since the AUV uses one sonar per side with a range of 50 m, we assume that the AUV is capable of observing an entire cell when it passes over it and can classify the seabed of the observed cell as flat or complex. Hence, once the lawnmower has been executed, the nature of all the cells traversed by the vehicle are known, whereas the other remain unknown (see Figure 5).

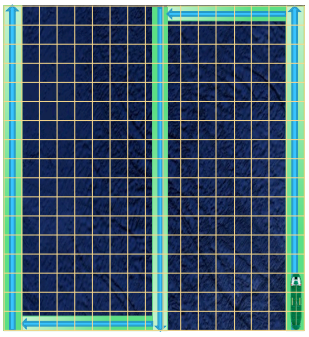


Figure 4: Grid over the area of operation and initial lawnmower (Phase 1).

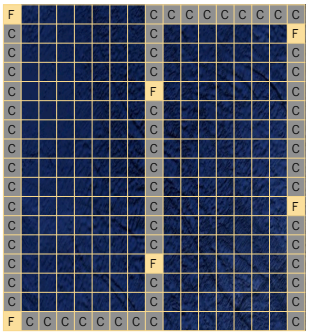


Figure 5: Initial information

4 Phase 2: Planning and Replanning Cycle

The planning and replanning cycle works in four steps:

1. Hypothesis generation;
2. Planning task formulation and planning;
3. Plan execution;
4. Replanning.

During the first step, based on the information collected during Phase 1 (see flat cells in Figure 5), we hypothesise potential clear segments in the area of operation by connecting together flat cells (see red arrows in Figure 6). These segments, appropriately combined, could form a clear passage from one side to the other side of the area of operation. Then, we ask the planner to check these hypotheses and discard the segments that contain complex surfaces since they are not suitable components of clear paths. In order to do that, we formulate the problem of disproving hypotheses as a planning task and we feed this task into an high-performing planner, called POPF-TIF (Piacentini et al. 2015). We then execute the plan provided by the planner until completion or until a plan failure occurs. In this context, a plan failure happens when the planner cannot discard a segment since, when checking it, it has not found complex cells in it. At this point, we abandon the current plan and generate a new planning problem based on the new information that we have acquired during the previous plan execution step. This replanning step provides a new plan that is in turn executed. This cycle of planning and replanning continues until we conclude that no passage exits or until we have gained sufficient information about a potential clear path, which is then validated in Phase 3. We will now describe all these four steps in more detail.

4.1 Hypothesis Generation

Considering a grid of $n \times n$, the initial lawnmower identifies flat cells in column 0, $n/2$, and $n-1$. Let us call these three sets of cells $\mathcal{C}_0$, $\mathcal{C}_1$ and $\mathcal{C}_2$. We generate hypothetical clear segments by connecting every flat cell in $\mathcal{C}_0$ with every flat cell in $\mathcal{C}_1$ and then the same for cells in $\mathcal{C}_1$ and $\mathcal{C}_2$ (see red arrows in Figure 6). We refer to these segments as our *hypotheses*.

Given these hypotheses with their corresponding initial and final cells, we also store information about other cells in

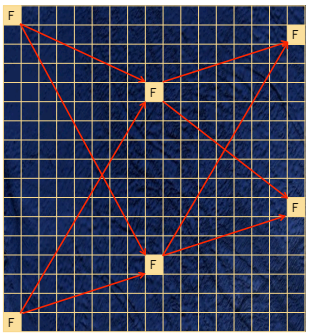


Figure 6: Hypothesis Generation

the grid (shown in orange colour in Figure 7), which will be useful to validate or discard hypotheses in the next steps of the procedure. In particular, for each hypothesis h_i, we identify the cells that contain the intersection points between h_i and each other hypothesis h_j (see, for example, cells 7 and 15 in Figure 7) and the cells that contain the middle points between the initial or final cell of h_i and the intersection cell (see, for example, cells 3, 4, 8 and 9 in Figure 7). Finally, if the hypothesis h_i does not intersect other hypotheses, we take the cell that contains its middle point (see, for example, cells 5 and 6 in Figure 7). The intersection and middle cells are interesting because, if the AUV inspects one of such cells and finds complex seabed in it, the hypotheses that pass through that cell can be discarded since they cannot represent valid components of clear passages. In what follows, we identify each cell with its corresponding central point and use the terms *waypoints* and cells interchangeably.

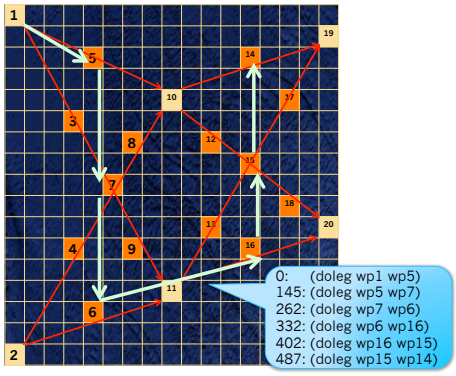


Figure 7: The waypoints of interest (in orange) and the initial plan (green segments and blue box).

4.2 Formulation of Planning Tasks

We use the language PDDL2.2 (Planning Domain Definition Language) (Edelkamp and Hoffmann 2004; Fox and Long 2003) to model the reconnaissance problem, taking advantage of several sophisticated features of this language to express all the properties of our domains.

We model all the relevant waypoints in our problem and their distances. As discussed in Section 3, they are the known flat waypoints identified during the lawnmower, the intersection points between hypotheses and the middle points of the hypotheses. We then associate a *reward* with each waypoint that represents an estimate of how many hypotheses can be disproved if that waypoint is found to be complex. These rewards guide the planner to explore the regions where the highest number of candidate paths can be discarded. By iteratively discarding more and more hypotheses, the search can then be directed towards the most promising regions where a clear passage can be found. The rewards are updated over time so that no hypothesis is checked more than once. A total reward keeps track of the accumulated capacity of the plan to be effective in discarding hypotheses.

The basic structure of the domain for the reconnaissance problem is simple: there is a navigation action that allows the vehicle to move from one waypoint to another. It has an entry waypoint and an exit waypoint. The effect, other than to move the vehicle from the entry to the exit point, is to increase the total reward by a quantity that represents the specific reward associated with the exit point discounted by the distance of such point from the entry point. This way of calculating the total reward allows us to maximise the opportunities of discarding unsafe hypotheses and also to minimise the total distance traversed by the AUV. The actions are durative and their duration is fixed in the problem instance to be the correct (computed) value for the traversal of the distance from the entry to the exit point.

Figure 8 displays the definition of the reconnaissance problem in PDDL2.2.

4.3 Planning Mechanism and Plans

We use POPF-TIF (Piacentini et al. 2015) to build plans for the observer. POPF-TIF is an advanced version of the planner OPTIC (Benton, Coles, and Coles 2012) (Optimizing Preferences and TIme-dependent Costs) and POPF (Coles et al. 2010), that allows us to handle complex temporal interactions. POPF-TIF is a temporal planner for use in problems where the cost function is not directly linked to the plan make-span, as it usually happens, but can be expressed as a continuous function. It combines grounded forward search with linear programming to handle continuous linear numeric change. POPF-TIF performs anytime, cost-improving search: it finds a first solution very quickly, since the empty plan is already a feasible solution, but it then spends the additional time improving on this solution by adding further manoeuvres to the plan or by trying different collections of manoeuvres. The search uses a weighted-$A^\star$ scheme with steadily changing weights in a tiered fashion. The plans produced are monotonically improving, so the final plan is selected for execution. We use a time-bounded search limited to 10 seconds because we are in a time-critical situation (this value is a configurable parameter). We use POPF-TIF because it is very fast at producing its first solution and provides an any-time improvement behaviour.

Figure 7 shows an example of a plan generated by POPF-TIFs for the single vehicle reconnaissance domain. Plans for this domain look like sequences of waypoints to visit. In particular, the planner chooses waypoint sequences that

```
(define (domain recon)
  (:requirements :typing :durative-actions :fluents :timed-initial-literals :conditional-effects)
  (:types waypoint hypothesis)
  (:predicates (at ?p - waypoint) (notVisited ?p - waypoint))
  (:functions (reward) (rewardof ?w - waypoint) (distance ?p1 ?p2 - waypoint) (mf ?w1 ?w2 - waypoint) )

  (:durative-action doLeg
   :parameters (?from ?to - waypoint)
   :duration (= ?duration (distance ?from ?to))
   :condition (and (at start (at ?from)) (over all (not (= ?from ?to))) (at start (notVisited ?to)))
   :effect (and (at end (at ?to)) (at start (not (at ?from))) (at start (not (notVisited ?to)))
    (at end (increase (reward) ( - (rewardof ?to) (distance ?from ?to))))
    (forall (?w - waypoint) (at end (assign (rewardof ?w) (* (rewardof ?w) (mf ?to ?w))))))
  )
)
```

Figure 8: Navigation action in PDDL2.2

maximise the opportunity to discard unsafe passages and, at the same time, minimise the use of resources. In Figure 7, for example, the planner chooses the path $wp_1 \rightarrow wp_5 \rightarrow wp_7 \rightarrow wp_6 \rightarrow wp_{16} \rightarrow wp_{15} \rightarrow wp_{14}$, which is the optimal route to inspect all the hypotheses while minimising the traversed distance. We call this plan π_1.

4.4 Plan Failure and Replanning

Plans are generated under the assumption that every waypoint that is checked by the AUV is found to be complex and that the corresponding hypotheses can be consequently discarded. However, there might be cases in which the AUV visits a waypoint to find that it is characterised by a flat surface. When this happens, the plan ceases to be valid and is abandoned. At this point, replanning is triggered. A new problem is formulated based on the information acquired during Phase 1, as before, and also during the execution of the plan until its failure. In particular, a new flat waypoint is added to the problem, which is the waypoint that has provoked the failure of the previous plan, and its corresponding new hypotheses. In addition, hypotheses relating to waypoints that have been found complex are removed. To continue with our example, let us assume that during the execution of plan π_1 the AUV finds that the cell corresponding to wp_7 has a flat surface (see Figure 9). At this point, the plan π_1 is abandoned and a new plan π_2 is generated. As it can be seen in Figure 9, the hypothesis from wp_1 to wp_{10} and its corresponding middle point have been removed from the planning task formulation, while the flat wp_7 and its corresponding hypotheses and middle points have been added. It is interesting to notice that the new plan focuses on checking waypoints that are around wp_7 since this now looks like a promising area to find a clear passage. During the problem formulation, waypoints wp_3, wp_4, wp_8 and wp_9 receive an higher rewards than the other points since disproving hypotheses in a promising area is more important than disproving hypotheses in unknown areas. This is why the planner now prioritises these waypoints with respect to others.

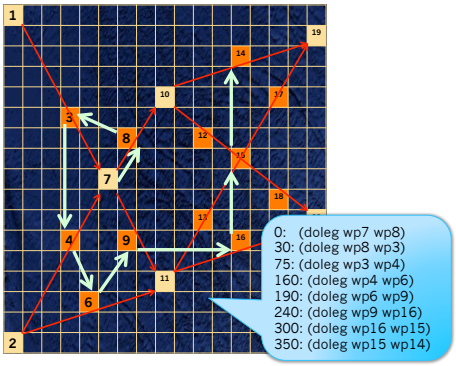


Figure 9: New plan after plan failure.

5 Phase 3: Path Confirmation

Both during Phase 1 and 2, the AUV accumulates information concerning flat cells in the area of operation. At regular intervals, the AUV checks whether the waypoints at the centre of these cells can be connected in such a way to create a path from left to right in the search area. If the AUV finds a potential path and is sufficiently confident that this path might be clear, which depends on how many cells in the path are known to be flat, the AUV exits the planning and replanning cycle and enters Phase 3, i.e. path confirmation. This phase simply consists in the AUV traversing the hypothetical clear path, which is composed of a sequence of segments that connect flat cells between each other. If the AUV confirms that a path has been found, it outputs such a path and the problem is solved. If not, it goes back to Phase 2 and a new task planning problem is formulated that encodes all the new information that has been acquired during the path confirmation phase.

The path confirmation phase is illustrated in Figures 10 and 11. Let us consider the figure on the left first and assume that the AUV already knows that waypoints $wp_1, wp_3, wp_7, wp_9, wp_{11}$ and wp_{20} are all flat. Let us also assume that the AUV now traverses the leg from wp_{11} to wp_{20} and verifies that wp_{16} is flat as well. At this point, starting from wp_{20}, the AUV traverses the path indicated by the green arrows in Figure 11 that goes through $wp_{20} \rightarrow wp_{16} \rightarrow wp_{11} \rightarrow wp_9 \rightarrow wp_7 \rightarrow wp_3 \rightarrow wp_1$. If we assume that the vehicle finds flat in all the cells along this path, then a clear passage is found and the algorithm exits.

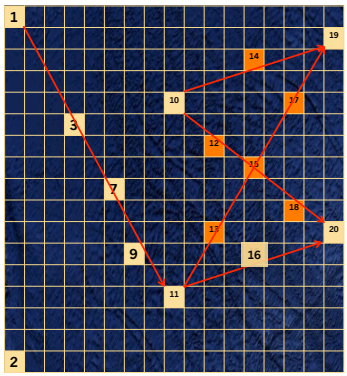


Figure 10: Phase 3: Path Confirmation

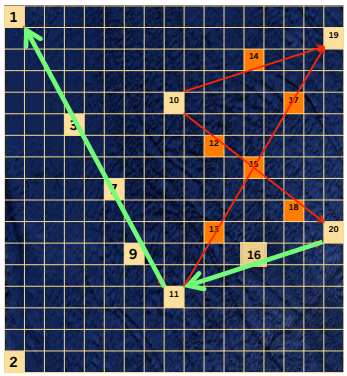


Figure 11: Clear path confirmed

6 Implementation

To demonstrate the viability of our approach to underwater reconnaissance, we have developed a simulation in consultation with our industrial collaborators at SeeByte. The simulation is intended to provide an appropriately abstracted view of the problem. The main abstraction is that we assume the control problem for the underwater vehicles solved. Our implementation can simulate the sea bed and multiple vehicles exploring it and performing reconnaissance tasks. In the next section, we will provide a detailed description of our simulator, while in Section 6.2, we will discuss our initial effort to integrate our plan-based approach to reconnaissance into the SeeByte's proprietary AUV simulator, called SeeTrack, and its relating planning system, called Neptune.

6.1 Architecture of the System

Our simulator, which is based on the DPE (Dreaded Portal Engine) 3D engine, is a cross-platform 3D engine that uses OpenGL 4.4. The AUV is modelled based on the "Girona 500" vehicle and the controller interface uses ROS to communicate with the vehicle. ROS, the Robot Operating System (Quigley et al. 2009), is a set of software libraries and tools used in building robotic systems, which has become increasingly popular both in industry and academia. Sensors, such as sonar and cameras, are simulated in the engine and use ROS topics, services, and actions to send and receive messages. This choice makes it very easy to decouple the simulator from the physical vehicle. Our 3D engine builds on a similar one, which we previously developed and successfully used in the context of the EU project PANDORA (Cashmore et al. 2013) to simulate a multi-AUV environment with sea life.

The general architecture of the system is shown in Figure 12. This is a two-layer architecture, in which the high-level layer (in blue) takes care of the deliberation capabilities of the system and provides an interface towards the bottom-level layer (in green), which corresponds to the control and execution capabilities of the system.

The high-level layer is completely generic and can be reused to embed a generic task planner in any ROS-based execution system. The planner node corresponds to an executable of our planner POPF-TIF (Piacentini et al. 2015)

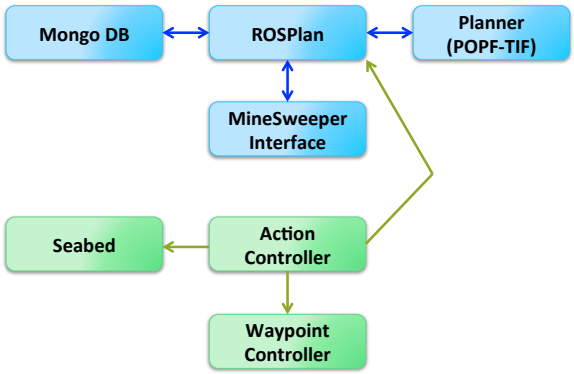


Figure 12: An high-level view of the architecture of our simulation for underwater reconnaissance missions.

(described in Section 4.3). Although we use POPF-TIF because of its high performance, any other PDDL planner can be embedded in this node. ROSPlan (Cashmore et al. 2015) is at the core of the high-level layer. ROSPlan is a framework for embedding a generic task planner in a ROS system. Together with the planner, ROSPlan is the crucial element of our architecture and provides tools to:

- automatically generate the initial state for the planner from the knowledge parsed from sensor data and stored in a knowledge base;
- automate calls to the planner, then post-process and validate the plan;
- handle the main dispatch loop, taking into account changing environment and action failure;
- match planned actions to ROS action messages for lower level controllers.

Figure 13 presents a general overview of the ROSPlan framework (red box), consisting of the Knowledge Base and Planning System ROS nodes. Sensor data is passed continuously to ROSPlan and is used to construct planning problem instances as well as to inform the dispatch of the plan (blue boxes). Actions are dispatched as ROS actions and executed by lower-level controllers (green box), which respond reactively to immediate events and provide feedback.

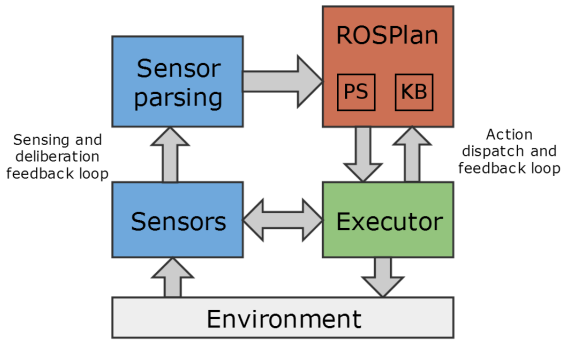


Figure 13: General overview of the ROSPlan framework.

Mongo DB implements the knowledge base of our system. It is a collection of interfaces and is intended to collate the up-to-date model of the environment. The knowledge base is updated as soon as new information becomes available and is used by the planning system to generate the PDDL problem instance by supplying an initial state and goals. This is done through an interface comprised of ROS services.

Let us now consider the bottom layer. The green modules and the MineSweeper module are application specific and take care of simulating the seabed and one or multiple AUVs exploring such a seabed in the context of reconnaissance missions. The Seabed node represents the model of the seabed and stores the knowledge that the AUVs acquire during their explorations. The model is a grid and, for each cell, we record whether the cell is flat or complex and if an AUV has observed it. As discussed in Sections 3 and 4, we use this model to create the initial state for the planner and update it based on the outcome of the exploration. During execution, the Action Controller node listens to the action dispatch topic of ROSPlan and translates each dispatched action to a command for a controller. It also monitors all the active controllers and provides feedback to ROSPlan. In our case, there is a single controller, the Waypoint Controller node. This controller receives 3D coordinates that are translated by the Action Controller and handles the movement of the AUVs toward the given coordinate during execution. The MineSweeper node sets up ROSPlan for our specific application.

Figures 14 and 15 concern a one vehicle mission and show the vehicle that performs the initial lawnmower and executes the plan initial respectively. In Figure 15, the plan is shown at the bottom and the path followed by the vehicle is highlighted in yellow.

6.2 Integration with SeeByte Neptune

SeeByte, our industrial collaborator, has provided us with details and specifications for modelling problems in the underwater domain and has given us access to their hardware and software products for integrating high-level planning behaviour in them. In particular, SeeByte has developed two main pieces of software:

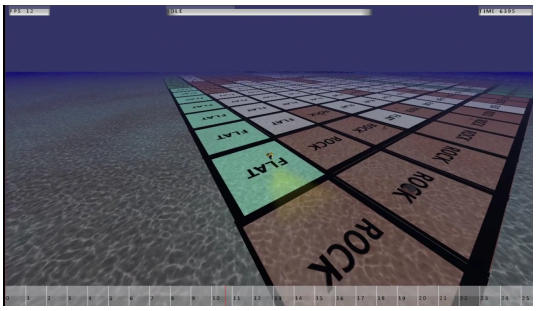


Figure 14: One vehicle performing a lawnmower.

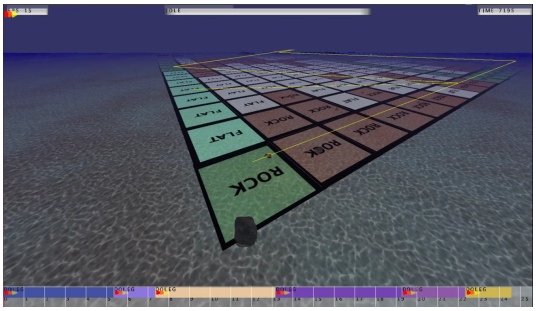


Figure 15: One vehicle executing the plan.

- SeeTrack: this is the overall infrastructure to directly control and monitor the AUVs. It has a graphical interface that allows the operator to send commands to the vehicle and to receive updates. It contains maps of the area of the operation, one window for each AUV with many parameters to set and observe.
- Neptune: this is the software in charge of autonomy. It has a graphical interface, more sophisticated than SeeTrack, through which a human operator can plan a high-level mission. Planning a mission means specifying the number of vehicles available, the area that needs to be surveyed, the targets that need to be inspected, what polygons to use around the targets to analyse them and what exclusion zones are present in the area of operation, which are regions that the AUVs cannot enter. Upon the description of a mission, Neptune creates a sequence of actions to execute the mission and push it to the AUVs. Neptune supports a limited form of autonomy based on the human operator's input and is domain-dependent (Patrón, Lane, and Petillot 2009).

SeeByte has also built a physical AUV simulator. It can simulate up to two unmanned maritime platforms (surface and underwater), sensors detections, communication links, environmental injects, time clocks and temporal events. It also supports combination of real and virtual platforms and sensors. The core Neptune modules are the same as run on real assets. The simulator is very useful as it allows the operator to experiment with a mission in simulation before running the same mission file on real assets.

The long-term goal of our work is to integrate our plan-

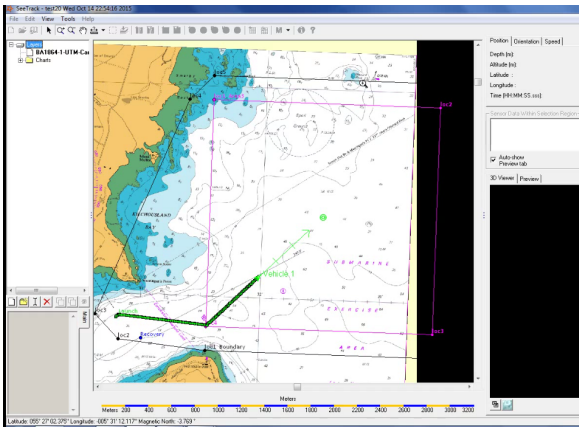


Figure 16: Screenshot of SeeTrack with a vehicle performing a reconnaissance mission. Commands are sent from ROSPlan to the vehicle simulator directly.

based reconnaissance behaviour in the SeeByte AUV simulator, which involves linking ROSPlan with Neptune. The integration between Neptune and ROSPlan is facilitated by the use of ROS in both systems. So far, we have demonstrated that is possible to connect ROSPlan with the SeeByte vehicle simulator by directly dispatching actions from ROSPlan to the vehicle controllers (see screenshot in Figure 16). The next step is to create an interface between ROSPlan and Neptune. Although this is feasible, there are engineering challenges involved in this since both Neptune and ROSPlan have been built to handle the dispatch loop. In order to function together, a hierarchy of command needs to be established that will decide when one must cede to the other. This will require that the protocol between them for determining precedence and the points at which control is devolved must be defined, which will require technical input from SeeByte.

7 Next Steps

We have already extended our work to deal with reconnaissance missions that employ multiple heterogenous assets that need to cooperate with each other during the search for a path. A search plan is devised centrally, decomposed into the sub-plans for individual vehicles and distributed to them for execution. Assuming that these vehicles can only communicate on the surface, the plan contains a number of communication activities which require all AUVs to surface in the same time windows in order to share information about the developing path. However, poor communications can lead to plan failure during plan execution, which force replanning by individual AUVs. We have devised a method for resolving failures within the sub-plans in such a way that, when replanning, the single AUVs do not violate the planned commitments that they have already made with the other assets to cooperate and to communicate during the task at hand.

In future work, we intend to perform a broad experimentation of our plan-based approach, both in simulation and in real sea trials. We aim to complete the integration between our planning system with Neptune, with the goal of managing command-and-control loop for plan dispatch and hand-over between plan execution and replanning phases.

In addition, we will extend initial work on sea-bed models to allow exploitation of probabilistic initial state construction in the deterministic planning models and explore the problem of deployment on larger fleets of cooperating assets across multiple modes (UAVs, USVs and AUVs).

References

Benton, J.; Coles, A.; and Coles, A. 2012. Temporal Planning with Preferences and Time-Dependent Continuous Costs. In *Proceedings of the Twenty Second International Conference on Automated Planning and Scheduling (ICAPS-12)*.

Bernardini, S.; Fox, M.; Long, D.; and Bookless, J. 2013. Autonomous Search and Tracking via Temporal Planning. In *Proceedings of the 23st International Conference on Automated Planning and Scheduling (ICAPS-13)*.

Bernardini, S.; Fox, M.; Long, D.; and Piacentini, C. 2016. Leveraging Probabilistic Reasoning in Deterministic Planning for Large-Scale Autonomous Search-and-Tracking. In *Proceedings of the 26st International Conference on Automated Planning and Scheduling (ICAPS-16)*.

Bernardini, S.; Fox, M.; and Long, D. 2014. Planning the Behaviour of Low-Cost Quadcopters for Surveillance Missions. In *Proceedings of the 24st International Conference on Automated Planning and Scheduling (ICAPS-14)*.

Bernardini, S.; Fox, M.; and Long, D. 2015. Combining Temporal Planning with Probabilistic Reasoning for Autonomous Surveillance Missions. *Autonomous Robots* 1–23.

Cashmore, M.; Fox, M.; Larkworthy, T.; Long, D.; and Magazzeni, D. 2013. Planning inspection tasks for auvs. In *In Proceedings of OCEANS'13 MTS/IEEE*.

Cashmore, M.; Fox, M.; Long, D.; Magazzeni, D.; Ridder, B.; Carrera, A.; Palomeras, N.; Hurtos, N.; and Carreras, M. 2015. ROSPlan: Planning in the Robot Operating System. In *In Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS-15)*.

Coles, A. J.; Coles, A. I.; Fox, M.; and Long, D. 2010. Forward-Chaining Partial-Order Planning. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS-10)*.

Edelkamp, S., and Hoffmann, J. 2004. PDDL2.2: The language for the classical part of the 4th international planning competition. In *Proceedings of the 4th International Planning Competition (IPC-04)*.

Faria, M.; Pinto, J.; Py, F.; Fortuna, J.; Dias, H.; Martins, R.; Leira, F.; Johansen, T. A.; Sousa, J.; and Rajan, K. 2014. Coordinating uavs and auvs for oceanographic field experiments: Challenges and lessons learned. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, 6606–6611.

Fox, M., and Long, D. 2003. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research* 20.

Patrón, P.; Lane, D. M.; and Petillot, Y. R. 2009. Continuous mission plan adaptation for autonomous vehicles: balancing effort and reward. In *4th Workshop on Planning and Plan Execution for Real-World Systems, 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*, 50–57.

Piacentini, C.; Alimisis, V.; Fox, M.; and Long, D. 2015. An extension of metric temporal planning with application to AC voltage control. *Artificial Intelligence* 229:210–245.

Quigley, M.; Gerkey, B.; Conley, K.; Faust, J.; Foote, T.; Leibs, J.; Berger, E.; Wheeler, R.; and Ng, A. 2009. ROS: an open-source Robot Operating System. In *Proceedings of the International Conference on Robotics and Automation (IJCAI)*.

SeeByte. 2013. Seetrack neptune - user manual.

Evaluating Scientific Coverage Strategies for A Heterogeneous Fleet of Marine Assets Using a Predictive Model of Ocean Currents

Andrew Branch[1], Martina Troesch[1], Selina Chu[1], Steve Chien[1], Yi Chao[2], John Farrara[2], Andrew Thompson[3]

[1]Jet Propulsion Laboratory, California Institute of Technology

[2]Remote Sensing Solutions

[3]California Institute of Technology

Correspondence Author: steve.a.chien@jpl.nasa.gov

Abstract

Planning for marine asset deployments is a challenging task. Determining the location to where the assets will be deployed involves considerations of (1) location, extent, and evolution of the science phenomena being studied; (2) deployment logistics (distances and costs), and (3) ability of the available vehicles to acquire the measurements desired by science.

This paper describes the use of mission planning tools to evaluate science coverage capability for planned deployments. In this approach, designed coverage strategies are evaluated against ocean model data to see how they would perform in a range of locations. Feedback from these runs is then used to refine the coverage strategies to perform more robustly in the presence of a wider range of ocean current settings.

Introduction

Study of the ocean is of paramount importance in understanding the Earth's environment in which we live. Oceans cover the majority of the Earth's surface and play a dominant role in climate and the Earth's ecosystems.

Space based remote sensing provides great information about ocean dynamics. However, remote sensing information is generally limited to measuring the ocean surface or the upper layer of the ocean. Ocean models can further augment this information. However, in order to probe the immense volume of the ocean most accurately generally requires marine vehicles such as autonomous underwater vehicles (AUVs), Seagliders, profiling buoys, and surface vehicles sampling in-situ. Deploying and operating these assets is very expensive. This means there is a very limited number of marine vehicles compared to the massive size of the ocean. Knowing where the assets should be deployed and operated is very difficult. One strategy is to deploy in-situ assets to study specific scientific features such as fronts, eddies, upwellings, harmful algal blooms, or other features of interest. A typical strategy would be to deploy marine assets to measure transects across the feature of interest at a scale that covers the feature, as well as a baseline signal around the feature. However, asset capabilities (e.g. mobility, endurance) and prevailing ocean currents may render these science goals unachievable. Our project targets automatic generation of mission plans for assets to follow these science derived templates. This paper specifically describes the use of this planning technology to assess feasibility of achieving these science templates to support both: deployment design (number of assets, where, which templates to follow) and science template design (how to adjust designed templates to be feasible in settings where they are not likely to succeed in original form).

The remainder of this paper is organized as follows. First, we discuss the problem that we are trying to solve and the inputs to that problem that we use, including the predictive ocean model and the types of assets. Then we discuss the approach that we took to solve the problem and the results of that approach. Finally, we discuss what needs to be done next to continue to develop a solution to this problem.

Problem Definition

The goal of the path planning software is to develop a plan of control directives that when executed by a marine asset in an actual ocean current field will cause the marine asset to follow a template path relative to an ocean feature of science interest, where a template path is a series of edges between waypoints. An example of a template path can be seen in figure 4. In this case, the templates in the figure are going

from one corner of a 15km x 15km box to the opposite corner. Nominally this path should take 24 hours to complete. Ideally the asset would perfectly follow the line but in reality the asset should follow the line as closely as possible and achieve the endpoint within 0.5 km.

There are really two related problems that have differing inputs and outputs but use much of the same search-based algorithm. First, before an actual deployment, it is useful to assess a range of deployment locations and science template coverage strategies. Second, during an actual deployment, we have an actual set of asset locations and the goal is to develop asset directives using a current model that will follow the template directed paths in reality.

The Template Assessment and Feasibility Problem

The focus of this paper is the pre-deployment assessment of locations and science templates for feasibility. The inputs to this are: (1) a set of template paths, (2) a set of asset models, (3) a planning ocean current model, (4) a nature ocean current model, and (5) a set of evaluation locations. The first template waypoint in this path is the start location for the asset. The asset model determines how the asset will behave when simulating actions in a current model. The planning and nature current models specify ocean currents for x, y, z, and over the relevant proposed deployment domains. The planning and nature models are used to simulate the inaccuracies of an ocean model with respect to the actual ocean. The planner constructs a set of control actions for the asset that when executed in the lower fidelity model i.e., the planning model, should follow the desired science template. These control actions are then evaluated in the higher fidelity model i.e., the nature model, to simulate planning model inaccuracies. This process is repeated over a set of evaluation locations.

A few assumptions are made in this problem. First, we assume that the discrepancies between the planning and nature ocean models are similar to the inaccuracies present between a planning model and the actual ocean during a deployment. If this is not the case, then the results are not helpful when preparing for an actual operational deployment. We also assume a number of things about the asset, namely that the asset motion model is accurate. We also do not model hardware issues such communication failures, GPS failures, and navigation inaccuracies.

Operational Deployment

A second related problem is an actual deployment usage problem. In this case we are given a set of template paths asset models, asset locations, and a single ocean current model. The templates and asset models are defined in the same manner as before. In our current approach only one current model is used and we do not evaluate, predict, or model the inaccuracies of the predictive ocean models (see future work on ensemble modelling). The output produced by the planner will be a series of control actions in the form of directed waypoints called command points, which are distinct from the waypoints that make up the template path. The command points are then used by the assets to navigate.

Many of the same assumptions are made in this problem as with the previous one. We assume that the ocean model currents reflect the actual ocean currents. When running the planner, we still assume that there will be no future hardware failures and that the properties for the assets are accurate.

Ocean Model

Any cell-based, predictive model with information about ocean currents over multiple depths and an extended period of time could be used for the path planning. Some widely spread ocean models include the Harvard Ocean Prediction System (HOPS) (Robinson 1999), the Princeton Ocean Model (POM) (Mellor 2004), the Regional Ocean Modelling System (ROMS) (Li et al. 2006), and the Hybrid Coordinate Ocean Model (HYCOM) (Chassignet et al. 2007).

For our ocean model, we used the Regional Ocean Modelling System (ROMS) (Chao et al. 2009; Li el al. 2006; Farrara et al. 2015). The grid spacing used for our experiment was approximately 3km x 3km and had 14 depths ranging from 0 to 1000m in non-uniform levels. Data was available at 1 hour intervals for a 72-hour period.

As stated previously, two different ocean models are used to represent the inaccuracies in predictive ocean models. The ROMS model with the best representation is used as the ocean, this is referred to as the nature model. The second model that is used does 6 days of advanced prediction. This is referred to as the planning model. Fewer days of advanced predication mean a higher fidelity model and thus the planning model is closer to the nature model. The list of inputs used for the planning and nature model can be found in (Troesch et al. 2016b).

When simulating the movement of an asset, the closest grid point in the latitude, longitude, and depth dimensions is used. Whenever the asset crosses into the next depth dimension the latitude and longitude information is updated. The time used is the previous hour. For example, in the first hour of operation the information at time index 0 is used. No interpolation is done in any dimension.

Assets

Three different assets are used for this experiment, Seagliders, AUVs, and Wave Gliders. The Seagliders repeatedly profile between the surface and some depth, with a specific bearing (Eriksen et al. 2001). It is only during these profiles where they have any forward movement. If the ocean floor or an obstacle is reached before the profile depth, the asset

will abort that profile and start to ascend. When the Seaglider is at the surface it is able to update its location using GPS and communicate with the shore. This allows the asset to receive new commands. The dive profile can be seen in figure 1.

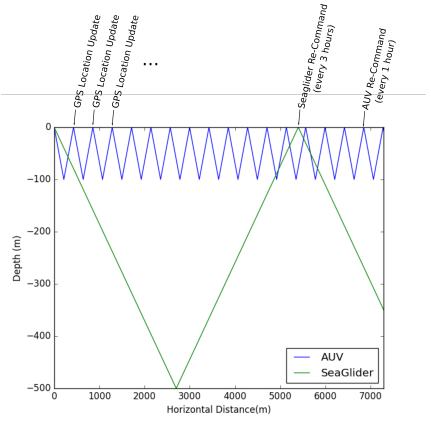


Figure 1: Graph of Seaglider and AUV movement with surface activities labelled.

The AUVs are much more flexible than the Seagliders in how they move through the water. However, for this experiment, they are treated very similar to the Seagliders. They repeatedly profile between the surface and some depth, only moving forward when profiling. The AUV will also avoid the ocean floor by ascending before the profiling depth has been reached. When at the surface, the AUV is able to update its location. Communication is done through an acoustic modem.

The final asset is the Wave Glider. The Wave Glider has two components, a float and a set of submerged fins, connected by a cable (Manley 2010). As such, the current that affects the asset is not at one single depth. For the purposes of this experiment, the current that the Wave Glider experiences is two-thirds the current at the surface and one-third the current at 10m.

Next State Generators

Next state generators are used to discretize the problem space. The generators use the properties of each asset, horizontal and vertical speed and maximum depth, a planning model, and different heuristics to generate the next states by simulating different actions that the asset can take in the planning model.

Baseline

This next state generator serves as the baseline for the experiment. Each time an asset is at the surface, the asset adjusts its heading to the direction of the next template waypoint. This is the simplest approach that will allow the asset to reach the waypoints along the path. As each state only has one neighbor, there is no actual search involved. This approach simulates commanding the assets with only the template waypoints.

This approach has the benefit of not needing an ocean model for an operational deployment. If there is poor correlation between the ocean model currents and the actual ocean currents, then this approach would be superior to others. In addition, there is very little in the way of operator intervention when deployed. Once the waypoints are given to the asset there is no need send any re-commands. The major downside is the affect the currents can have on the asset. If it is important to precisely follow the template path, then this approach may perform poorly in the presence of cross-currents. This approach was chosen as the baseline for two reasons, the lack of a need for a planning model and the similarity to the default behavior of the assets when given the list of template points.

Beam Search

The beam search next state generator can be seen in algorithm 3. This algorithm limits the number of possible next states to N bearings that are selected over some search angle theta. This search angle is centered on the bearing that points to the next template waypoint. For example, with a beam size of 5 and a search angle size of 30 degrees centered at 0 degrees, the bearings used to determine the next states would be -15, -7.5, 0, 7.5, and 15 degrees. The command point that is selected for each next state is set at a distance from the assets current location equal to the distance that asset would travel before the next time that it could be commanded, or the distance to the next waypoint on the template path, whichever is closer. Figure 2 shows the next states

Algorithm 3: Beam Search Next State

Note: Uses Planning Model

function BeamSearchNextState(*node*, *templatePath*)
  curWaypoint ← next waypoint in *templatePath*
  bearing ← bearing from *node* to *curWaypoint*
  curBearing ← *bearing* – search angle
  while *curBearing* < *bearing* + search angle **do**
    if distance to *curWaypoint* > command time * speed **then**
      point ← distance to *curWaypoint* at *curBearing*
    else
      point ← command time * speed at *curBearing*
    newNode ← simulate movement from current location to *point*
    newNode.planningPoints add *point*
    neighbors ← *neighbors* + *newNode*
    curBearing += search angle / (branching factor – 1)
  return *neighbors*
end function

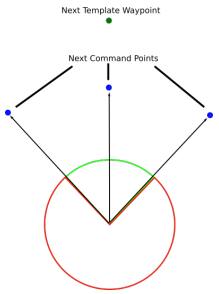


Figure 2: Graphic representation of the beam search next state generator. The search angle is the green arc and the next command points are labelled.

produced by beam search. The green arc is the valid angle that the bearings are chosen from, notice that it is centered on the bearing from the asset to the next template waypoint. N bearings are selected uniformly over this arc. In the case of the figure, N is 3. Note that this is different from a standard beam search. Normally every possible next state would be judged by a heuristic, then the top N would be used (Russell and Norvig 2009). In this case we calculate a fixed delta angle between the bearings as follows

$$\Delta\, angle = \frac{a}{(n-1)}$$

where *a* is the size of the search angle and *n* is the beam size. This is similar to using the top scoring heuristic with no currents, as the optimal bearing would be directly toward the next template waypoint.

The beam search approach has the benefit of using the predictive ocean model currents to better predict the trajectory that an asset will take. When the planning model is accurate then this helps to keep the asset on course and arrive at the next template waypoint more quickly and reliably. However, when the planning model is not accurate this approach can actually make the situation worse.

The approach was chosen as a way to discretize the possible command points that the asset could be commanded with. The set of possible bearings is limited to some search angle toward the next waypoint as it is very unlikely that the optimal direction of travel is going to be significantly different from the direction that the waypoint is in. The search angle is then discretized into N bearings as a small different in the angle is unlikely to have a large effect on the end result. This discretization greatly simplifies the search process by taking it out of the continuous space.

Algorithms

For this experiment, a continuous planner is used. A graphical representation of this algorithm is shown in figure 3. This algorithm runs a best-first search, using the planning model, starting from the assets current location, and with a template path that contains the waypoints that have not yet been visited. The best-first search algorithm generates a list of command points that are then given to continuous planner. Note that these command points are distinct from those that make up the template path. The blue lines in the figure represent that path that the best-first search finds and the red points are the command points that the are returned to the continuous planner. These points are used to simulate the movement of the asset with the nature model. The bearing of the asset is set so that it is heading toward the next command point. Every time the asset is able to update its location, this bearing is updated. How often the location can be updated depends on the asset being used.

The asset is simulated for an amount of time equal to the time between re-commanding the asset, this also depends on the asset being used. If the command point is reached before this re-command time, then the next command point in the list is used. A command point is considered to be reached if the asset passes within a certain threshold distance from it.

Each time an asset can be re-commanded the best-first search is run again, starting at the updated location of the asset. The old command points are discarded and the new ones from the best-first search are used until the next re-command. This process repeats until the goal is reached. In this case the goal state is successfully visiting every template waypoint in order. This process of repeated planning and simulation emulates the actual deployment of these assets. The best-first search also stores the best result seen so far. This is returned after a fixed number of iterations to prevent the search from attempting to exhaust every path when it is not possible to reach the goal state before the mission length has been exceeded, as this is impractical even for small

Algorithm 1: Continuous Planner

Note: Uses Nature Model for simulating movement

```
function ContinuousPlanner(startLocation, templatePath)
  curPath ← startLocation
  curWaypoint ← second waypoint on templatePath
  while true do
    endNode ← last node in curPath
    planPoints ← BestFirstSearch(endNode, templatePath)
    point ← first point in planPoints
    while time till re-command > 0 do
      newNode ← simulate movement to planPoint
      curPath ← curPath + newNode
      if newNode distance to point <= threshold then
        point ← next point in planPoints
      if newNode distance to curWaypoint <= threshold
  then
        if curWaypoint is final waypoint in templatePath then
          return success
        curWaypoint ← next waypoint on template
  path
    if curPath duration > mission length then
      return failure
end function
```

branching factors. Increasing this threshold will improve the results at the cost of extended runtime.

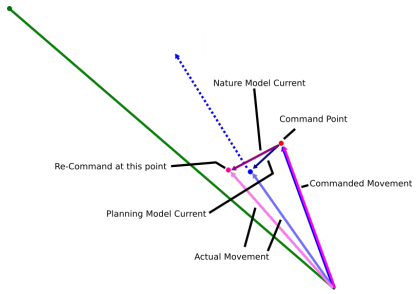


Figure 3: Graphic representation of the continuous planning process.

Best-First Search

Algorithm 2: Best-First Search
Note: Uses Planning Model
function BestFirstSearch(*startNode*, *templatePath*)
  Q ← *startNode*
  best ← *startNode*
  while *Q* not empty **do**
    curNode ← lowest score node in *Q*
    if *best* score < *curNode* score then
      best ← *curNode*
    if *curNode* is a goal state **then**
      return *curNode*.planningPoints
    if node expansion limit reached **then**
      return *best*.planningPoints
    neighbors ← next-state-generator(*curNode, templatePath*)
    for each *neighbor* in *neighbors* **do**
      Q ← *neighbor*
end function

The objective function that was used for best-first search combines distance travelled and time taken. The equation is

$$w_d * \frac{d_p}{d_t} + w_t * \frac{t_p}{t_t}$$

where w_d and w_t are weighting factors for the distance and time portions of the equation respectively, d_p is distance travelled thus far by the asset, d_t is the total distance of the template path, t_p is the total time taken by the asset so far, and t_t is the target time to complete the template path. The larger the ratio of w_d to w_t the more the algorithm will favor shorter distance paths over shorter time paths. By reducing the distance travelled the resulting path will stay closer to the template path even though the average distance from the template path is not included in the calculation.

The objective function takes into account the two metrics that we are using to evaluate the quality of the paths, time and distance. As these two metrics have completely separate units the ratio of distance travelled to expected total distance and the ratio of current time to target time are used instead. This allows them to be equated more easily.

The heuristic function used by the best-first search is the following equation

$$w_t * \frac{d_l}{s_a} * \frac{1}{t_t} + \ w_d * \frac{d_l}{d_t}$$

where w_d and w_t are weighting factors for the distance and time portions of the equation respectively, s_a is the horizontal speed of the asset, d_l is distance left to travel, t_t is the target time to complete to template path, and d_t is the total distance of the template path. As the asset is further along in the template path this number will decrease. In practice, this heuristic is only a measure of the distance left to travel, as the time left has a linear relationship with the distance. The two terms are calculated separately so the heuristic is weighted appropriately with respect to the objective function.

An approach similar to the one used for the objective function is used for the heuristic function. The ratio of the estimated distance left to the total distance of the template path and the ratio of the estimated time left to the total target time is used.

Experiment Setup

Seaglider

The Seagliders were given a speed of 0.266 m/s, a glide slope of 20 degrees, and a maximum depth of 500 meters. They were commanded every 3 hours. This is equivalent to one complete profile. At each surface from a profile the Seaglider location is updated. This allows the asset to adjust its bearing to point toward the next command point it is travelling to. The template path was from one corner of a 15km x 15km box to the opposite corner. This was done for the two diagonal pairs in each direction, for a total of 4 runs per location. Each run has a target completion time of 24 hours. The template waypoints have a threshold distance of 0.5 km. This is the distance that the Seaglider can be from the waypoint while still be considered to have visited it. During a deployment there would be two Seagliders operating concurrently, one for each diagonal. This pattern can been see in figure 4.

AUV

The AUVs were given a speed of 2.0 m/s, a glide slope of 25 degrees, and a maximum dive depth of 100 meters. They were commanded every hour. The AUVs also update their location whenever they surface. The template path is a "bowtie" pattern on a 3km by 3km box. The target completion time for a single bowtie is 1 hour. The template waypoints had a threshold of 0.1 km. This distance is smaller than that used for the Seagliders because of the shallower dive depth, which allows the AUVs to change bearings more often and be more precise. Similar to the Seagliders, during a deployment there would be 2 assets operating concurrently. They would be travelling in opposite directions on the path. This pattern can been see in figure 5.

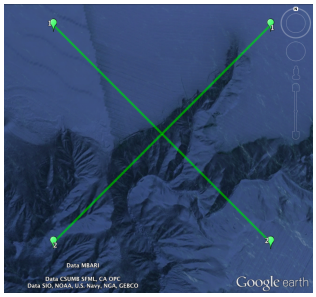


Figure 4: Template paths for the Seaglider experiments.

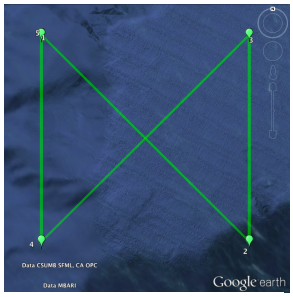


Figure 5: Template paths for the AUV and Wave Glider experiments.

Wave Glider

The Wave Gliders were given a speed of 2.0 m/s. As they only operate on the surface they do not have a glide slope or a maximum depth. They are also commanded every hour. The Wave Gliders updated their location every 10 minutes. The template path is the same bowtie pattern that is used for the AUVs, with the same target completion time. A waypoint threshold of 0.1 is also used for the Wave Gliders.

Test Locations

The testing was done in the model of Monterey Bay. Each test was executed at 100 different locations. The locations represent the center of the box that defines the template paths specific to each asset. Note that some of the locations are very close to shore or even located on land. These locations are discarded in testing as it is not possible to complete the template path starting from them. The locations can be seen in figure 6.

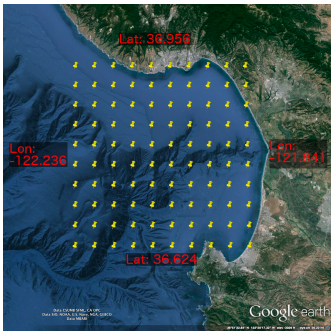


Figure 6: The 100 test locations for each of the experiments in the Monterey Bay model. The latitude and longitude labels represent the boundaries.

Results in Simulation

Seaglider

Figure 7 shows the feasibility analysis for the Seagliders when using the baseline. For each location there are 4 icons, 1 for each run. White diamond shaped icons represent locations that are invalid because they are too close to land or the Seaglider could not navigate the sea floor. The green icons represent the runs where the template path is possible to complete in a 36-hour window. The red icons with the dot show the runs where the path was not possible in the 36-hour window. Forty-seven locations for the Seaglider contain invalid runs. These runs are not included in any calculations.

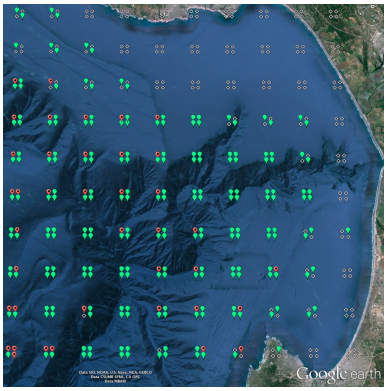


Figure 7: Seaglider feasibility analysis using the baseline. The result of 4 runs at each location are shown. White diamonds are invalid, green markers are successful, and red markers with a dot are failures.

For each next state generator, the number of successful runs, the average time to complete the runs, and the average distance from the template path weighted by time are used as metrics. Only the successful runs were considered when calculating the distance and time metrics. The results for the Seagliders tests can be seen in figure 10 and figure 11. In the time and distance metrics a 95% confidence interval is shown. Using the beam search approach reduced the average distance from the template path, but increased the average time taken. As a result, a lower percentage of the runs finished successfully in the 36-hour window.

A scatter plot comparing the results of the baseline and beam search can be seen in figure 12. Each data point represents a single run of the planner. We selected the best performing parameters for the beam search, a search angle of

20 degrees and a beam size of 7. In order to better understand the results, we filter the data in two different ways, when the currents are too strong for the Seaglider to complete the path in a reasonable amount of time and when the planning model is extremely inaccurate. In order to filter the case where the currents are too strong, we employed beam search using a planning model that is identical to the nature model, as this is the best performing search. If this was unable to complete the template path in 36 hours, then we removed the data point due to current strength. In order to filter out the cases where the planning model is inaccurate, we used the root-mean-square error of the currents along the template path. If the error is greater than 0.10 then we remove the data point due to error in the planning model. This number was selected by taking the average error of the cases in which beam search performed worse than the baseline in both the time and distance metrics. The averages of the data are also marked on the plot. From this scatter plot we can see that the beam search improves the distance to the template path, similar to what we saw in figure 10. The time to complete the template path does not change much between the baseline and the beam search next state generators.

Two paths are shown in figure 8 and figure 9. Figure 8 is an example of a failed path using the the baseline next state generator. The strong currents pushed the Seaglider significantly off course, preventing it from reaching its goal in time. Figure 9 shows an example from the beam search next state generator. The green line is the template path, the yellow line with yellow icons containing circles is the results from using the beam search next state generator, and the red path is from using the baseline next state generator. By having some information on the currents, the beam search next state generator is able to counteract them to stay closer to the template path.

AUV

Figure 13 shows the feasibility analysis for the AUVs. The icons represent the same outcomes as with the Seaglider, however there is only 1 icon per location. Eighteen locations are invalid when using the template path for the AUVs. In all other locations, the AUVs are successful in completing the bowtie paths within 36 hours. This was expected because of the short template path lengths and high speed of the AUVs compared to the currents.

The same metrics that were used with the Seaglider tests were used with the AUV tests. The results of the test can be

Figure 8: Seaglider example of a failed path when using the baseline.

Figure 9: Seaglider example where beam search performs better than the baseline.

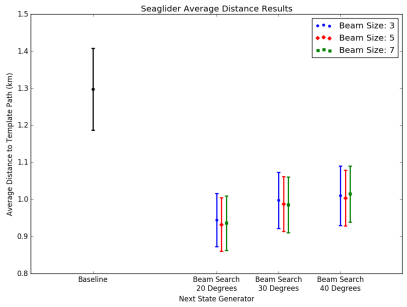


Figure 10: Seaglider distance results for the baseline and beam search next state generators with various search angles and beam sizes.

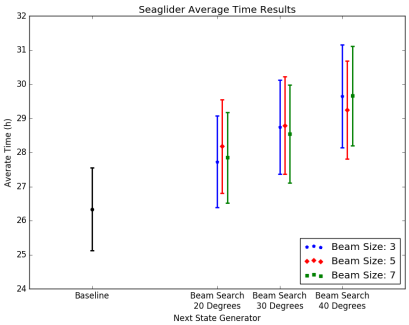


Figure 11: Seaglider time results for the baseline and beam search next state generators with various search angles and beam sizes.

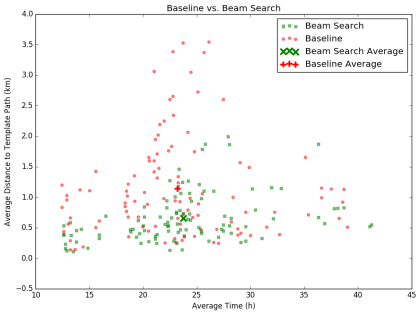


Figure 12: Seaglider baseline vs. beam search with a search angle of 20 and beam size of 7, comparing the distance and time metrics.

seen in figure 14 and figure 15. Using the beam search approach did not result in an improved path over the baseline. The quality of the paths actually decreased. The currents do not have as large of an affect on the AUVs because of the relatively large speed compared to the current speed, reducing the gain from using the planning model.

Wave Glider

The feasibility analysis for the Wave Glider is similar to that of the AUVs as they *have* the same template path. There are a few additional invalid locations because the Wave Gliders have a float and a submerged component. This means that they need slightly deeper waters to operate. There are 20 invalid locations. As with the AUVs, the feasibility analysis is the same for both next state generators.

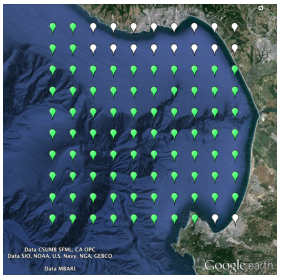


Figure 13: AUV feasibility analysis. White is an invalid location and green is a location where every run was successful.

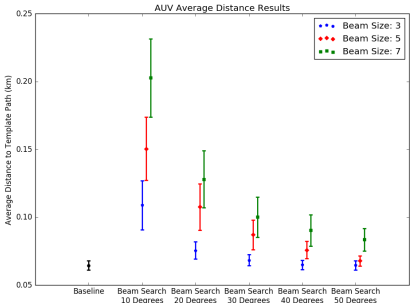


Figure 14: AUV distance results for the baseline and beam search next state generators with various search angles and beam sizes.

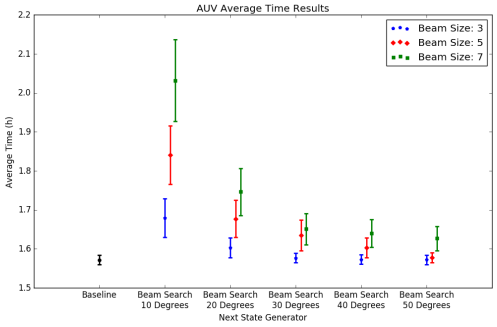


Figure 15: AUV time results for the baseline and beam search next state generators with various search angles and beam sizes.

The results for the Wave Glider test were very similar to that of the AUV. They can be seen in figure 16 and figure 17. The beam search next state generator did not improve the paths in either the average distance from the template or the average time to complete the path. The Wave Gliders are slightly slower than the AUVs, but not enough as to where to currents drastically affect them.

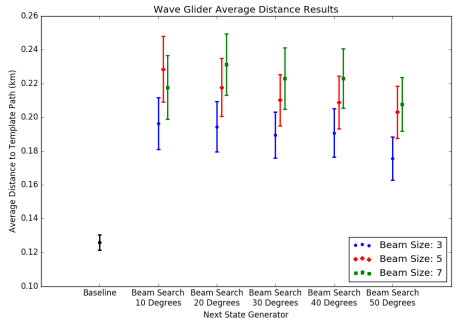


Figure 16: Wave Glider distance results for the baseline and beam search next state generators with various search angles and beam sizes.

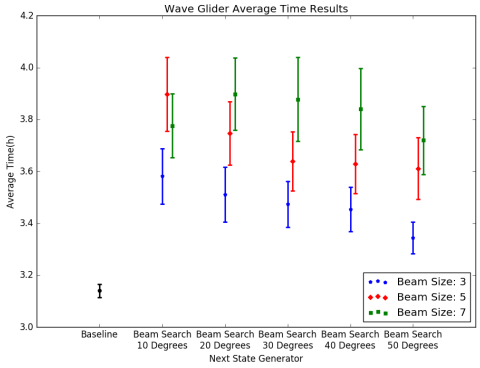


Figure 17:Wave Glider time result for the baseline and beam search next state generators with various search angles and beam sizes.

Related Work

A significant amount of work has been done in regards to path planning for underwater vehicles. However, there has been little work done on planning over short distances and following a given path. (Rao and Williams 2009) does planning for gliders over long distance, minimizing the energy used to reach some location, while we are planning over relatively short distances following some template path. (Thompson et al. 2010) uses the ROMS model to do path planning, but minimizes the time taken from the start location to the goal location. (Pereira et al. 2013) prevents gliders from surfacing in dangerous areas, such as shipping lanes, while travelling to a goal location, while we focus on

following a specific path. (Cashmore et al. 2014) uses AUVs to inspect features at a site efficiently. No ocean model similar to ROMS was used. (Alvarez, Garau, and Caiti 2007) also does not use an ocean model, but instead uses synthetic data with general algorithms to control a set of floats and gliders. (Dahl et al. 2011) and (Troesch et al. 2016a, Troesch et al. 2016b) address the control of vertically profiling floats using a current model but do not address other types of marine vehicles.

Continuous planning has become more prevalent in recent years and the evolution of this planning technique, with respect to multiple assets, is clearly described in (Durfee et al. 1999). (Myers 1999) describes a Continuous Planning and Execution Framework (CPEF), which integrates planning and execution through plan generation, monitoring, execution, and repair. Using an iterative repair process, as well as user interaction, CPEF is able to plan in unpredictable and dynamic environments, which is shown through tests in a simulation of an air-campaign for dominance. (Chien et al. 2000) presents Continuous Activity Scheduling Planning Execution and Replanning (CASPER), which also uses iterative repair as part of continuous planning, specifically for autonomous spacecraft control.

Future Work

There are a number of different possible extensions to this experiment. Different next state generators and heuristics could be developed that focus on the assets that did not benefit from the approach in this work. More research into the performance characteristics of beam search and the associated heuristics could be done. The beam search next state generator could be improved to select the next states more intelligently. Tests could be performed in different areas, such as those with stronger currents and different template paths could be used to better understand the behavior of the planner. The drop off location of the asset could be included in the planning. Ocean models with different fidelity could be used to understand the performance of the planner with more or less accurate models. A range of methods for interpolating the current model information between data points could be explored. Additionally, ROMS provides ensemble information from multiple runs with varying conditions, the planner could use search in the ensemble space and/or use ensembles to predict execution uncertainty and incorporate this to inform the generation process. Multi-agent planning could be used for multiple assets to achieve a goal. The usage of the model could be improved to include interpolation between the grid points.

Conclusion

This experiment has shown the benefits of using a predictive ocean model to do planning in order for an underwater vehicle to follow a template path. With the Seaglider, the beam search next state generator improved how well the asset could follow the template path compared to the baseline. However, with this result came a slight increase on the time taken to complete the template path.

However, this benefit does not extend to every type of asset. The baseline performed better than beam search when using Wave Gliders and AUVs in both how well the template path was followed and the time to complete the path. It is clear that the amount of benefit from this approach depends heavily on the vehicle and path in question. As such, a single approach may not be applicable to a wide variety of assets.

More research needs to be done in order to fully understand the behavior of the beam search next state generator and the associated objective and heuristic functions.

Acknowledgements

Portions of this work were performed at the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

References

Alvarez, A.; Garau, B.; and Caiti, A. 2007. Combining networks of drifting profiling floats and gliders for adaptive sampling of the ocean. In *Robotics and Automation, 2007 IEEE International Conference on*, 157–162. IEEE.

Cashmore, M.; Fox, M.; Larkworthy, T.; Long, D.; and Magazzeni, D. 2014. Auv mission control via temporal planning. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, 6535–6541. IEEE.

Chao, Y.; Li, Z.; Farrara, J.; McWilliams, J. C.; Bellingham, J.; Capet, X.; Chavez, F.; Choi, J.-K.; Davis, R.; Doyle, J.; et al. 2009. Development, implementation and evaluation of a data-assimilative ocean forecasting system off the central california coast. *Deep Sea Research Part II: Topical Studies in Oceanography* 56(3):100–126.

Chassignet, E. P.; Hurlburt, H. E.; Smedstad, O. M.; Halliwell, G. R.; Hogan, P. J.; Wallcraft, A. J.; Baraille, R.; and Bleck, R. 2007. The hycom (hybrid coordinate ocean model) data assimilative system. *Journal of Marine Systems* 65(1):60–83.

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

Dahl, K. P.; Thompson, D. R.; McLaren, D.; Chao, Y.; and Chien, S. 2011. Current-sensitive path planning for an un- deractuated free-floating ocean sensorweb. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, 3140–3146. IEEE.

Durfee, E. H.; Ortiz Jr, C. L.; Wolverton, M. J.; et al. 1999. A survey of research in distributed, continual planning. *Ai magazine* 20(4):13–22.

Eriksen, C. C.; Osse, T. J.; Light, R. D.; Wen, T.; Lehman, T. W.; Sabin, P. L.; Ballard, J. W.; and Chiodi, A. M. 2001. Seaglider: A long-range autonomous underwater vehicle for oceanographic research. *Oceanic Engineering, IEEE Journal of* 26(4):424–436.

Farrara, J. D.; Chao, Y.; Zhang, H.; Seegers, B. N.; Teel, E. N.; Caron, D. A.; Howard, M.; Jones, B. H.; Robertson, G.; Rogowski, P.; and Terrill, E. 2015. Oceanographic conditions during the orange county sanitation district diversion experiment as revealed by observations and model simulations. Submitted to Estuarine, Coastal and Shelf Science.

Li, P.; Chao, Y.; Vu, Q.; Li, Z.; Farrara, J.; Zhang, H.; and Wang, X. 2006. Ourocean-an integrated solution to ocean monitoring and forecasting. In *OCEANS 2006*, 1–6. IEEE.

Manley J. 2010. The Wave Glider: A persistent platform for ocean science. In *OCEANS 2010*, 1-5. IEEE-Sydney.

Mellor, G. L. 2004. Users guide for a three dimensional, primitive equation, numerical ocean model. Princeton, NJ: Princeton University.

Myers, K. L. 1999. Cpef: A continuous planning and exe- cution framework. *AI Magazine* 20(4):63–69.

Pereira, A. A.; Binney, J.; Hollinger, G. A.; and Sukhatme, G. S. 2013. Risk-aware path planning for autonomous underwater vehicles using predictive ocean models. *Journal of Field Robotics* 30(5):741–762.

Rao, D., and Williams, S. B. 2009. Large-scale path planning for underwater gliders in ocean currents. In *Australasian Conference on Robotics and Automation (ACRA), Sydney*. Citeseer.

Robinson, A. R. 1999. Forecasting and simulating coastal ocean processes and variabilities with the Harvard Ocean Prediction System. In Mooers, C. N. K., ed., *Coastal Ocean Prediction*, AGU Coastal and Estuarine Studies Se- ries. Washington, DC: American Geophysical Union. 77– 100.

Russell S.; Norvig P. Artificial Intelligence: A Modern Approach (Third Edition), Prentice Hall, 2009

Thompson, D. R.; Chien, S.; Chao, Y.; Li, P.; Cahill, B.; Levin, J.; Schofield, O.; Balasuriya, A.; Petillo, S.; Arrott, M.; et al. 2010. Spatiotemporal path planning in strong, dynamic, uncertain currents. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, 4778– 4783. IEEE.

Troesch M.; Chien S.; Chao Y.; and Farrara J. 2016 Planning and control of marine floats in the presence of dynamic, uncertain currents . *International Conference on Automated Planning and Scheduling (ICAPS) 2016*.

Troesch M.; Chien S.; Chao Y.; and Farrara J. 2016 Evaluating the Impact of Model Accuracy in Batch and Continuous Planning for Control of Marine Floats. Scheduling and Planning Applications Workshop, *International Conference on Automated Planning and Scheduling (ICAPS) 2016*. (under review)

YSI Systems. “EcoMapper AUV”. http://www.ysisystems.com. Accessed February, 2016.

Search Challenges in Natural Language Generation with Complex Optimization Objectives

Vera Demberg and **Jörg Hoffmann** and **David M. Howcroft**
Dietrich Klakow and **Álvaro Torralba**
Saarland University
Saarbrücken, Germany
{vera, howcroft}@coli.uni-saarland.de, {torralba, hoffmann}@cs.uni-saarland.de, dietrich.klakow@lsv.uni-saarland.de

Abstract

Automatic natural language generation (NLG) is a difficult problem already when merely trying to come up with natural-sounding utterances. Ubiquituous applications, in particular companion technologies, pose the additional challenge of flexible adaptation to a user or a situation. This requires optimizing complex objectives such as information density, in combinatorial search spaces described using declarative input languages. We believe that AI search and planning is a natural match for these problems, and could substantially contribute to solving them effectively. We illustrate this using a concrete example NLG framework, give a summary of the relevant optimization objectives, and provide an initial list of research challenges.

Introduction

As mobile devices are getting more and more ubiquitous, and speech recognition and synthesis have seen large performance improvements in recent years, NLG is necessary in an increasingly large number of applications and situations. Highly domain-specific template-based NLG approaches are becoming less viable, due to a lack of ability to adapt the generated utterances flexibly to a user or a situation, as would be especially important in companion technologies. For instance, a dialog system should generate utterances that are easier to comprehend and more redundant when a user is concentrating on another task (such as driving a car), but should generate concise utterances (which are often more complex) when the user can fully concentrate on the interaction with the dialog system (Demberg and Sayeed 2011).

Achieving flexible situation-adaptive NLG is a major challenge to, and an active research area in, Computational Linguistics, requiring the identification of suitable objectives and measures for controlling utterance complexity. It is also a major challenge to the design of search algorithms, for optimizing (combinations of) such objectives. We believe that the AI search community could make major contributions towards the latter. AI planning in particular is relevant given its focus on automation and powerful declarative models. Our mission in this paper is to provide a concise summary of the problem and the challenges ahead, in terminology accessible to the AI community, as a first step towards bringing the two communities together in this endeavor.

NLG traditionally proceeds in a pipeline comprising three phases: *document planning*, *microplanning*, and *surface realization*. During document planning, the system decides on the content to be conveyed, and what rhetorical structure connects said content. Microplanning then lexicalizes this content, choosing which words should be used to express it, and performs aggregation and referring expression generation. This results in a syntactico-semantic representation, forming the input to the surface realization component which generates the final natural language utterances from that input.

The boundary between microplanning and surface realization is fluid, varying the granularity of the microplanning output and, accordingly, the degrees of freedom assigned to the surface realization grammar. The surface realization process can include: lexical choice (e.g., restaurant vs. bar); structural choice (e.g., active vs. passive voice: "restaurant serves food" or "food is served by restaurant"); and choosing among adjective, prepositional phrase, or relative clause options as in "Almaz is an Eritrean restaurant", "Almaz is a restaurant with Eritrean food", or "Almaz, which is an Eritrean restaurant, . . . ".

The desired situation-adaptivity in NLG is a function not of *what* to say, but *how* to say it, and as such is naturally associated with the microplanning and surface realization phases. We here focus on the surface realization problem, the understanding being that, in applications, the surface realization grammar (and therewith the search) will be given sufficient freedom to encompass the relevant formulation differences.

Traditional optimization objectives for surface realization are to generate grammatically correct, natural-sounding sentences. Effectively optimizing these objectives is, already, not a solved problem, and could potentially benefit from AI search algorithms expertise. This is even more true for the complex optimization objectives required to achieve intelligent behavior in companion technologies and other ubiquitous applications. In what follows, to make matters concrete, we first consider a particular search-based surface realization framework, OpenCCG, overviewing its search algorithm in AI terms and in relation to AI search algorithms. We then summarize current research issus in the design of more complex NLG objectives & measures, towards the desired flexibility. We conclude the paper with a discussion

Figure 1: Example input for the realization algorithm representing the propositions be(Almaz, Eritrean) and have(Almaz, good_food).

of challenges to search algorithms, as well as possible approaches to address these.

A Concrete Example: OpenCCG

OpenCCG is a prominent state-of-the-art method for surface realization via search (White 2006, White and Rajkumar 2012). As our purpose is to illustrate basic aspects of the search, we do not provide a comprehensive summary and present a simplified version only. OpenCCG is based on *combinatory categorial grammars (CCG)*. It uses a so-called *chart realization* algorithm (e. g. (Kay 1996, Cahill and van Genabith 2006, Carroll and Oepen 2005)). Chart realization is a dynamic programming approach to language generation that performs a best-first search in the space of (partial) sentences, storing partial results in a *chart* table, and generating new search nodes by combining expanded nodes with entries from that table.

The input to surface realization is a labeled directed graph, representing the so-called *semantics*, i. e., the content we wish to convey: objects, properties, activity, and how they are connected. The target is to find a valid sentence that contains all the semantics in the input. Figure 1 illustrates an example input. Valid sentences containing this semantics are, e. g.:

(a) Almaz has good food and is an Eritrean restaurant.

(b) The Eritrean restaurant Almaz has good food.

Search nodes in OpenCCG consist of a string, i. e. the partial sentence contained in the node, as well as a *grammar category* that determines how the node can be combined with other nodes. A category may be a grammar item (S stands for sentence, N for noun, NP for noun phrase, and so on), or a function that composes several atomic expressions with *forward concatenation* ($/$) or *backward concatenation* ($\backslash$). For example, NP/N means that, if concatenated with another item of type N, we get an item of type NP; and NP concatenated with $S\backslash NP$ yields an item of type S.

We denote nodes as $\alpha(X)$, where α is the string and X the category. Nodes can be transformed and combined using different types of *rules*. Two strings can be concatenated via *application*, forward $[\alpha(X/Y)\ \beta(Y) \rightarrow \alpha\beta(X)]$ or backward $[\alpha(Y)\ \beta(X\backslash Y) \rightarrow \alpha\beta(X)]$. For example, we may concatenate "Eritrean" (NP/N) and "restaurant" (N) to obtain "Eritrean restaurant" (NP). Two strings can also be concatenated via *composition*, forward $[\alpha(X/Y)\ \beta(Y/Z) \rightarrow \alpha\beta(X/Z)]$ or backward $[\alpha(Y\backslash Z)\ \beta(X\backslash Y) \rightarrow \alpha\beta(X\backslash Z)]$. Additionally, there are unary rules changing the grammar type of a string in order to enable new combinations, e. g. "restaurant" (N) $\rightarrow$ "restaurant" $NP\backslash(NP/N)$. Importantly, rules combining two strings only ever allow to concatenate these, in either order – i. e., we can *not* insert another string later on in between the two. This is a design decision intended to keep the branching factor feasible.

In addition to the string and category $\alpha(X)$, search nodes are associated with information regarding how much of the input semantics is being conveyed, i. e., which parts of the input graph are *covered*. This is simply a bitvector that, for every element in the input, maintains the information whether or not a word covering that element has already been added to α.

The target of the search is to find a complete sentence, i. e. a node of category S covering the entire input semantics, maximizing the score with respect to the desired optimization metric (discussed further below).

To initialize the search space, a pre-process performs a lookup in a *dictionary*: a collection of all words – *lexical items* – that may be used, each associated with its grammar category and with the semantics it can cover. The preprocess retrieves all lexical items relevant to the input semantics at hand. For example, the semantics ⟨*restaurant*⟩ may be covered by the lexical items "restaurant"(N), "bistro"(N), and "café" (NP/N); the semantics ⟨*win*⟩ may be covered by different variants of that verb, differing with respect to tense, as well as the number of objects to be associated with the verb (intransitive, transitive, ditransitive verb). All these lexical items are inserted into (what the AI search community would refer to as) the open list, and search begins.

In the search, the chart serves as a dynamic programming cache. It stores the nodes that have already been expanded. Whenever a new node is expanded, successors are generated by *combining the node with every compatible node in the chart*. Two nodes are compatible if their semantics coverages are disjoint (we must not cover the same input element with more than one lexical entry), and the grammar categories can be concatenated by application or composition. Additional successors are generated as the result of applying unary rules, and adding connective words such as "that", "to", etc.

Duplicate elimination prunes nodes with the same semantics and category, even if their strings differ. However, in order to increase diversity, not all duplicates are pruned. Instead, the chart is divided into equivalence classes of same semantics and category, and the K-best nodes in each equivalence class are preserved, where "best" is according to optimization criterion score (see below), and K is a parameter that controls the trade-off between search efficiency and quality of the result.

The search is best-first, ordered by a *scoring function*. Contrary to heuristic search methods in AI, current OpenCCG scoring functions do not attempt to estimate "goal distance" (the number of steps until a complete sentence), nor the quality of the best completion of the partial solution at hand. The scoring functions do not attempt to predict the future at all, instead computing the optimization objective score solely on the content of the search node itself. In this sense, the use of scoring functions is akin to the use of evaluation functions in local search optimization methods,

despite the fact that search nodes are not feasible solutions (i. e., do not correspond to grammatically valid sentences). This is a simple and feasible solution, but may obviously be detrimental to the search. One research challenge is to find remedies based on the methods devised for the generation of heuristic functions in AI.

A common optimization objective – a means of measuring "how natural-sounding" a sentence is – is based on *n-grams*, measures of how common particular word tuples (e. g. triples, *trigrams*) are in natural language. For each word tuple, we get a probability measure for this word tuple occuring in a natural language text. The score of a partial sentence α is the sum of the negative logarithms of these probabilities, over all word tuples contained in α (we get back to this in the next section). Note that, applied to partial sentences during search, this creates a strong bias towards longer strings, simply because these contain more word tuples. Note furthermore the crucial difference to optimization criteria commonly considered in AI search problems: these are typically described declaratively as a function of the solution structure, and satisfy particular decomposition properties such as additivity. In contrast, n-grams are defined outside the input grammar, and behave in irregular ways gleaned from natural language text bodies. This gap between search model and optimization objective is intended and necessary, as the objective – a "natural-sounding" sentence – is difficult to capture in terms of a formal sentence-generation model.

The algorithm has an anytime flavor: When a valid solution, i. e. a complete sentence, is found, we continue looking for other sentences of better quality; the search stops only when the state space is exhausted, or a time limit is reached. Then, all complete sentences are extracted from the chart, and are evaluated by a refined quality objective for the final ranking, such as a neural network trained with more features than the n-grams used during the search (Nakatsu and White 2006, White and Rajkumar 2009, Rajkumar and White 2010, White and Rajkumar 2012, Rajkumar and White 2014).

One characteristic of OpenCCG search spaces, and a huge source of performance difficulties, are *dead ends*, i. e., search nodes that cannot be completed into valid sentences. There are at least two major sources of dead-ends: (a) wrong grammar categories that cannot anymore be completed into a sentence conveying the desired semantics; (b) applying combination rules in the wrong order. The former arises, e. g., when selecting an intransitive verb, with a category allowing no direct object, for a sentence that requires such an object. An example for the latter is the partial sentence "this restaurant has", which is a dead-end in our example as we cannot concatenate it with anything expressing that the restaurant is Eritrean – recall that we cannot insert new strings *within* the partial sentence.

Difficulty (a) is inherent to surface realization, and poses an interesting challenge to dead-end detection (we get back to this later). Difficulty (b) is more harmless, in the sense that it is an artifact of the way the combination rules are designed. *Chunking* has been designed as an optimization to counter-act this artifact (amongst other things) (White 2004, White 2006). It identifies sub-sentences in a preprocessing stage, and forbids combining nodes that refer to different sub-sentences until the semantics within each sub-sentence have been fully covered. For example, chunking avoids combining "restaurant" with "has" until it has been combined with "Eritrean".

New Complex Optimization Objectives in Natural Language Generation

Traditionally, natural language generation systems have mostly focussed on generating text for some fixed quality objective (e.g., grammaticality). More recently however, there is an increased interest in more flexible types of generation targets. In particular, how to automatically generate utterances with an optimal trade-off between complexity and conciseness for a specific user in a given situation? An example use case would be an in-car spoken dialog system. In this kind of setting, what is "optimal" changes based on the driving situation. Passengers adapt to the difficulty of driving conditions, speaking less overall, using less complex utterances, and speaking more about traffic when drivers face a challenging task (Crundall et al. 2005, Drews, Pasupathi, and Strayer 2008). Remote conversation partners, e. g. on a cell phone, do not adapt to the driver's cognitive load (as they can't directly observe it), and are therefore more likely to exceed the driver's channel capacity, increasing the likelihood of an accident. Balancing communicative efficiency against, e.g., safety concerns therefore requires the development of adaptive natural language generation systems which can target different levels of information density in different contexts.

In psycholinguistics, (Hale 2001) has suggested the information-theoretic notion of *surprisal* for quantifying the processing difficulty caused by a word. Surprisal is defined as the Shannon Information (Shannon 1948), i. e., the negative log probability of a word in context (Equation 1), where context is usually operationalized as the preceding sequence of words (Equation 2):

$$\begin{aligned} surprisal(w_n) &= -logP(w_n|\text{CONTEXT}) && (1)\\ &= -logP(w_n|w_1 w_2 ... w_{n-1}) && (2) \end{aligned}$$

Here, w_i denotes the i^{th} word in the sentence. In this formulation, a word carries more *information*, and is more difficult to process, when it is less predictable based on the preceding words. Likewise, a word that is totally predictable carries no new information, and is easy to process. This formulation of surprisal has been shown to correlate with reading times (Levy 2008, Demberg and Keller 2008) and the N400 component of electroencephalographic (EEG) event-related potentials (ERPs) (Frank et al. 2015), suggesting that surprisal is a valid measure of comprehension difficulty.

On the production side, the task of the producer is to distribute information across the utterance. Human speakers appear to be sensitive to information density, altering their use of optional linguistic markers in a way that avoids large peaks in surprisal (Levy and Jaeger 2007). The *uniform information density* hypothesis (UID; (Jaeger 2006, Jaeger

2010)) observes that rational speakers should want to communicate as much information as possible without overwhelming their listener, leading speakers to use a relatively uniform information distribution near the *channel capacity*, the maximum amount of information comprehensible to their listener. Initial evidence suggests that NLG systems sensitive to information density produce outputs more highly rated by humans (Rajkumar and White 2011, Dethlefs et al. 2012).

For example, consider the case where we want to convey the message shown in Figure 1; the two alternative verbalizations shown in (a) and (b) differ in information density[1]: (a) has an average surprisal of 7.15 bits per word while (b) has 7.84 bits per word, meaning that (b) is more informationally dense than (a). This suggests that (a) should be preferred over (b) in situations where the user cannot give their full attention to the linguistic task. Note however, that utterance (b) is the one that conveys information more uniformly, with an average change in surprisal from word to word of 2.44 bits, versus 3.31 bits in (a).

More detailed information on UID and surprisal may be found in (Crocker, Demberg, and Teich 2015), published in this same issue.

Another measure is *propositional idea density*, which has been shown to affect reading times and recall (Keenan and Kintsch 1973). The semantic representation can then be used to calculate the number of words used to convey a proposition. For example, (a) and (b) contain 9 and 7 words respectively though they encode the same 5 propositions from Fig. 1, resulting in idea densities of 0.555 and 0.714, respectively. On this analysis, therefore, (a) is less informationally dense and may be easier to read.

Another possible objective is minimizing the length of syntactic dependencies (Gibson 1998, Gildea and Temperley 2010): in a nutshell, how far apart related words are placed in the sentence. Such features have been shown to improve surface realization quality in a generate-and-rank approach (Rajkumar and White 2011) where complete solutions are first generated and then ranked. But they have yet to see use *during* the generation process coming up with the solutions in the first place.

Challenges for AI Search Algorithms

Current surface realizers do, generally speaking, exhibit reasonable performance, yet significant deficiencies remain to be overcome. In particular, they often do not succeed in generating grammatically valid sentences within the given runtime limit, which in practice is typically small, in the order of one second. In such cases, they have to resort to other approximate methods that relax the grammar rules. Apart from these basic issues relevant to sentence realization as it stands, new challenges are posed by the more complex optimization objectives as just discussed. We list some concrete challenges and ideas in what follows.

[1]Surprisal values based on (Demberg, Keller, and Koller 2013) obtained from http://tinyurl.com/pltagdemo

Dead-end detection. As we have outlined, dead-ends are a major issue in OpenCCG (and related) search spaces. The same can be said of the search spaces in manifold AI problems, and there is a wealth of ideas whose potential in sentence realization is worth exploring. The AI Planning literature in particular has recently considered a variety of approaches towards dead-end detection. In particular, many known heuristic functions (*critical paths* (Haslum and Geffner 2000), *abstraction* (Edelkamp 2001, Helmert et al. 2014, Hoffmann, Kissmann, and Torralba 2014), *partial delete-relaxation* (Keyder, Hoffmann, and Haslum 2014)) have this capability, and could be useful in sentence realization problems. Planning languages are powerful enough to capture the category (as well as semantic) aspects of CCG search nodes and node combination rules, so a compilation approach could be feasible in principle. In practice, implementing the techniques natively, and exploiting the particular structure of CCG specifications, seems more promising.

Partial-order reduction. As in many other search problems, surface realization search spaces may contain permutative transition paths leading to identical search states. Partial-order reduction techniques (e. g. (Valmari 1989, McMillan 1993, Bonet et al. 2008, Wehrle and Helmert 2014)), originating in Verification and well established also in AI Planning, are a possible remedy which, to our knowledge, remains unexplored in NLG.

Predictive scoring functions. Scoring functions in OpenCCG evaluate search nodes based solely on their partial-solution content, without any predicition of possible completions. This can (obviously) be detrimental. For example, if we want to keep information density below a threshold, or keep syntactic dependencies short, then the score of partial sentences should take into account how we may be able to cover the remaining semantics. The obvious remedy, from an AI heuristic search perspective, is to devise *relaxations* that complete a search node $\alpha(X)$ into an approximate full sentence $\overline{\alpha}$. Arbitrary optimization objectives can then be evaluated on $\overline{\alpha}$, and be taken as an estimate of the quality of the best possible completion of $\alpha(X)$. But, for this to make sense, we need to ensure that $\overline{\alpha}$ does indeed correspond to a *best possible* completion. This is a known problem in AI for good-natured objectives that decompose over the structure of the solution, like additive action costs. But what if the optimization objectives are *non-local*, not decomposing as easily?

Non-local optimization objectives. The traditional optimization objectives in sentence realization, and all the more so the new complex optimization objectives discussed in the previous section, depend on *context*. N-grams depend on the neighboring words. Surprisal depends on the sentence prefix. Uniform density distribution is a property of the sentence as a whole, so depends on the sentence parts already fixed during the search. The same applies to the length of syntactic dependencies. To approximate best possible completions, the computation of relaxed solutions must take into

account such complex objectives. In some approaches, like abstractions (projections and merge-and-shrink), this might be relatively straightforward as we can apply non-local criteria to abstract solution paths. In other approaches, like (partial) delete-relexation methods, this is less clear. One possibility could be a limited post-optimization of relaxed solutions (within each call to the predictive scoring function), so that the objectives can, again, be applied to complete sentences.

Target-value search. In adaptive language generation systems, in particular when we wish to adapt information density as appropriate in the current user context, the objective often is not to minimize a function, but instead to *find a solution whose score is close to a particular value of that function*. This is known as *target-value search*, a topic that has been considered in AI graph search already (Kuhn et al. 2008, Linares López, Stern, and Felner 2014), but has not been given extensive attention. Substantial challenges still remain on the AI side itself, in particular pertaining to the design and computation of heuristic functions: The "best possible completion" is now no longer the cheapest possible path postfix, but instead one whose cost corresponds to the "remaining target cost" as closely as possible. This raises completely new challenges for AI Planning heuristics. It raises more complex challenges still for non-local optimization objectives, cf. above.

In conclusion, there is a lot of work still to do, but many ideas from AI Search and Planning appear to be promising for better NLG surface realization. We hope that our paper can contribute to making this happen.

Acknowledgements

This work was partially supported by the DFG excellence cluster EXC 284 "Multimodal Computing and Interaction", the DFG collaborative research center SFB 1102 "Information Density and Linguistic Encoding", as well as the EU FP7 Programme under grant agreement no. 295261 (MEALS). We thank Maximilian Schwenger for discussions. We are also grateful to Almaz for great Eritrean food.

References

Bonet, B.; Haslum, P.; Hickmott, S. L.; and Thiébaux, S. 2008. Directed unfolding of petri nets. *Transactions on Petri Nets and Other Models of Concurrency* 1:172–198.

Cahill, A., and van Genabith, J. 2006. Robust pcfg-based generation using automatically acquired LFG approximations. In Calzolari, N.; Cardie, C.; and Isabelle, P., eds., *Proceedings of the 21st International Conference on Computational Linguistics (ACL'06)*. ACL.

Carroll, J. A., and Oepen, S. 2005. High efficiency realization for a wide-coverage unification grammar. In *Natural Language Processing–IJCNLP*, 165–176.

Crocker, M. W.; Demberg, V.; and Teich, E. 2015. Information density and linguistic encoding (ideal). *KI - Künstliche Intelligenz*.

Crundall, D.; Bains, M.; Chapman, P.; and Underwood, G. 2005. Regulating conversation during driving: a problem for mobile telephones? *Transportation Research Part F: Traffic Psychology and Behaviour* 8(3):197–211.

Demberg, V., and Keller, F. 2008. Data from eye-tracking corpora as evidence for theories of syntactic processing complexity. *Cognition* 109(2):193–210.

Demberg, V., and Sayeed, A. 2011. Linguistic cognitive load: implications for automotive uis. In *Adjunct Proceedings of the 3rd International Conference on Automotive User Interfaces and Interactive Vehicular Applications (AutomotiveUI 2011)*.

Demberg, V.; Keller, F.; and Koller, A. 2013. Incremental, predictive parsing with psycholinguistically motivated tree-adjoining grammar. *Computational Linguistics* 39(4):1025–1066.

Dethlefs, N.; Hastie, H.; Rieser, V.; and Lemon, O. 2012. Optimising Incremental Dialogue Decisions Using Information Density for Interactive Systems. In *EMNLP-CoNLL*, 82–93. ACL.

Drews, F. A.; Pasupathi, M.; and Strayer, D. L. 2008. Passenger and cell phone conversations in simulated driving. *Journal of Experimental Psychology: Applied* 14(4):392–400.

Edelkamp, S. 2001. Planning with pattern databases. In Cesta, A., and Borrajo, D., eds., *Proceedings of the 6th European Conference on Planning (ECP'01)*, 13–24. Springer-Verlag.

Frank, S. L.; Otten, L. J.; Galli, G.; and Vigliocco, G. 2015. The erp response to the amount of information conveyed by words in sentences. *Brain and language* 140:1–11.

Gibson, E. 1998. Linguistic complexity: Locality of syntactic dependencies. *Cognition* 68(1):1–76.

Gildea, D., and Temperley, D. 2010. Do Grammars Minimize Dependency Length? *Cognitive Science* 34:286–310.

Hale, J. T. 2001. A Probabilistic Earley Parser as a Psycholinguistic Model. In *NAACL*.

Haslum, P., and Geffner, H. 2000. Admissible heuristics for optimal planning. In Chien, S.; Kambhampati, R.; and Knoblock, C., eds., *Proceedings of the 5th International Conference on Artificial Intelligence Planning Systems (AIPS-00)*, 140–149. Breckenridge, CO: AAAI Press, Menlo Park.

Helmert, M.; Haslum, P.; Hoffmann, J.; and Nissim, R. 2014. Merge & shrink abstraction: A method for generating lower bounds in factored state spaces. *Journal of the Association for Computing Machinery* 61(3).

Hoffmann, J.; Kissmann, P.; and Torralba, Á. 2014. "Distance"? Who Cares? Tailoring merge-and-shrink heuristics to detect unsolvability. In Schaub, T., ed., *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI'14)*. Prague, Czech Republic: IOS Press.

Jaeger, T. F. 2006. *Redundancy and Syntactic Reduction in Spontaneous Speech*. Unpublished dissertation, Stanford University.

Jaeger, T. F. 2010. Redundancy and reduction: speakers manage syntactic information density. *Cognitive Psychology* 61(1):23–62.

Kay, M. 1996. Chart generation. In Joshi, A. K., and Palmer, M., eds., *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, 200–204. Morgan Kaufmann / ACL.

Keenan, J., and Kintsch, W. 1973. Reading Rate and of Propositions Retention as a Function of the Number in the Base Structure of Sentences. *Cognitive Psychology* 5:257–274.

Keyder, E.; Hoffmann, J.; and Haslum, P. 2014. Improving delete relaxation heuristics through explicitly represented conjunctions. *Journal of Artificial Intelligence Research* 50:487–533.

Kuhn, L.; Price, B.; de Kleer, J.; Do, M.; and Zhou, R. 2008. Heuristic search for target-value path problem. In *Proceedings of the 1st International Symposium on Search Techniques in Artificial Intelligence and Robotics*.

Levy, R., and Jaeger, T. F. 2007. Speakers optimize information density through syntactic reduction. *Advances in Neural Information Processing Systems* (20).

Levy, R. 2008. Expectation-based syntactic comprehension. *Cognition* 106(3):1126–77.

Linares López, C.; Stern, R.; and Felner, A. 2014. Solving the target-value search problem. In Edelkamp, S., and Bartak, R., eds., *Proceedings of the 7th Annual Symposium on Combinatorial Search (SOCS'14)*. AAAI Press.

McMillan, K. L. 1993. Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. In von Bochmann, G., and Probst, D. K., eds., *Proceedings of the 4th International Workshop on Computer Aided Verification (CAV'93)*, Lecture Notes in Computer Science, 164–177. Springer.

Nakatsu, C., and White, M. 2006. Learning to say it well: Reranking realizations by predicted synthesis quality. In Calzolari, N.; Cardie, C.; and Isabelle, P., eds., *Proceedings of the 21st International Conference on Computational Linguistics (ACL'06)*. ACL.

Rajkumar, R., and White, M. 2010. Designing agreement features for realization ranking. In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, 1032–1040.

Rajkumar, R., and White, M. 2011. Linguistically Motivated Complementizer Choice in Surface Realization. In *UCNLG+Eval: Language Generation and Evaluation Workshop*, 39–44. ACL.

Rajkumar, R., and White, M. 2014. Better surface realization through psycholinguistics. *Language and Linguistics Compass* 8(10):428–448.

Shannon, C. E. 1948. A Mathematical Theory of Communication. *The Bell System Technical Journal* 27(3):379–423.

Valmari, A. 1989. Stubborn sets for reduced state space generation. In *Proceedings of the 10th International Conference on Applications and Theory of Petri Nets*, 491–515.

Wehrle, M., and Helmert, M. 2014. Efficient stubborn sets: Generalized algorithms and selection strategies. In Chien, S.; Do, M.; Fern, A.; and Ruml, W., eds., *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS'14)*. AAAI Press.

White, M., and Rajkumar, R. 2009. Perceptron reranking for CCG realization. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1-Volume 1*, 410–419.

White, M., and Rajkumar, R. 2012. Minimal dependency length in realization ranking. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, 244–255.

White, M. 2004. Reining in CCG chart realization. In Belz, A.; Evans, R.; and Piwek, P., eds., *Proceedings of the 3rd International Conference atural Language Generation*, volume 3123 of *Lecture Notes in Computer Science*, 182–191. Springer.

White, M. 2006. Efficient realization of coordinate structures in combinatory categorial grammar. *Research on Language and Computation* 4(1):39–75.

TIAGO– Tool for Intelligent Allocation of Ground Operations on Cluster-II

Simone Fratini, Nicolas Faerber, Nicola Policella, Bruno Sousa
European Space Agency, Germany
name.surname@esa.int

Abstract

ESA's four Cluster-II satellites conduct three dimensional in-situ measurements of the Earths magnetosphere. The mission is operated from the European Space Operations Centre in Darmstadt, where the Flight Control Team prepares the routine operations for the spacecraft. A major aspect of mission operations is planning and scheduling of ground station passes. The main driving factor of ground station scheduling is to keep the fill level of the on-board memory of the four satellites as low as possible and avoid any overwriting of scientific and housekeeping data.

The process of plan creation is usually done by a member of the Flight Control Team and requires heavy manual manipulation of the plan and trial-and-error approaches. The vast amount of constraints which have to be considered during this process, result in a time intensive activity which requires an investment of up to 1.5 man-days per week.

To ease this process, an AI-based Tool for Intelligent Allocation of Ground Operations, TIAGO, has been developed for automated ground station pass planning and optimization. TIAGO uses a domain independent planner and scheduler as core solving process and it is fully integrated in the mission ground segment software. This paper describes the problem, the methodological approach followed for the translation into a planning problem and the architectural aspects of the integration of the planning technology with the Cluster-II mission control software platform.

Introduction

The Cluster-II satellite constellation is a cornerstone mission in the European Space Agency's Horizon 2000 missions programme. The tetrahedral formation of the four identical, spin-stabilised satellites allows them to do precise in-situ measurements of the Earth's magnetosphere in three dimensions. The mission's objective is to map the electromagnetic environment of Earth in space and time. Especially regions where solar particles interact with the Earth's magnetic field – like the polar cusps, the bow shock and the magnetotail – are of high interest. For this, the spacecraft are equipped with eleven instruments from European and American institutions. To be able to compare the results from the measurements of the four spacecraft in space and time, the instruments on all spacecraft need to be identical and operated simultaneously.

The Cluster-II mission is controlled and operated from the European Space Operations Centre (ESA-ESOC) in Darmstadt, Germany. The main responsibility for the healthcare of the spacecraft lies with the Flight Control Team (FCT), consisting of engineers, analysts and spacecraft controllers. A web-based front-end, ClusterWeb, which was developed by the FCT itself, is used by mission operators and planners to interact with mission data, plans, and active constraints.

An important aspect of the mission operation is the planning of ground station passes. Aside from the input of the science operation planning, the ground station plan also depends on the input of the Flight Dynamics team and on ground station availabilities. Ground station passes enable the FCT to download the scientific data from the spacecraft and check the overall health of the satellites. The large number of constraints involved in pass scheduling requires a member of the Cluster-II FCT to dedicate about 1.5 working days to planning and scheduling activities during one week. Due to the complexity of the problem, the resulting plans are often not optimal which results, for instance, in higher costs for tracking hours than necessary.

To ease this process, a tool based on AI technologies, TIAGO, Tool for Intelligent Allocation of Ground Operations, has been developed as a collaboration between the Cluster-II FCT and the Advanced Mission Concepts section at the European Space Agency. The approach chosen was to model the problem as planning problem and to use a planner deployed on top of the ESA APSI (Advanced Planning and Scheduling Initiative) platform. The solving technology is based on timeline planning, an approach well consolidated to provide advanced solutions to support space operations (Chien et al. 2012). One of the reasons is the capability of enabling, in a flexible way, the integration of planning and scheduling. Moreover, various software development environments exist for rapid prototyping, test and synthesis of new planning and scheduling applications based on timeline planning (EUROPA (EUROPA 2008), ASPEN (Chien et al. 2000)). ESA contributed in this area by promoting the development of APSI (Fratini and Cesta 2012) and APSI-related activities (Cesta et al. 2011; Policella, Oliveira, and Benzi 2013). In general, a series of activities have proved the capability of planning systems in supporting different aspects of space mission operations like spacecraft autonomy (Muscettola et al. 1998), Earth Obser-

vations allocation (Bensana, Lemaitre, and Verfaillie 1999), Planning & Execution (Knight et al. 2001), Operations support (Cesta et al. 2007), and Instrument experiment allocation[1] (Johnston and Giuliano 2011).

Compared to previous APSI-based examples, TIAGO follows the same approach, but the role played by the domain independent planning technology is more central. In fact TIAGO models the whole problem into a planning problem, and solves it using a domain independent planner with automatized elicitation of relevant information from the model to drive the search (needed to cope with the size and the complexity of the real problem). This approach required an improvement of the planner both at technological level and at the level of modeling capabilities, but gave much more flexibility and reusability at the end. Moreover the integration of TIAGO with the the mission ground segment software has been implemented to automatize the generation of the planning problem from the mission data and the translation of the plan back to the mission database.

In the next section we present the mission scenario, then we describe the application and the planning technology behind the application, discussing the model and the problem solving process. We conclude by providing a concise evaluation of the current system and its possible improvements.

Mission Scenario

The main objective of the ground station scheduling for Cluster-II is to keep the SSR (Solid State Record, the satellite on-board memory) fill level as low as possible and avoid any loss of scientific or housekeeping data. In particular it is necessary to empty the SSR when the satellites are approaching a long no-visibility period (like before an eclipse). For this, a ground station plan with the exact contact times to the four spacecraft is created and revised by the FCT on a weekly basis.

The ground station pass plan initially provided by the Ground Station Planning Office, or ESTRACK, does not fully satisfy the constraints and requirements of the Cluster mission. The reason for this is that specific and important information, like the SSR fill level of the spacecraft or the science modes cannot be made available in advance. Therefore heavy manual interaction and manipulation of the original plan is required, which makes a non trivial investment of time, requiring a lot of trials and errors, based on the operator's experience. Due to the large number of constraints involved, the process of scheduling is very complex and the solution is often not optimal.

The planning and scheduling process of the Cluster mission is driven by different factors with diverging priorities. As already mentioned, the most important factor of the ground station scheduling is to avoid overwriting of SSR data. For instance, ground station passes need to be allocated more frequently in periods of high data production rate and less frequently in no science periods. The success of the mission is strongly related to the provided scientific data. Therefore, a maximized data return is desirable.

[1]This work also support the operations of Cluster II, focusing on the scheduling of the Wave Band Data plasma experiments.

To achieve this goal, the produced ground station plans should also be robust to the unplanned loss of ground station passes and therefore tracking hours during a planning period. The loss of tracking hours can be caused by unpredictable malfunction of the spacecraft or the ground station, so called anomalies. Satellite launches require short-term booking of ground stations to support the launch and early orbit phase of a mission which results in a cancellation of all bookings on certain ground stations. A further cause for losing tracking hours can be imposed by a natural phenomenon of the ionosphere. It causes fluctuations of the Automatic Gain Control and a bad signal-to-noise ratio which results in the loss of data which need to be recovered during subsequent passes.

Ground station planning and scheduling is an activity which needs to be conducted on a weekly basis. Planning actions and the release of plans are grouped into planning periods which usually span about a week and start on Friday. The FCT receives multiple input which need to be considered for this problem. Passes on ground stations can be distinguished by their different types, describing the kind of properties of the pass and the operations that need to be carried out before, during and after that pass. For what concerns this specific problem, the following have to be considered:

- Ground Station Visibility: an envelope of time windows over which passes have to be allocated. Currently the four Cluster satellites share 4 ground stations;
- ESTRACK Schedule: a schedule of booking of ground stations by other missions or maintenance activities;
- Schedule of Science Modes: a science schedule which gives the envelope of data production;
- Link Budget: Cluster uses mostly satellite dishes with a diameter of 15m which does not allow a downlink in high bit rate (hbr, 262144 bps) close to apogee when the slant range between ground station and satellite is very high with about 100 000 km. Hence, data are mostly dumped in low bit rate (lbr, 131072 bps).
- SSR Fill Level: it takes into consideration residual data from previous operational periods and other data production foreseen during the current period besides the normal operational ones (provided by the schedule of the science modes).

In order to operate the spacecraft without problem and to be able to conduct ground station passes, some operational requirements have to be considered during:

- Availability: a ground station must be visible and should not be booked by any other activity in order to be able to book a Cluster ground station pass.
- Configuration: before the actual begin of tracking, a ground station configuration activity of 45 min has to be considered. During this time, the ground station must be available and should not be booked by any other activity. After the end of tracking, a ground station post-pass activity of 15 min has to be considered. Similar to the pre-pass activity, no other activity should be booked during this time. When two consecutive passes of two different Cluster-II spacecrafts are scheduled on the same

ground station, the time accounting for the post and pre-pass activities can be reduced to *exactly* 10 min (in order to change between the different carrier frequencies of the spacecraft). If this 10 min interval cannot be satisfied, a configuration time of at least 1 h (15 min post-pass and 45 min pre-pass activities) has to be considered.

- Science Modes: the satellite's scientific operations are defined by the Telemetry Data Acquisition (TDA) modes. Depending on the TDA mode, the data production rate and the type of recorded data differs. Overall eleven different TDA modes are defined.
- Pass Properties: for cost efficiency reasons, scheduled passes should have a duration of at least 1 h. Also, the distance between two consecutive ground station passes should not be bigger than 40 hours.
- Eclipses: Eclipse operation is a very critical phase during the year. Due to the lack of operational batteries, the spacecraft need to be shut down before every eclipse and re-configured for nominal operation after each eclipse. For this, dedicated Eclipse Switch-OFF and Eclipse Recovery passes have to be guaranteed (this implies the need of emptying or reducing as much as possible the SSR level).

The main driving factor which dictates the plan generation is the SSR level. Furthermore, it is important to generate plans robust to the loss of tracking hours. An increasing robustness of a plan can be achieved by decreasing the hours between two consecutive passes and reducing the duration of single passes while the overall number of passes increases (according to the constraints stated above). The disadvantage of increasing the robustness in this way is the rising number of configuration hours as the number of passes is directly proportional to it (one configuration hour per pass).

TIAGO

TIAGO, the Tool for Intelligent Allocation of Ground Operations, has been implemented using a domain independent planner and scheduler, PLASMA (PLAn Space Multi-solver Application), developed on top of the ESA's APSI platform (Fratini et al. 2015). In order to guarantee a smooth integration of the planning technology with the existing mission databases the key point is to translate data into planning problems in PDL, the problem description language of the APSI platform, and to translate the plan back. In addition to that, a planning model specified in DDL.3, the domain modeling language of the APSI platform, has to be considered as well as an input of TIAGO. This file is used both to avoid having the domain information compiled into the tool, and to support the translation of the mission data to the final PDL problem.

Analysis of the Requirement

In order to elicit the requirements and to translate the scenario into a planning model, various interactions with the Cluster-II FCT have been performed, ending up in having one member of the FCT working directly on the planning model and with the planning technology.

It was clear from the beginning that, for a better usability, the scheduling tool should have been integrated seamlessly in the current scheduling workflow which consists of the actual scheduling activities performed through their main tool, ClusterWeb, and the whole ground segment software architecture. The amount of manual interactions should have to be minimized and the overall planning process should have been kept as simple as possible.

Moreover, as mission operation underlies a constant change of constraints, the need to implement new models of the scheduling constraints and properties during the operational life of the application have been raised. This was a very critical point, because the models should be then easily exchangeable and understood by different members of the FCT and the planning technology cannot be provided as a black-box, with the model designed by experts and never updated afterward. Finally the plans generated by the tool had to be made available to the mission database, to allow the use of the existing interfaces to existing mission scheduling software.

Regarding the size of the problem, a planning period of Cluster-II stretches usually over a week. Nevertheless, planning and adjustments to the schedule have to be done whenever new inputs concerning visibilities, bookings or science modes are received. The final plan is usually submitted two weeks in advance, but scheduling activities can span up to 3.5 months in advance and are regularly sent to the ground station planning office. During a planning period up to 40 visibilities need to be analyzed and distributed between the four different spacecraft. Schedules of a planning period contain approximately 35 ground station passes during routine operations.

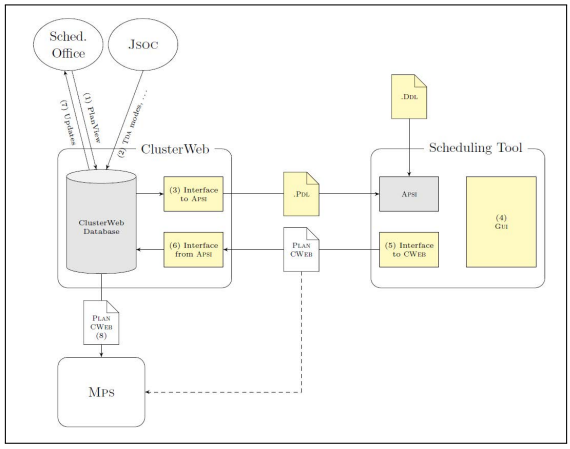


Figure 1: TIAGO Software Architecture

Architectural Design and Workflow

Fig. 1 displays the deployment of the planning tool in the overall system architecture. The scheduling tool interacts with the mission databases via the displayed interfaces. The workflow for creating ground station pass schedules with TIAGO is the following: (1) The mission planning process is initiated by the scheduling office which provides a first

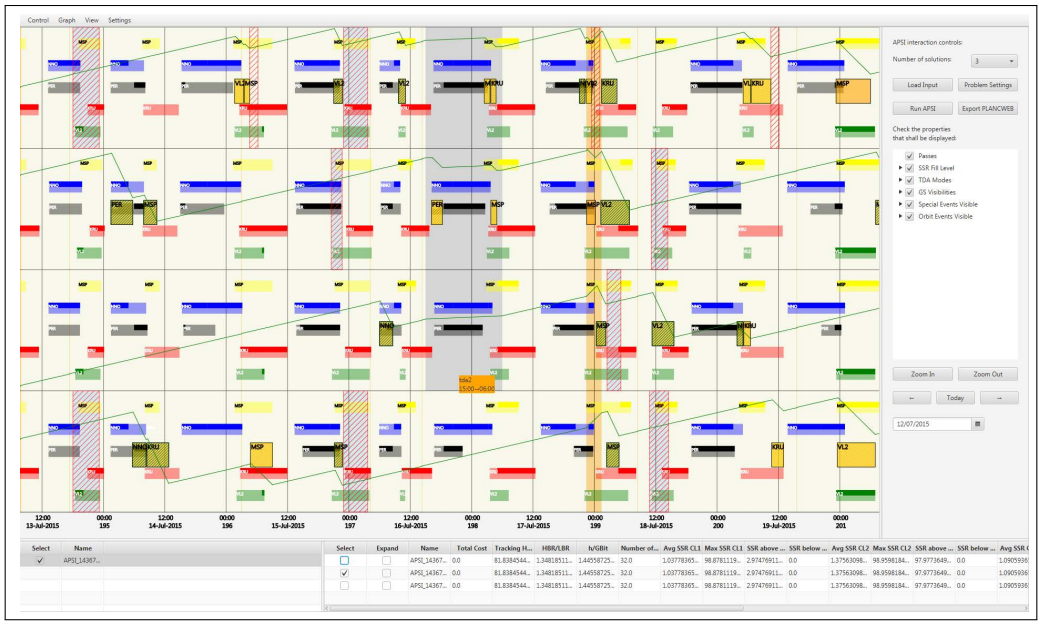


Figure 2: TIAGO's screenshot.

ground station allocation; (2) Scientific Operations planned at JSOC are provided in input; (3) With the information available in the mission database, the automated planning process can be started by converting the necessary data for the given time-frame into the PDL format and provide it to the planner; (4) The problem is imported into the planner and the solving process is started. Various allocation plan are generated and solutions are identified and selected by Cluster-II operators; (5) The selected solution is exported to the PlanCWeb[2] format; (6) The newly generated ground station plan is imported to the mission database (and can be accessed via the ClusterWeb frontend); (7) The ESA Ground Station scheduling office is informed about the new updates of the Cluster schedule.

User Interaction

As mission operations are constantly changing, the user should be able to modify certain settings and preconditions for different planning periods. For example the necessity of pre-heating before manoeuvres might be respected for some of the spacecrafts but not for others. This requires an high user interaction when creating the planning problem. The PDL file is in fact generated directly through the planning section of ClusterWeb. ClusterWeb is mainly developed in PHP and uses MySQL queries to access the Cluster database. The animation processes and validity checks of the form are implemented by using jquery and JavaScript functions. According to the selected options, the user interface initiates the pre-processing of the required data. The tool sends multiple queries to the ClusterWeb database and filters the information appropriately and writes it in a PDL file. The produced file is then downloaded from the ClusterWeb server, available for the further procedure of creating a ground station schedule.

The tool's main display (Fig. 2) is based on the representation of ground station pass scheduling in ClusterWeb. This includes the different science modes for the spacecraft, the SSR fill level, ground station visibilities and ground station bookings. Fig. 2 shows a period of 7 days during which 32 ground station passes have been allocated. Pass properties are represented by the different color patterns of the boxes. A low bit rate pass has a yellow pattern and a high bit rate pass has a green pattern, for example. Red patterns in the timeline of each spacecraft indicate periods where no pass can be scheduled. Hovering over the different elements of the schedule will provide the user with more detailed information about the event.

The right side pane contains elements to control the tool. The controls in the top initiate the solving process and the creation of the output file. The selectable options in the middle allow to manipulate the plan representation. With the multiple available options, the user is able to create a view which is suitable for him to identify good plan. The overview of the schedule can be for example increased by disabling the ground station visibilities which reduces the number of elements displayed in the schedule significantly.

The bottom pane displays the currently selected scenario and the computed solutions. It is also possible to display multiple solutions for a single scenario by selecting them. The expand option allows to highlight and enlarge one solution to inspect and compare it with the others. Multiple solutions have a different pass coloring. A numeric representation of plan properties enables the operator to judge on the quality of a plan based on different criteria. The columns

[2]PlanCWeb input format is an XML file containing the necessary information about all passes during a planning period in order to allow the Mission Planning System to create the operational sequences for commanding the spacecraft.

can be arranged in an arbitrary order and hidden if the user wishes so. The preferred user settings are saved automatically or can be exported if different views are desired.

Modeling Approach

TIAGO's underlying planning technology is based on constraint based temporal planning with timelines ((Muscettola 1994; Frank and Jonsson 2003; Fratini, Pecora, and Cesta 2008; Chien et al. 2012)). More precisely the planner is deployed on top of the APSI Framework using two classes of APSI modeling components: state variables and resources.

State Variables. State variables represent components that can take sequences of symbolic states subject to various (possibly temporal) transition constraints. This primitive allows the definition of *timed automata*; here the automaton represents the constraints that specify the logical and temporal allowed transitions of a timeline. A timeline for a state variable is valid if it represents a *timed word* accepted by the automaton. The timed automaton (or in the APSI case, the state variable) is a very powerful modeling primitive, widely studied (Alur and Dill 1994), and for which different algorithms exist to find valid timelines.

Resources. The second APSI primitive used in TIAGO is the resource. This is used to model any physical or virtual entity of limited availability, such that its profile represents its availability over time whereas a decision on the resource models a quantitative use/production/consumption of the resource over a time interval. Three types of resources are currently available in the APSI Framework: *reusable* resources abstract any real subsystem with a limited capacity, where an *activity* uses a quantity of resource during a limited interval and then releases it at the end. *Consumable* resources abstract any subsystem with a minimum capacity and a maximum capacity, where *consumptions* and *productions* consume and restore a quantity of the resource in specific time instants. The third type of resources is the *linear reservoir* resource. This resource does not have a stepwise constant profile of consumption like reusable and consumable ones, but the activities specify the amount of production/consumption per time, namely `slope`, resulting in a profile of resource that is linear in time. As a consequence the amount of resource available at each transition of the timeline depends on the duration of the time intervals over which this production or consumption has been performed (conversely with the other types of resource where the profile of the resource availability at each transition depends only on when and how much is produced/consumed and not on the duration of the production/consumption). And this in turn induces the need for integrated reasoning on time and data to guarantee the flexibility of the plan/schedule generated.

Synchronizations. In timeline-based modeling the physical and technical constraints that influence the interaction of the sub-systems (modeled either as state variables or resources) are represented by temporal and logical synchronizations among the values taken by the automata and/or resource allocations on the timelines. Conceptually these constructs define valid schema of values allowed on timelines and link the values of the timelines with resource allocations. In particular they allow the definition of Allen's relations like quantitative temporal relations among time points and time intervals as well as constraints on the parameters of the related values (Allen 1983). From a planning perspective, the synchronizations define the cause-effect relationships among the states of the system modeled, describing how a given status can be achieved.

TIAGO Domain and Problem Modeling

The problem of ground station scheduling for the four Cluster-II satellites is an example for resource driven timeline-based planning for space application. The main objective of scheduling ground station passes for Cluster-II is to keep the on-board memory of the spacecraft as low as possible to avoid the overwriting of valuable scientific data. The Solid State Recorder is modeled as a reservoir resource with a minimum and a maximum fill level. Different science modes are represented as states on a timeline for each spacecraft and give information about the data production rate of the satellite's scientific instruments. The science modes are synchronized to the SSR timeline of the different spacecraft and increase the fill level with a fixed positive activity according to the different modes. Below an example of DDL.3 synchronization that models this cause-effect relationship among spacecraft modes and SSR consumption (resource consumptions are in bit/second, positive in this case as the mode produces data in the SSR).

```
SYNCHRONIZE tda_modes.clu1_timeline{
  VALUE tda4(){
    cd1 ssr_fill_level.clu1_ssr_timeline.ACTIVITY(21845);
    EQUALS cd1;
  }
}
```

On the other hand, the SSR fill level can be decreased by means of downlink activities during the ground station passes assigned to a spacecraft. The ground station visibilities can be further divided into visibilities where no dump is possible due to bad signal to noise ratio, visibilities with low signal to noise where data can only be downlinked in low bit rate and visibilities with a high signal to noise ratio where high bit rate downloads are possible. It is also possible to have a change in the allowed downlink rate during a visibility and a pass. Ideally, the amount of high bit rate passes should be as high as possible to achieve a higher data return with equal or smaller tracking hours. But, before a ground station visibility can be assigned to one of the Cluster-II spacecrafts, it must be ensured that no other mission is using the ground station at the given time. This requires then to consider all ground station bookings and to plan against actual available windows of opportunity instead of against station visibilities.

These constraints are represented with a set of synchronizations stating that passes have to be placed during station's visibilities and that a pass can be decomposed into a set of activities that consume SSR (see Figure 3). In the following synchronization, for instance, the pass for `clu_1`,

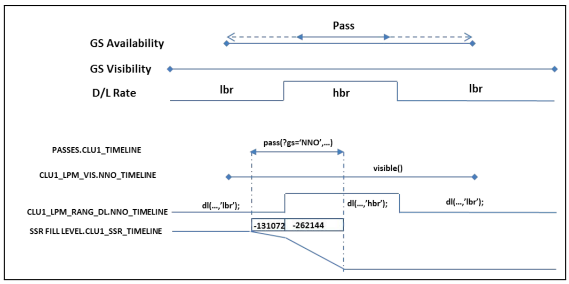


Figure 3: Pass and SRR Fill Level

placed on the New Norcia ground station (NNO) on a low bit rate is synchronized as: (1) it contains an activity on the SSR reservoir resource that download data with a low bit rate; (2) the pass is during the NNO low bit rate availability; (3) some temporal constraints holds between passes and science modes of the satellite to state that a pass should stay 6 to 15 minutes "away" from a science mode switch; (4) it has to be synchronized with other Cluster satellites bookings and pre-booked time windows, hence an activity on a timeline has to be generated (and scheduled) to guarantee that the station is not over booked.

```
SYNCHRONIZE passes.clu1_timeline
  {
    VALUE pass(NNO,LPM,?mspa,lbr)
    {
(1)   LBR ssr_fill_level.clu1_ssr_timeline.ACTIVITY(-131072);
      CONTAINS[00:09,00:09][00:01,00:01] LBR;

(2)   LBR_V clu1_lpm_rang_dl.nno_timeline.dl(?ul_bool_1,lbr);
      DURING LBR_V;

(3)   cd_start <?> tda_modes.clu1_timeline.*;
      cd_end <?> tda_modes.clu1_timeline.*;
      STARTS_DURING[00:00,+INF][00:15,+INF] cd_start;
      ENDS_DURING[00:06,+INF][00:15,+INF] cd_end;

(4)   cd_book <!> gs_bookings.nno_timeline.clu_booked();
      EQUALS cd_book;
    }

    ... OTHER TYPES OF PASS;\
  }
```

Furthermore, configuration jobs of the ground station stated in the Section Mission Scenario need to be considered before and after a pass. To represent the configuration time, intermediate states are introduced on the booking timeline of the ground station and transition rules are specified, hence the state variable in Figure 4 models the ground station booking logic. In this state variable, external bookings must be followed by a configuration activity with a duration of 45 minutes and only after that a Cluster pass can be booked. Additionally a Cluster pass must be followed by another configuration activity of exactly 15 minutes before the timeline can take another state. An exception can be made when a second Cluster pass of a different spacecraft follows. Then a configuration time of exactly 10 minutes between the two passes must be considered.

Modeling Issues

The modeling primitives shown above allow the representation of traditional integrated planning and scheduling

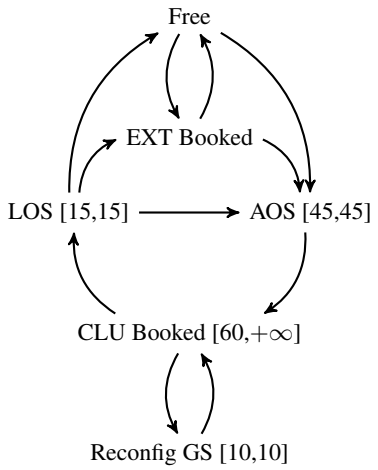


Figure 4: Ground station bookings state varible.

problems by modeling, for instance spacecrafts modes and passes as state variable values, resource requirements, and synchronized statements on corresponding resources. However there are important aspects of the problem that can't be modeled only by means of synchronizations.

The first problem was the need to state goals directly on the resource profiles to force not only to be under the limit (normal scheduling process) but also to verify some properties along the plan, like being as low as possible before eclipse events. Another problem was how to efficiently specify that the real objective in this scenario is to allocate passes during ground station visibility windows in order to empty the Solid State Recorder. The domain model contains both synchronizations associated to data productions (science modes) and synchronizations on data consumption (passes), but there is no logical connection that says that an SSR over consumption has to be solved by allocating passes. These aspects were discussed in our previous work (Fratini et al. 2015) and are below summarized.

The first issue has been solved by introducing the *resource level request*. This feature allows the modeler to specify some desiderata directly on the resource level, as minimum and maximum value required in a given instant. It is substantially different from a resource allocation: in fact it has no direct effect on the resource profile like productions and consumptions, but it raises a flaw in the plan when the condition is not fulfilled.

This is the case of Cluster-II where it needed to empty the SSR before a spacecraft is entering a period of eclipse (where is not possible to communicate). This goal is stated synchronizing the value modeling the eclipse with a resource requirement as it follows:

```
SYNCHRONIZE passes.clu1_timeline {
  VALUE no_pass_possible(?reason=eclipse_ssr_off){
    req ssr_fill_level.clu1_ssr_timeline.level(0.0,0.0);
    STARTS_AT req;
  }
}
```

It is also worth to point out that a resource level request,

as any other statement on timelines, can be used as an high level goal for the planner. In the case of Cluster-II, it has been used to cope with the presence of eclipses.

For what concerns the second problem, in the Cluster-II case the objective is to allocate a sufficient number of passes during ground station availability in order to avoid SSR's overwriting. To cope with this issue, we have introduced the concepts of *producer/consumer* and *opportunity* in the model. These concepts allow to make explicit the information (that is in any case already modeled in the synchronizations as well) as suitable for the efficiency of the solver that can then take its decisions in a more informed way.

A producer (consumer) is a value of a state variable that can lead to a production or consumption of resource. A producer (consumer) value has not a specified production (consumption), but only bounds derived from all the possible ways it can be planned. This information can be calculated automatically by pre-processing the model or, simply, via the presence of tags in the model. In our example, it is possible to deduce that a pass can consume an amount of resource between a minimum of 131072 b/s (when completely synchronized with a low bit rate opportunity) and 262144 b/s (when completely synchronized with an high bit rate opportunity). These bounds provide some early and useful information to allocate consumptions on the SSR.

An opportunity is an uncontrollable timeline that provides, through one of more of its values, temporal slots to allocate planning decisions (other state variable values or activities). When these timelines are completely specified and static, this information can be pre-calculated analyzing the domain and the problem: if a value v of a state variable (or an activity for a resource) is synchronized during a value o of a static timeline, than o provides an opportunity for v. This information can be also obtained by the presence of tags in the model.

It is worth remarking that this information, being only necessary but not sufficient to allocate a pass, can only be used to bound the decision process. Synchronizations will then have to be planned to find the actual temporal position of the passes. In the Cluster scenario, the information on producers/consumers and opportunities is exploited, in a domain independent way, to allocate a set of flexible passes. These passes lead to an envelope of possible SSR allocation profiles, to be refined by subsequent planning and scheduling steps, as described in the following section.

Problem Solving

In the APSI framework, an application, being a generic planner and/or scheduler or a domain specific deployed application is designed as a collection of *solvers*. Based on the constraint-based paradigm, the search space of an APSI solver is made of planning and scheduling statements on timelines and temporal and data relations among them. An application solves a conceptually more complex problem, like a planning problem for instance, or a global resource optimization problem. The main difference between a solver and an application consists in the fact that a solver searches in the space of decisions and constraints, while an application searches in the space of solvers solutions.

In the Cluster-II scenario we have used (and improved) the PLASMA (PLAn Space Multi-solver Application) planner. It is made of a collection of solvers, activated on a *flaw detection base*: when a flaw is detected on a timeline, the planner activates an available list of solvers that can try to fix the problem (De Maio et al. 2015).

PLASMA incorporates the principles of Partial Order Planning (POP, (Weld 1994)), like a plan-space search space and the least commitment approach. Starting from an initial partial plan only made of partially specified timelines and goals to be achieved, the planner iteratively refines it into a final plan that is compliant with all the requirements expressed by the goals.

The PLASMA planner is grounded on different solvers to solve various flaws during the planning process:

- Partial Order Scheduler. It supports the scheduling process resulting from planning to guarantee temporal flexibility to the plan. In this domain is not possible to schedule only on fix-times, because the process iteratively interleaves planning and scheduling steps;
- Resource Production Allocator. It makes sure that linear resource can be adequately managed by avoiding any over-consumption;
- MaxFlow Resource Profile Bounder. Its task consists in bounding position and duration of a set of activities (consumption activities in the case of Cluster-II) which assures that all the resource constraints and/or requirements are consistent.

Partial Order Schedules

PLASMA exploits the time-flexibility properties of Partial Order Schedules (Policella et al. 2007) to support the planning process and to guarantee temporal flexibility to the plan and schedule (necessary for execution). While the POS concept was originally introduced to cope with uncertainty and provide robust solutions, a similar need is present when considering an integrated planning and scheduling approach where iteratively a planner and a scheduler are called to modify a shared representation. In fact, the schedule produced should be able to accommodate possible changes introduced by the planner preserving sufficient flexibility.

It is worth recalling the POS is a set of activities partially ordered such that any possible complete order that is consistent with the initial partial order, is a resource and time feasible schedule. Therefore the planner, when moving inside the space identified by POS, will have the possibility of modifying the temporal allocation preserving the feasibility of the overall solution.

Allocating Resource Productions

The task of the Resource Production Allocator is to identify those time-windows among the available one which can be used to compensate the different over-consumption on the different linear resources, that is, *to produce resource availability*. It is worth remarking that this solver does not produces directly consumption activities (that are synchronized with ground station passes in our specific case), but rather identifies a set of opportunities where these station passes

should be allocated. In other words the solver produces a set of requests/goals for the next solver. It is the task of the next planner's solver to plan in detail these consumption activities by considering the given opportunities but also all those aspect not considered by the Resource Production Allocator such as synchronization of consumption activities with other timelines (in our case, for instance, the bit-rate timeline), logical consistency with respect to the spacecraft timelines, etc.

For instance the solver below described, the MaxFlow Resource Profile Bounder, generates a final set of consumption activities by considering as input the set of consumption opportunities together with bit-rate bounds and the data production activities.

Managing Linear Reservoir via MaxFlow

As mentioned above, in this domain, one of the main driving feature is the control of the resource level. In fact in this situation we have not only capacity constraints to be held but also specific resource levels can be requested in order to allocate some activities.

To evaluate the resource level a timeline is taken into account. The latter represents an ordered sequence of not-overlapping intervals which have:

- flexible start and end time
- a fixed data slope (positive in case of data generation, negative in case of consumption)
- resource preconditions which establish the minimum and maximum feasible resource level - these consist in a specific resource level request. In case there is no specific request the two levels are equivalent to the resource capacity constraints.

Given this information, the goals are (1) to detect possible inconsistencies and, when consistent, (2) to find a possible solution which consists in deciding a consistent duration for all the time intervals in the timeline. In order to satisfy these goals, the problem has been modeled as a particular flow network, a Maximum Flow with Edge Demands Problem.

Given then Maximum Flow with Edge Demands Problem representation, if this does not admit a solution then the initial timeline is not consistent. On the other side, given a max-flow solution of the problem, the duration of each time interval can be computed by considering the different flows through the network. From a max flow solution, the solver either generates a set of temporal constraints to reduce activities' durations or, if this is not possible, generates a new production and restart the algorithm.

TIAGO: Current Status and Future Work

At this stage a first version of the TIAGO tool has been completely developed. As mentioned above a driving factor considered during the development of the application was to re-use as much as possible domain independent planning & scheduling technologies to prove the feasibility of the AI approach. With this aim, during the development phase, the system has been intensively tested with several real-case problem benchmarks.

These tests have shown that TIAGO can return different alternative solutions in the order of a few minutes (worst case) for problem instances defined as challenging by the Cluster-II Flight Control Team (big data production periods, with many ground stations already allocated to other missions and various periods where no passes can be allocated). Moreover it has been assessed that the tool is flexible enough with respect to changes of the domain model and can support the optimization of the overall pass duration.

The tool has recently been delivered to the Cluster-II Flight Control Team to go through an operational evaluation phase. We expect that this, still on-going, phase will generate new requirements/goals and spot the limitations of the current approach. An already collected feedback is that the current version of TIAGO, while reduces the total pass time (compared to the manually generated solutions), does not include other aspects like reducing the number of passes (instead of the total duration) or minimizing the usage of the on-board memory. The latter is probably the major contributor to the robustness's solution: in fact a lower lever of used resource allows to better face unpredictable situations like the loss of a pass.

Future work will therefore aim at evolving the current solving approach in order to (1) generate more robust solutions and (2) consider multiple optimization criteria beside the total dumping time.

In particular, in the current approach the on-board memory is managed with a least commitment approach based on a max-flow solver. While it assures that the resource usage is always consistent, this approach is not sufficient to cope with the requirement of minimizing the resource level while balancing at the same time the ground station total allocation time. Future versions of TIAGO will need to overcome this limitation managing the resource level in a more adequate way. Possible solutions could be to generate resource allocation requests during the planning period to reduce the average SSR allocation, or to plan with a resource capacity that is actually a percentage of the real one.

Conclusions

TIAGO is a tool based on AI planning and scheduling technologies developed for automated ground station pass planning and optimization on the ESA Cluster-II mission. TIAGO in its initial design uses a set of domain independent planners and schedulers as core solving process. As a methodological approach we have chosen to address the problem entirely with a domain independent planner, in order to fulfill the need of the users in being free of modifying the model and changing the constraints.

TIAGO is also designed to be fully integrated in the Cluster-II ground segment software, and it is currently under operational evaluation to define optimization criteria and plan quality measurement (to be added to the planner as heuristics) for orienting TIAGO from a pass allocation tool into a pass optimization tool.

References

Allen, J. 1983. Maintaining knowledge about temporal intervals. *Communications of the ACM* 26(11):832–843.

Alur, R., and Dill, D. L. 1994. A theory of timed automata. *Theor. Comput. Sci.* 126(2):183–235.

Bensana, E.; Lemaitre, M.; and Verfaillie, G. 1999. Earth Observation Satellite Management. *Constraints: An International Journal* 4(3):293–299.

Cesta, A.; Cortellessa, G.; Denis, M.; Donati, A.; Fratini, S.; Oddi, A.; Policella, N.; Rabenau, E.; and Schulster, J. 2007. MEXAR2: AI Solves Mission Planner Problems. *IEEE Intelligent Systems* 22(4):12–19.

Cesta, A.; Cortellessa, G.; Fratini, S.; Oddi, A.; and Bernardi, G. 2011. Deploying interactive mission planning tools - experiences and lessons learned. *JACIII* 15(8):1149–1158.

Chien, S.; Rabideau, G.; Knight, R.; Sherwood, R.; Engelhardt, B.; Mutz, D.; Estlin, T.; Smith, B.; Fisher, F.; Barrett, T.; Stebbins, G.; and Tran, D. 2000. ASPEN - Automating Space Mission Operations using Automated Planning and Scheduling. In *SpaceOps-00. Proceedings of the 6^{th} International Conference on Space Operations*.

Chien, S.; Johnston, M.; Frank, J.; Giuliano, M.; Kavelaars, A.; Lenzen, C.; and Policella, N. 2012. A generalized timeline representation, services, and interface for automating space mission operations. In *Proceedings of the 12th International Conference on Space Operations, SpaceOps*. AIAA.

De Maio, A.; Fratini, S.; Policella, N.; and Donati, S. 2015. Resource driven planning with "plasma": the plan space multi-solver application. In *ASTRA 2015. 13^{th} Symposium on Advanced Space Technologies in Robotics and Automation*.

EUROPA. 2008. Europa Software Distribution Web Site. http://code.google.com/p/europa-pso/wiki/EuropaWiki.

Frank, J., and Jonsson, A. 2003. Constraint based attribute and interval planning. *Journal of Constraints* 8(4):339–364.

Fratini, S., and Cesta, A. 2012. The APSI Framework: A Platform for Timeline Synthesis. In *Proceedings of the 1st Workshops on Planning and Scheduling with Timelines at ICAPS-12, Atibaia, Brazil*.

Fratini, S.; Policella, N.; Faerber, N.; Donati, A.; Sousa, B.; and De Maio, A. 2015. Resource Driven Timeline-Based Planning for Space Applications. In *IWPSS-15. Proceedings of the 9^{th} International Workshop on Planning and Scheduling for Space*.

Fratini, S.; Pecora, F.; and Cesta, A. 2008. Unifying Planning and Scheduling as Timelines in a Component-Based Perspective. *Archives of Control Sciences* 18(2):231–271.

Johnston, M., and Giuliano, M. 2011. Multi-Objective Scheduling for the Cluster II Constellation. In *Proceedings of the 7^{th} International Workshop on Planning and Scheduling for Space, IWPSS11*.

Knight, R.; Rabideau, G.; Chien, S.; Engelhardt, B.; and Sherwood, R. 2001. CASPER: Space Exploration through Continuous Planning. *IEEE Intelligent Systems* 16(4):70–75.

Muscettola, N.; Nayak, P. P.; Pell, B.; and Williams, B. C. 1998. Remote agent: To boldly go where no ai system has gone before. *Artificial Intelligence* 103(1-2):5–47.

Muscettola, N. 1994. HSTS: Integrating Planning and Scheduling. In Zweben, M. and Fox, M.S., ed., *Intelligent Scheduling*. Morgan Kauffmann.

Policella, N.; Cesta, A.; Oddi, A.; and Smith, S. F. 2007. From Precedence Constraint Posting to Partial Order Schedules. *AI Communications* 20(3):163–180.

Policella, N.; Oliveira, H.; and Benzi, E. 2013. Planning spacecraft activities: An automated approach. In *ICAPS-13. Proceedings of the 23^{rd} International Conference on Automated Planning & Scheduling*.

Weld, D. S. 1994. An introduction to least commitment planning. *AI Magazine* 15(4):27–61.

Automatic Resolution of Policy Conflicts in IoT Environments Through Planning

Emre Goynugur[§] and **Kartik Talamadupula**[†] and **Geeth de Mel**[‡] and **Murat Sensoy**[§]

[§]**Ozyegin University**
Department of Computer Science
Istanbul, Turkey
emre.goynugur,murat.sensoy @ ozyegin.edu.tr

[†]**IBM Research**
T.J. Watson Research Center
Yorktown Heights, NY, USA
krtalamad @ us.ibm.com

[‡]**IBM Research**
Daresbury Laboratory
Warrington, UK
geeth.demel @ uk.ibm.com

Abstract

The Internet of Things (IoT) is a highly agile and complex environment managed via the Internet. The management of such an environment requires robust automated mechanisms, since manual curation becomes a prohibitively expensive and near-impossible task. Motivated by this observation, we present a mechanism to resolve conflicts among the services provided by IoT devices in such environments, by reformulating conflict resolution as a planning problem. Specifically, we propose to use planning with soft constraints (preferences) to resolve conflicts that arise when fulfilling multiple goals using varied services in an IoT environment. We build on previous work on creating a semantic policy framework for detecting conflicts within an IoT environment, and present a proof-of-concept implementation of our approach to demonstrate its promise in the IoT space.

1 Introduction

The Internet of Things (IoT) is a highly agile (sensitive to availability, connectivity, and so forth) and complex (i.e., multitude of cross-connected devices) environment managed via the Internet. IoT promises a paradigm shift in which a multitude of internet-enabled devices – and the services provided by them – are seamlessly meshed together such that end-users can experience improved situational awareness (e.g., "*Your usual route home has a 30 min. delay*"), added context (e.g., "*An accident at Broadway and 8th*"), and so forth to effectively and efficiently function in the environment. Due to advances in technology, the promise of IoT is fast becoming a reality, and in recent years there has been strong evidence in both the research and commercial sectors showing the applicability of this technology in multiple domains (Vermesan and Friess 2011; Iera et al. 2010; Gubbi et al. 2013). Furthermore, software systems and digital assistants such as IBM's Watson, Apple's Siri, Google's Now, and Microsoft's Cortana – all utilizing IoT enabled services – are fast becoming the go-to assistants for a variety of users. There is evidence to suggest that more and more users are tasking services provided by IoT ecosystems with automatically informing them about their environments (Holler et al. 2014; Li et al. 2011).

However, with growing adoption and deployment, the complexity of such IoT systems is fast growing. Such complexity is exacerbated in urban environments where large human populations will deploy ubiquitous devices (and in turn services), consume them, and interact with such systems to fulfill daily goals by meshing IoT services with external services such as location, weather and so forth. According to the United Nations, 54% of the world's population lives in cities – a number that will increase to 66% by 2050 (Jayarajah et al. 2015). This will introduce many new services into IoT environments, and put a huge strain on the management of such systems. Furthermore, due to this explosion in services, conflicts are bound to occur more frequently, especially unexpected ones. For example, a mobile application may instantiate a meshed service that uses a location service which is prohibited in some locations. Manually, curating such exceptions is impractical, if not impossible; thus, an emerging requirement in IoT environments is intelligent management of services, handling of exceptions, and the provision of automatic resolution techniques, so that the cognitive overload on the user is reduced.

Policies are typically used in system design to specify *obligations* and *prohibitions*, and automatically handle exceptions when obligations and prohibitions overlap due to unforeseen situations. As IoT systems gain more complex capabilities – especially capabilities to learn, reason, and understand their environments and user needs – one can consider applying policy techniques to handle conflicts found in the IoT environment. However, the dynamism of IoT environments when compared with traditional systems must be taken into account; this necessitates more intelligent automation techniques for policy conflict management.

In our recent work, we have created a semantic policy framework to efficiently detect conflicts within an IoT environment by restricting the expressivity of OWL-POLAR (Sensoy et al. 2012) to OWL-QL (Fikes, Hayes, and Horrocks 2004). In this paper, we present our initial work and thoughts on automatically handling these conflicts by reformulating the conflict resolution problem as an automated

planning problem. Specifically, we formulate policy conflicts as preferences to enable the use of preference-based planning techniques to automatically resolve such conflicts to the best extent possible.

The rest of the paper is organized as follows: Section 2 provides the motivation for our work by means of use-cases. In Section 3 we formalize the policy representation and conflict detection, and define a number of terms that will be used in the paper. Section 4 presents the use of automated planning techniques in our system to automatically resolve conflicts, and other extensions that could be explored. In Section 5 we briefly discuss a preliminary proof-of-concept evaluation of our current system, and in Section 6 we discuss related literature. We conclude the document in Section 7 by discussing future work and by providing final remarks.

2 Motivation

When compared with traditional IT systems, one of the major issues in managing IoT-based systems is the impracticality of using humans to configure, maintain, and manage all these connected devices, and the services associated with them. This is because services related to IoT are dynamic (especially in terms of availability), agile, and context sensitive. Gartner, Inc. forecasts that 6.4 billion connected things will be in use worldwide in 2016, up 30 percent from 2015; this number will reach 20.8 billion by 2020 (van der Meulen 2015). This rapid increase in device numbers precipitates the need for an expressive and efficient policy framework, one which would allow its users to define high level rules that can be refined to individual devices. As there are a lot of diverse products from different manufacturers, it is difficult to tailor rules to cover all these individual devices. In addition to creating rules for device X or product type Y, users should be able to create rules by describing device properties or capabilities (e.g. devices with displays or devices that can play sound, and so forth).

Furthermore, these connected devices will be used by different people who have different preferences, habits, or expectations. It is safe to assume that these users together will generate a large number of policies and there will be more than one policy applying to a specific device; indeed, it is essential that multiple policies apply to a device in order to cover the diversity of management functions and of management domains (Lupu and Sloman 1999). However, whenever multiple policies apply to an object, there is usually a potential for some form of conflict.

For example, let us assume that in a smart home environment there are two policies: if someone rings the doorbell, then there should be a notification; and if the baby is sleeping, devices should not make any sound. In this scenario, a conflict occurs if someone rings the doorbell while the baby is sleeping. The default action for a doorbell is to make sound to notify the household; however, if it fulfills this goal, it will violate the "no-sound" policy. In a truly connected environment, the notify action could be delegated to another device which uses different notification methods such as displaying a visual message, sending an SMS, and so on. Through the rest of the paper, we use this simple scenario as a running example.

Our policy framework is based on OWL-QL, which offers efficient reasoning mechanisms, can handle large numbers of policies, and detect conflicts before they actually happen; however, it cannot resolve or offer solutions for these conflicts. Users cannot be expected to manually resolve all such conflicts. In order to automate the process of conflict resolution and to address the missing features of our framework, we believe that automated planning techniques can be used.

3 Problem Definition

In this section, we formalize the representation of policies, and discuss the conditions for policy activation, expiration, and conflicts. We first introduce the terminology that will be used in the rest of this paper to represent policies.

3.1 Policy Terminology

As our method is based on an OWL-QL ontology, we use the terms *OWL-QL ontology* and *knowledge base* synonymously. In our context, a QL knowledge base (KB) consists of a *TBox* and an *ABox*. Concepts, properties, and axioms that describe relationships between concepts form the TBox of an ontology. We borrow syntax and semantics from the DL-Lite family to illustrate our TBox (Calvanese et al. 2007). For example, the statement: $Computer \sqsubseteq ElectronicDevice$ means that $Computer$ class is a subclass of $ElectronicDevice$; and the statement $ElectronicDevice \sqcap \exists playSound$ represents devices that can play sound. In PDDL terms, a TBox could be considered as the collection of types, type hierarchy, predicates, and derived predicate rules.

On the other hand, an ABox is a collection of extensional knowledge about individual objects, such as whether an object is an instance of a concept, or two objects are connected by a role (Artale et al. 2009). In Description Logic, roles are binary relations between two individual objects; these can be considered binary PDDL predicates in our context: e.g. $livesIn(John, NewYork)$. In this statement, $livesIn$ is the role that connects $John$ and $NewYork$. In PDDL, objects and predicates that exist in a planning state could be considered as an ABox—e.g. initial and goal states represent two different ABoxes.

3.2 Policy Representation

In this work, we formalize a policy as a six-tuple (α, N, $\chi : \rho$, $a : \varphi$, e, c) where:

- α is the activation condition of the policy;
- N is either obligation (O) or prohibition (P);
- χ is the policy addressee and ρ represents its roles;
- $a : \varphi$ is the description of the regulated action; a is the variable of the action instance and φ describes a;

- e is the expiration condition; and
- c is the policy's violation cost.

In a policy, ρ, α, φ, and e are expressed using a conjunction of query atoms. A query atom is in the form of either $C(x)$ or $P(x, y)$, where C is a concept, P is either a object or datatype property, x is either a variable or individual, and y may be a variable, an individual, or a data value.

For instance, using variables b and f, the conjunction of atoms $Baby(?b) \wedge Sleeping(?b) \wedge inFlat(?b, ?f)$ describes a setting where there is a sleeping baby in a flat. Concepts, properties, and individuals used in the definition of a policy should come from the underlying OWL-QL knowledge base. An example TBox of an OWL-QL ontology is shown in Table 1.

Table 1: An example TBox for an OWL-QL ontology.

An OWL-QL TBox
$Sleeping \sqsubseteq State$
$Awake \sqsubseteq State$
$Awake \sqsubseteq \neg Sleeping$
$Baby \sqsubseteq Person$
$SoundNotification \sqsubseteq Sound \sqcap Notification$
$TextNotification \sqsubseteq Notification$
$Speaker \sqsubseteq Device$
$Doorbell \sqsubseteq Device$
$\exists playSound \sqsubseteq Device$
$PortableDevice \sqsubseteq Device$
$MobilePhone \sqsubseteq PortableDevice$
$\exists hasSpeaker \sqsubseteq \exists playSound$
$MediaPlayer \sqsubseteq \exists playSound$
$TV \sqsubseteq \exists hasSpeaker \sqcap \exists hasDisplay$
$MakeSound \sqsubseteq Action \sqcap \exists playSound$
$Notify \sqsubseteq Action$
$NotifyWithSound \sqsubseteq MakeSound \sqcap Notify$
$Baby \sqsubseteq Person$
$SomeoneAtDoor \sqsubseteq Event$

3.3 Policy Activation and Expiration

A policy is activated for a specific set of instances that fulfill its activation condition. Likewise, an active policy instance expires if its expiration condition holds true or the goal of that policy is fulfilled. For instance, in our scenario, the activation condition for the policy in Table 2 holds for the binding $\{?d = dbell, ?b = John, ?f = flt\}$. As a result, an activated policy instance is created with this binding;"*dbell is prohibited to perform MakeSound action until John is awake*". Whenever the expiration condition of an active policy instance holds, that policy should be removed; e.g., the activated policy expires if the baby John wakes up.

Some active policies also expire when they are satisfied. For instance, obligation policies can expire after obligations are fulfilled. Let us consider the policy example in Table 3, which defines an obligation policy stating that a doorbell has to notify adult residents of a flat if someone rings the bell.

Table 2: An example prohibition policy.

$\chi : \rho$	$?d : Device(?d)$
N	P
α	$Baby(?b) \wedge Sleeping(?b) \wedge inFlat(?b, ?f) \wedge inFlat(?d, ?f)$
$a : \varphi$	$?a : MakeSound(?a)$
e	$Awake(?b)$
c	10.0

Table 3: An example obligation policy.

$\chi : \rho$	$?d : Doorbell(?x)$
N	O
α	$SomeoneAtDoor(?e) \wedge producedBy(?e, ?x) \wedge$ $belongsToFlat(?x, ?f) \wedge hasResident(?f, ?p) \wedge Adult(?p)$
$a : \varphi$	$?a : NotifyWithSound(?a) \wedge hasTarget(?a, ?p)$
e	
c	4.0

In this case, since Bob rings the bell $dbell$, the active policy "*dbell is obliged to notify an adult resident of the flat with sound*" should be created. After notifying the targeted person in the flat, the obliged action would be performed and the activated policy would be satisfied. Alternatively, there could be an expiration condition to keep that policy instance active until someone explicitly acknowledges the notification or the door is opened.

3.4 Policy Conflicts

In our work, three conditions have to hold true for two policies to conflict. First of all, these policies should be applied to the same policy addressee, e.g., same device or individual. Second, one policy must oblige an action, while the other prohibits the same action. Third, these two policies should be active at the same time in a consistent world state according to the underlying ontology. This situation forces the addressee to make a decision and violate one of the policies. It is important to state here that unless an addressee has to violate one of its own policies to fulfill another one, there is no conflict.

For instance, in our scenario, the doorbell is obliged to notify the household with sound due to one policy while the very same doorbell is prohibited to make any sound due to another policy. As it has to violate its own prohibition policy to fulfill its goal policy, these policies are considered to be in conflict. Let us note that the subsumption relation between the make sound and notify with sound actions are explicitly defined in the TBox. Table 4 illustrates a state in which the policies in Table 3 and Table 2 are in conflict.

It is almost trivial to figure out policy conflicts within a specific state of the world, as in our example. However, it is non-trivial at design time to reason if two policies will

Table 4: ABox representing the sandbox.

1	$Device(d0)$
2	$Doorbell(d0)$
3	$Baby(b0)$
4	$Sleeping(b0)$
5	$inFlat(b0, f0)$
6	$inFlat(d0, f0)$
7	$SomeoneAtDoor(e0)$
8	$producedBy(e0, d0)$
9	$belongsToFlat(d0, f0)$
10	$hasResident(f0, p0)$
11	$Adult(p0)$

ever get into conflict. Our policy framework *can* detect such conflicts during design time, but we refrain from discussing that capability in this paper; instead, we focus on conflict resolution.

4 Using Planning for Conflict Resolution

Policy conflicts may arise between two given policies when the conditions outlined in the previous section are met. In such cases, it is essential for the system to devise a way to resolve the conflict and move forward. In this section, we outline a way of posing this conflict resolution problem as a planning problem, and using automated planning technology to solve that problem instance.

4.1 Representing OWL-QL in PDDL

Our policy framework exploits OWL-QL to cope with very large volumes of instance data. OWL-QL is less expressive compared to other OWL languages; however, it makes the implementation of conjunctive query answering using relational databases possible. Furthermore, we can map the concepts and relations of our knowledge base to PDDL using derived predicates.

Table names and their fields in our database can be mapped as predicates in a PDDL domain file. The instance data that exist in the database would represent the initial state of the system. In this context, a predicate is in the form of either $C(x)$ or $P(x, y)$, where C is a concept, P is either a object or datatype property, x is either a variable or individual, and y may be a variable, an individual, or a data value.

However, directly querying the knowledge base does not reveal the inferred information that may be deduced through the TBox. For this purpose, OWL-QL uses query rewriting to expand queries. Also, consistency checking is done by a disjunctive query that consists of conditions that might cause inconsistency based on the axioms in the TBox (Artale et al. 2009).

PDDL's typing feature allow us to encode simple class hierarchies into the domain file. However, typing alone is not sufficient to express multiple inheritance and subclass expressions with object or data properties, e.g. $TV \sqsubseteq \exists hasSpeaker \sqcap \exists hasDisplay$. For this reason, instead of using the typing support of PDDL, we represent type(s) of an object with predicate(s). Hence, all these reasoning formulas can be integrated into the PDDL domain file by either rewriting action preconditions or using derived predicates. We believe that the latter approach is a cleaner solution rather than filling up action predicates with disjunctions.

Finally, OWL-QL doesn't offer support for numerical constraints to provide efficient reasoning: e.g. it is not possible to express that one object can only be in one place or that a room can only have one temperature at a given time. PDDL can also help compensate for this limitation by embedding numerical constraints into the PDDL domain file. Below, we give an example of a quasi-PDDL domain and its contents as it relates to our application[1].

```
(define (domain iot)
(:requirements :adl :derived-predicates)

(:predicates
  (Action ?action)
  (Awake ?person)
  (Baby ?baby)
  (Device ?device)
  (Display ?display)
  (Doorbell ?doorbell)
  (Event ?event)
  (MakeSound ?action)
  (MediaPlayer ?MediaPlayer)
  (MobilePhone ?mobile)
  (Notification ?notfication)
  (Notify ?action)
  (NotifyWithSound ?action)
  (PortableDevice ?device)
  (Sleeping ?person)
  (SomeoneAtDoor ?event)
  (Sound ?sound)
  (SoundNotification ?soundNotification)
  (Speaker ?speaker)
  (State ?state)
  (TextNotification ?textNotification)
  (TV ?tv)
  (gotNotifiedFor ?person ?event)
  (hasDisplay ?device ?hasDisplay)
  (hasSpeaker ?device ?speaker)
  (playSound ?device ?soundAction)
  (isConsistent))

(:derived (SoundNotification ?s)(
  (and (Sound ?s) (Notification ?s))))

(:derived (TextNotification ?t)(
  (Notification ?t)))

(:derived (TV ?tv)
```

[1]Note that this is not a complete and correct PDDL specification that a planner can parse.

```
  (exists (?unbound_1 ?unbound_2) (and
      (hasDisplay ?tv ?unbound_1)
      (hasSpeaker ?tv ?unbound_1)))
)

(:derived (MakeSound ?action)
  (and Action(?action) (playSound ?action TRUE))
)

(:derived (NotifyWithSound ?notify)
  (and (MakeSound ?notify) (Notify ?notify))))
)

(:action check-consistency
        (:parameters ...)
        (:precondition ...)
        (:effect (isConsistent)))

(:action notify-with-sound
:parameters ...)
        (:precondition ...)
        (:effect (... (not (isConsistent))))))
```

4.2 Consistency Check

As explained in the previous section, an ontology consists of a TBox and an ABox. Each world state created after applying an action during planning represents an ABox, and an ABox of an ontology is valid as long as it is consistent according to the rules defined in the TBox. Hence, we need to be sure that none of the steps in a generated plan make the ontology inconsistent; otherwise the generated plan is inapplicable.

In other words, as each state during planning represents an actual, real-world state, none of the actions of a valid plan should put the world in an inconsistent state; e.g. a door cannot be both open and closed at the same time. Action preconditions could be designed to handle such inconsistencies; however, here it is important to focus on the fact that this state cannot be achieved in real life. For this reason, we have to check for consistency of the current state every time the planner applies an action. The rules that may cause inconsistency in an ontology are derived from its TBox. Hence, either an external program must check if the generated plans cause inconsistencies, or the planner must handle this. Most planners do not provide a mechanism to run a program after each step; hence we propose the following solution.

Since we can express the consistency query (generated using the TBox) in PDDL using predicates, we create a special action called `check-consistency` and use a special empty predicate called `isConsistent`. `isConsistent` is true in the initial state and it must also be true in the goal state and in all states that lead to the goal state. Furthermore, all of the action descriptions are modified to include `isConsistent` in their preconditions along with `(not (isConsistent))` in their effects. This simply means that we need the `(isConsistent)` predicate to apply an action, and that the predicate is deleted after an action is applied. Furthermore, the special `check-consistency` action has the negation of the consistency check in its preconditions and `(isConsistent)` in its effects. As `check-consistency` is the only action that can add the `(isConsistent)` predicate, it has to be applied after each action. If the world state is inconsistent, the `check-consistency` action will not add the `(isConsistent)` predicate and all actions will become inapplicable; the goal state will then be unreachable. This will prevent the planner from going even deeper in the current branch of its search space, as that branch will not produce a valid plan.

4.3 Policies as Preferences

The central contribution of this paper is automating the policy conflict resolution process using automated planning techniques. The first step towards this goal is the modeling of the conflicting situation and its attendant information into a planning problem instance. Specifically, obligations and prohibitions relating to a specific entity need to be handled, since they are the primary reason that a conflict might arise.

The key concept here is the framing of obligations and prohibitions as *preferences* on a given policy that must be handled by the underlying planner. In this notion, obligations and prohibitions can be seen first and foremost as goals that an entity must achieve. These goals may be soft in nature, i.e., there may be degrees of fulfillment rather than just binary *true* or *false*. Additionally, since we are considering conflicts in the policy space, obligations and prohibitions may be competing with each other. In such instances, it may be the case that not every conflict can be fully resolved; instead, a plan needs to be formulated that least violates some goodness metric defined for the domain.

We illustrate this idea with the use of our running example, outlined in Section 3.4 previously. Envision a scenario where if the doorbell is pressed, an obligation to notify someone within the house (that there is someone at the door) is immediately created due to an existing policy. However, there is also a current prohibition on making sound, since someone is sleeping – this is due to a second policy that exists on the doorbell. This forces the addressee (in this case, the doorbell) to make a determination and pick between one of the two policies to violate.

However, if there were another way to fulfill both of these policies (one of them with an obligation, the other with a prohibition) at the same time without violating the other, the conflict and ensuing violation could be avoided. Given that we are dealing with complex domains with multiple entities and services, it is entirely possible that notification is possible in a number of ways. For example, instead of making a sound to notify (and fulfill the obligation) that someone is at the door, the doorbell could instead hand off the notification task to another currently active entity. An example of such

an entity could be a television; the television has a service that can visually notify that someone is at the door. This takes care of the obligation on the doorbell's policy, while at the same time not violating the prohibition on the other policy of not making a sound while someone is sleeping.

In our domain, these preferences (one on the obligation, and one on the prohibition) would be represented as follows:

1. `preference pref-0 (gotNotifiedFor Adam Doorbell1)`
2. `preference pref-1 (NotifyWithVisual Doorbell1)`

In the above, `pref-0` stands for the obligation that when the doorbell rings, Adam (a person in the house) must be notified. `pref-1` is the prohibition that whenever the doorbell rings, the notification must happen visually[2]. These two preferences are in conflict, and will be resolved by the planner based on the violation cost that is prescribed for each.

One question that crops up is whether this can just be achieved with regular PDDL actions, without the use of preferences. For example, consider the following actions:

```
(:action notify-with-sound
(:parameters ?person ?device ?event ?notifyWithSound)
(:precondition (and (playSound ?device ?notifyWithSound)
        (NotifyWithSound ?notifyWithSound)(Event ?event)
        (Person ?person) (isConsistent) (Device ?device)))
(:effect (and (gotNotifiedFor ?person ?event)
        (not (isConsistent)) (increase (total-cost) 4))))
```

```
(:action notify-with-visual
(:parameters ?person ?device ?event ?notifyWithVisual)
(:precondition (and (playVisual ?device ?notifyWithVisual)
       (NotifyWithVisual ?notifyWithVisual) (Event ?event)
       (Person ?person) (isConsistent) (Device ?device)))
(:effect (and (gotNotifiedFor ?person ?event)
       (not (isConsistent)) (increase (total-cost) 4))))
```

The two actions above both give the same main effect, namely `(gotNotifiedFor ?person ?event)`. Therefore it bears asking why the conflict resolution problem cannot just be handled in a straightforward manner by the planner without the need to invoke preferences; clearly, there are two choices, and the planner can take the visual notify action if the sound notify will violate some other constraint. However, this line of thought precludes the possibility that sometimes there *may be no other way* to uphold a specific obligation, or avoid a certain prohibition. In certain cases, the problem may be overconstrained to the point where some constraint *has* to be violated. In such cases, it is useful to think of these constraints as no longer hard goals but instead soft constraints that carry violation costs – preferences. The planner now has more room in a complex problem setting to decide which constraints can be violated, and to arrive at the best possible solution.

For instance, in our running example, let us assume that in addition the baby, the baby's parents are sleeping as well, and the doorbell is rung. In this case, the system has to make a decision; to either violate the sound policy, or to not notify and ignore the visitor at the door. In this extended scenario, the planner needs to make a decision according to the violation costs set by the policy's authors. In a real-world, deployed IoT scenario, there may be much more complicated scenarios in which multiple policies are active and the solution is much more complicated; this demonstrates the need for soft constraints of some nature, such as preferences.

Future work in casting policy constraint violations as planning preferences includes modeling the domain such that the planner can pick not only which constraints are violated, but also the *degree* to which those constraints are violated. For example, in a scenario where there is a temperature controller, and two different people have competing preferences (one wants the room to be cold, the other would like it warm), the domain could be modeled in a way that the eventual solution is a compromise between too cold and too warm.

Action Descriptions Given the importance of the constituent actions in our domain description, we briefly describe the genesis of this knowledge. In the context of our application, an action can be an API offered by a device or a web service. For example, an action could be moving a robot, downloading information from the internet, or turning a TV on. For an illustration, see the simplified version of the `notify-with-sound` action shown in the previous section. In order to keep things simple, we assume that service descriptions are available to us in quasi-PDDL form using our ontology and that we do not need to do complicated conversions from a description language. Furthermore, as with many other real-world applications which do service compositions, interleaving planning is essential for IoT applications. However, in this paper we do not focus on these issues.

5 Preliminary Evaluation

At the outset, we clarify that the current evaluation is presented in the spirit of a proof-of-concept rather than as a full-scale evaluation of our design choice to implement policy conflict resolution as a preference-based planning problem. The approach explained in the previous section requires a PDDL planner that supports action costs, derived predicates, and preferences at the same time. Unfortunately, we could not obtain a planner that implements all of those requirements, and were thus unable to evaluate our approach using a single planner. Although our approach is based on PDDL, we initially tested it with the JSHOP2 (Ilghami and

[2] An alternate way of encoding this preference would be to define a `(DoNotUseSound ?entity)` predicate; this is a domain modeling question.

Nau 2003) HTN planner. We then used the LAMA (Richter and Westphal 2010) planner to determine if we could integrate OWL-QL rules into PDDL. Finally, we converted policies into preferences and used SGPlan5 (wei Hsu, Wah, and et al. 2006). As future work, we intend to build a planning system that supports all three requirements: action costs, derived predicates, and preferences.

5.1 JSHOP2

We initially used the JSHOP2 planner since our policy framework was also developed in Java. Additionally, JSHOP2's external calls allowed us to simulate interleaved planning that requires the execution of non-deterministic actions such as locate and search. In an IoT environment, where there are a lot of sensors, interleaving planning with sensing and execution seems to be essential.

However, JSHOP2 does not support domain predicates or preferences. We thus had to develop a component to validate the plans that were generated by JSHOP to check if caused an inconsistent world state or violated/activated new policies. Since JSHOP2 does not support derived predicates, we had to rewrite action preconditions with disjunctions. Unfortunately, disjunctions in the domain file cause JSHOP2 to produce a combinatorial number of duplicate plan files – a large burden on computational resources.

5.2 Evaluation of Derived Predicates

LAMA, which is a planner that builds on Fast Downward (Helmert 2006), supports ADL descriptions, actions costs, and derived predicates. We thus used LAMA to check if we could accommodate OWL-QL reasoning into PDDL.

Throughout our experiments, we used the same ontology that we used with JSHOP2. However, instead of filling up action descriptions with disjunctions in the domain file, we defined re-write rules as derived predicates. Without any additional effort, we successfully managed to find the same plans that we found using JSHOP2 and external programs. Additionally, we tested our approach by slightly modifying the ontology (by making some classes disjoint) to make the planner reach an inconsistent state. As expected, the planner did not find any plans with the modified ontologies. Finally, utilizing derived predicates did not cause LAMA to produce duplicate plan files, which was the case with JSHOP2. Below we reproduce one of the plans produced by LAMA.

```
(locate-people locationdiscoveryinflat flat1 mother room1)
(check-inconsistency)
(notify-people-with-sound mother someoneatfrontdoor
                          speaker playsoundspeaker room1)
(check-inconsistency)
```

Using our example ontology and the derived predicates feature of PDDL, we eliminated the need for externally intervening in the planning process, or re- analyzing generated plans. As part of future work, we will seek to evaluate the performance of using derived predicates versus re-analyzing generated plans using the same PDDL planner.

5.3 Policies as Preferences

The main focus of this paper was to automate policy conflict resolution using AI planners. In order to realize this goal, we first needed to reformulate the conflicting situation as a planning problem. In the previous sections, we discussed how we did this reformulation. In this section, we only discuss how we conducted our preliminary experiments.

As mentioned previously, we could not find a planner that supported all the features we needed in this work. Thus, class type inferences and inconsistent states (according to the underlying ontology) are excluded from these experiments. However, we still use the same initial and goal state files without disjunctions and derived predicates. We chose the winner of the IPC 2006 satisficing track, SGPlan5 (wei Hsu, Wah, and et al. 2006) for our evaluation. SGPlan5 does not support action costs, and actions have to be defined in a STRIPS-like representation. After slightly modifying our domain and problem files to make them SGPlan5 compatible and to include preferences (policies), we were able to produce the same results as we did with the other planners. Below, we reproduce a plan similar to the one shown previously; however, this plan is produced by SGPlan5 taking preferences into account. Thus the `notify-people-with-sound` action is now replaced by the `notify-people-with-visual` action.

```
(locate-people locationdiscoveryinflat flat1 mother room1)
(check-inconsistency)
(notify-people-with-visual mother someoneatfrontdoor tv
                           visual displaymessageontv room1)
(check-inconsistency )
```

Although using preferences makes the formulation of our original problem more natural, it does not completely solve it. Policy activation and expiration conditions are queries. Hence, during planning, we need to query the current state with the activation and expiration conditions of policies to check if a new preference is activated or an active policy is expired.

Generally speaking, the semantic representation of our policies is very suitable for generating a goal state with preferences. Moreover, specifying a goal state instead of specifying an action name is more intuitive in an heterogeneous IoT domain. Although HTN planners can solve the same problems, PDDL makes it easier to automatically generate domain files, as we just need to add action descriptions without defining an action hierarchy. Furthermore, PDDL's support for derived predicates allows us to embed rewrite rules of a QL ontology into the domain file without filling action preconditions with disjunctions. This allows us to find plans by using class inference rules.

In our implementation we were able to integrate QL reasoning into PDDL; however, the planners we found do not offer a way of dealing with interleaved planning, or with the issue of updating preferences (policies) during planning. In our context, these are the shortcomings of existing PDDL-based planners, and we currently still need to analyze generated plans with an external script and re-plan if needed.

6 Related Work

Given the exploratory nature of this work, as well as the use of multiple planners, there is a lot of related work that must be cataloged and explored. Web-PDDL (Dou 2008) adopts and extends PDDL with namespaces and sameAsClass to make ontologies more suitable for web applications. From the same author, another software tool called PDDOWL (Dou et al. 2006) converts OWL-QL queries to Web-PDDL, which are then converted to SQL. Our work does not share the same goal as PDDOWL and Web-PDDL; however (Dou 2008) explains what PDDL lacks to fully represent ontologies.

In addition to PDDL planners, there is an HTN planner called HTNPLAN-P (Sohrabi, Baier, and McIlraith 2009) that supports preferences. HTNPLAN-P extends PDDL3 with HTN-specific constructs; in contrast to the state-centric preferences of PDDL, HTPLAN-P supports preferences that apply to tasks; e.g., "when booking inter-city transportation, I prefer to book a flight".

KAoS (Uszok et al. 2003) was the first effort to offer an ontology based approach for creating a policy framework. Policies are defined as concepts in the ontology using description logic class expressions based on constructs such as subclassOf or object properties like hasCapability, inRoom etc. Thus, it is not possible to use variables in policy descriptions. KAoS can detect conflicts and if a conflict is detected, KAoS checks policies' update times and priority values to resolve conflicts.

OWL-POLAR (Sensoy et al. 2012), which is an OWL-DL based policy framework, can also detect conflicts during design time. Although a conflict resolution strategy is not implemented in that work, authors suggest that an AI planner could be used to avoid conflicts. In addition, it is stated in their paper that doctrines of legal theory and practice could be adopted to resolve policy conflicts. However, OWL-POLAR uses the Pellet reasoner, which is slow and inefficient considering IoT requirements. Furthermore, it may not be possible to completely add the inference and consistency rules of OWL-DL into the PDDL domain.

Rei (Kagal, Finin, and Joshi 2003) is another effort towards an ontology based policy framework. It is based on OWL-Lite and allows policies to be specified as constraints over allowable and obligated actions on resources in the environment (Kagal, Finin, and Joshi 2003). Rei is implemented with Prolog and allows using a logic-like language to describe policies. Prolog provides Rei with the flexibility of specifying relations like role-value maps that are not directly possible in OWL. However, Prolog cannot be used for expressing all DL expressions. Furthermore, Rei cannot detect conflicts between policies at design time; it can detect conflicts when they happen and uses meta-policies to resolve them.

A different method – which is not based on an ontology – proposed in (Vasconcelos, Kollingbaum, and Norman 2009) resolves conflicts by manipulating the constraints associated with the policy's variables, removing any overlap in their values. This approach is similar to adding the conflicting obligation policy as an exception to the prohibition policy. For example, if devices are not allowed to make sound when the baby is sleeping, this approach modifies the prohibition policy to exclude the doorbell.

7 Future Work & Conclusion

In this paper, we discussed how an AI planner could be used in a lightweight policy framework to automate policy conflict resolution. The policy framework is based on OWL-QL mainly because it targets IoT applications that generate large volumes of instance data, and query answering is an essential task in these systems. We managed to encode type inference and consistency check rules of an OWL-QL ontology into a PDDL domain file automatically using a Java program. Furthermore, we converted policies into preferences on PDDL, and used an AI planner to search for a solution which could alleviate the possible conflict.

Our current approach allows us to embed OWL-QL reasoning rules and initial policies (preferences) into PDDL. However, there is much room for improvement. As discussed in previous sections, we could not find a singular planner that could check if new preferences were activated, expired, or violated during planning. We had to develop a small program solely for this purpose. Furthermore, interleaving actions and translating web and device service descriptions into PDDL are not trivial tasks to achieve, but they are necessary for applicability in real-world scenarios.

Another aspect of future work that we hope to focus on is a conflict resolution strategy, which could try to find a middle ground between conflicting policies. For instance, instead of keeping all devices silent when the baby is asleep, devices could be allowed to use sound as long as they don't wake the baby up. Another example (outlined in Section 4.3) could be a conflict between two people who prefer different room temperatures. A solution for this case could be setting to a temperature somewhere in the middle of both preferences. We think that planners would also be helpful in this approach.

Acknowledgements

We wish to thank Biplav Srivastava and Shirin Sohrabi for their help and discussions on planning and preferences. We would also like to thank the anonymous reviewers for their helpful feedback.

References

Artale, A.; Calvanese, D.; Kontchakov, R.; and Zakharyaschev, M. 2009. The dl-lite family and relations. *J. Artif. Int. Res.* 36(1):1–69.

Calvanese, D.; De Giacomo, G.; Lembo, D.; Lenzerini, M.; and Rosati, R. 2007. Tractable reasoning and efficient query answering in description logics: The dl-lite family. *Journal of Automated reasoning* 39(3):385–429.

Dou, D.; LePendu, P.; Kim, S.; and Qi, P. 2006. Integrating databases into the semantic web through an ontology-based framework. In Barga, R. S., and Zhou, X., eds., *ICDE Workshops*, 54. IEEE Computer Society.

Dou, D. 2008. The formal syntax and semantics of web-pddl. Technical report, University of Oregon.

Fikes, R.; Hayes, P.; and Horrocks, I. 2004. Owl-ql?a language for deductive query answering on the semantic web. *Web semantics: Science, services and agents on the World Wide Web* 2(1):19–29.

Gubbi, J.; Buyya, R.; Marusic, S.; and Palaniswami, M. 2013. Internet of things (iot): A vision, architectural elements, and future directions. *Future Generation Computer Systems* 29(7):1645–1660.

Helmert, M. 2006. The fast downward planning system. *J. Artif. Intell. Res.(JAIR)* 26:191–246.

Holler, J.; Tsiatsis, V.; Mulligan, C.; Avesand, S.; Karnouskos, S.; and Boyle, D. 2014. *From Machine-to-machine to the Internet of Things: Introduction to a New Age of Intelligence*. Academic Press.

Iera, A.; Floerkemeier, C.; Mitsugi, J.; and Morabito, G. 2010. The internet of things [guest editorial]. *Wireless Communications, IEEE* 17(6):8–9.

Ilghami, O., and Nau, D. S. 2003. A general approach to synthesize problem-specific planners. Technical report, DTIC Document.

Jayarajah, K.; Yao, S.; Mutharaju, R.; Misra, A.; Mel, G. D.; Skipper, J.; Abdelzaher, T.; and Kolodny, M. 2015. Social signal processing for real-time situational understanding: A vision and approach. In *Mobile Ad Hoc and Sensor Systems (MASS), 2015 IEEE 12th International Conference on*, 627–632. IEEE.

Kagal, L.; Finin, T.; and Joshi, A. 2003. A Policy Language for A Pervasive Computing Environment. In *IEEE 4th International Workshop on Policies for Distributed Systems and Networks*.

Li, X.; Lu, R.; Liang, X.; Shen, X.; Chen, J.; and Lin, X. 2011. Smart community: an internet of things application. *IEEE Communications Magazine* 49(11):68–75.

Lupu, E. C., and Sloman, M. 1999. Conflicts in policy-based distributed systems management. *IEEE Trans. Softw. Eng.* 25(6):852–869.

Richter, S., and Westphal, M. 2010. The lama planner: Guiding cost-based anytime planning with landmarks. *J. Artif. Int. Res.* 39(1):127–177.

Sensoy, M.; Norman, T.; Vasconcelos, W.; and Sycara, K. 2012. Owl-polar: A framework for semantic policy representation and reasoning. *Web Semantics: Science, Services and Agents on the World Wide Web* 12(0).

Sohrabi, S.; Baier, J. A.; and McIlraith, S. A. 2009. Htn planning with preferences. In *Proceedings of the 21st International Jont Conference on Artifical Intelligence*, IJCAI'09, 1790–1797. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Uszok, A.; Bradshaw, J. M.; Jeffers, R.; Suri, N.; Hayes, P.; Breedy, M. R.; Bunch, L.; Johnson, M.; Kulkarni, S.; and Lott, J. 2003. Kaos policy and domain services: Toward a description-logic approach to policy representation, deconfliction, and enforcement. In *Proceedings of Policy*. Como, Italy: AAAI.

van der Meulen, R. 2015. Gartner says 6.4 billion connected "things" will be in use in 2016, up 30 percent from 2015. `http://www.gartner.com/newsroom/id/3165317`. Accessed: 2016-02-29.

Vasconcelos, W. W.; Kollingbaum, M. J.; and Norman, T. J. 2009. Normative conflict resolution in multi-agent systems. *Autonomous Agents and Multi-Agent Systems* 19(2):124–152.

Vermesan, O., and Friess, P. 2011. *Internet of Things-Global Technological and Societal Trends From Smart Environments and Spaces to Green ICT*. River Publishers.

wei Hsu, C.; Wah, B. W.; and et al. 2006. Handling soft constraints and goals preferences in sgplan.

Deploying a Schedule Optimization Tool for Vehicle Testing

Jeremy Ludwig, Annaka Kalton, and **Robert Richards**
Stottler Henke Associates, Inc.
San Mateo, California
{ludwig, kalton, richards} @ stottlerhenke.com

Brian Bautsch Craig Markusic, and **Cyndi Jones**
Honda R&D Americas, Inc.
Raymond, OH
{ CMarkusic, Bbautsch, CJones } @ oh.hra.com

Abstract

Whenever an auto manufacturer refreshes an existing car or truck model or builds a new one, the model will undergo hundreds if not thousands of tests before the factory line and tooling is finished and vehicle production begins. These tests are generally carried out on expensive, custom-made prototype vehicles because the new factory lines for the model do not exist yet. The work presented in this paper describes how an existing intelligent scheduling software framework was modified to include domain-specific heuristics used in the vehicle test planning process. The result of this work is a scheduling tool that optimizes the overall given test schedule in order to complete the work in a given time window while *minimizing* the total number of vehicles required for the test schedule. The tool was validated on the largest testing schedule for an updated vehicle to date. This model exceeded the capabilities of the existing manual scheduling process but was successfully handled by the tool. Additionally the tool was expanded to better integrate it with existing processes and to make it easier for new users to create and optimize testing schedules.

Introduction

Vehicle testing is an essential part of building new cars and trucks. Whether an auto manufacturer refreshes an existing model or builds a new one, the model will undergo hundreds if not thousands of tests. Some tests are exciting, such as a 48 km/h dynamic rollover and measuring the impact on the crash-test dummies. Other tests are not quite as sensational but still important, like testing the heating and air conditioning system.

What these tests have in common is that they are generally carried out on hand-built prototype vehicles because the new factory lines for the models do not exist yet. These vehicles can each cost as much as an ultra-luxury Bentley or Lamborghini, which results in pressure to reduce the number of vehicles. There are two additional complications with the test vehicles. First, the hand-built vehicles take time to build and are not all available at once, but instead become available throughout the testing process based on the *build pitch* of the test vehicles. An example of this is one new test vehicle being made available each weekday. Second, there are many particular types of a model, and each test might require a particular type or any of a set of types (e.g., any all-wheel-drive vehicle). There may be dozens of types of a particular vehicle model to choose from, varying by frame, market, drivetrain, and trim.

At the same time, market forces dictate when new or refreshed models must be released. The result is additional pressure to complete testing by certain dates so model production can begin.

Finally, testing personnel and facilities are limited resources. For example, it would be desirable to schedule all of the crash tests at the very end of the project so other tests could be carried out on those vehicles first. However there aren't enough crash labs or personnel to support this, so the crashes must be staggered throughout the project.

To summarize, the constraints placed on creating a valid schedule in this domain are:

- **Temporal**: Tests must be scheduled between the project start and end date; each test has duration and an optional start date and an optional end date.
- **Calendar**: Tests can only be scheduled during working shifts; tests cannot be scheduled on holidays.
- **Ordering**: Tests can optionally be assigned to follow either immediately after another test or sometime after another test.
- **Resource**: Each test can only be scheduled on certain vehicle types; tests may optionally be required to use the exact same vehicle as another test; tests may require personnel to be available; and tests may require facilities to be available.
- **Build Pitch**: Vehicles are not available for tests until the date they are created; creation dates follow a given build pitch schedule with additional constraints.
- **Exclusive**: Test indicated as exclusive must be the first test on the selected vehicle.

- **Destructive**: Tests indicated as destructive must be the last test on the selected vehicle.

The work presented in this paper describes how Aurora, an existing intelligent scheduling software framework (Kalton, 2006), was modified to include domain-specific algorithms and heuristics used in the vehicle test planning process. The framework combines graph analysis techniques with heuristic scheduling techniques to quickly produce an effective schedule based on a defined set of activities, precedence, and resource requirements. These heuristics are tuned on a domain-specific basis to ensure a high-quality schedule for a given domain. The resulting domain-specific scheduler is named Hotshot.

The end product of this work is a deployed system that automatically creates a valid schedule from a set of constraints provided by the planner. The created test schedule will complete the work in a given time window and observe all of the scheduling constraints. The schedule optimization process includes determining which vehicle types are built and the order in which they are built and *minimizes* the total number of vehicles required for the entire test schedule.

Results from the deployed system are presented from applying the system to a large-scale testing effort for a vehicle model update. This effort was not considered manageable using the existing manual scheduling process, so there is no direct comparison to the pre-existing scheduling process. Prior work reported elsewhere does include a direct comparison between Hotshot and the previous scheduling process with a 12% reduction in number of vehicles required (Ludwig et al., 2014).

In the remainder of this paper, we first discuss related work. Following this we describe the Aurora scheduling framework and summarize changes made to create the domain-specific Hotshot scheduling tool, focusing on the features added to support the transition from prototype to deployed system. The methods and results sections contains the details of how the deployed system was validated by creating one of the largest test schedules for a single vehicle model to date. Finally, we present future work in the conclusion.

The primary contributions of this case study are describing the customization of an existing general scheduling framework to solve a specialized and highly constrained problem and discussing the requirements included in deploying a scheduling system that both supports novice planners and integrates with existing processes.

Background and Related Work

The current version of the software extends the prototype Hotshot system (Ludwig et al., 2014), which demonstrated the ability to generate a valid schedule with a significant reduction in the number of vehicles required relative to the existing planning process. The deployed version of Hotshot includes a number of significant improvements to the initial prototype.

Schwindt & Zimmerman (2015) provide a thorough review of related work aimed at creating test schedules that respect testing constraints and minimize the number of prototype vehicles required. The work presented in this paper is most similar to that of Limtanyakul and Schwiegelshohn (2012, 2007). They use constraint programming to solve nearly the same problem of creating a test schedule for prototype vehicles. Both papers work towards a valid test schedule that meets the same scheduling constraints described previously (temporal, resource, ordering, build pitch, etc.), minimizes how many vehicles are built, determines the vehicle types to build, and determines the order in which the prototypes should be built according to a build pitch.

Bartels and Zimmerman (2007) also worked on the problem of scheduling tests on prototype vehicles meeting temporal, resource, and ordering constraints while minimizing the number of vehicles required. Initially they use a mixed integer linear program model for smaller schedules, moving to a heuristic scheduling method to find solutions for larger schedules. They found that dynamic, multi-pass heuristics produced the best results. These are the same type of prioritization heuristics used in Aurora.

Zakarian (2010) took a different approach in their prototype scheduling work for General Motors. They focused on developing a scheduling and decision support tool that considers the uncertainty in the test process, such as duration of tests, possibility of failure, and prototype availability. The tool helps users trade off between competing goals such as completing the tests according to schedule, quality of testing, and number of prototype vehicles required. Similar to their work, Aurora will highlight conflicted tests that cannot be scheduled because of insufficient resource availability in the given time frame.

One primary difference from previous research is that our work focuses on domain specific customization of a general-purpose scheduling framework already in use in other applications. A scheduling framework takes advantage of the large degree of commonality among the scheduling processes required by different domains, while still accommodating their significant difference. This is accomplished by breaking parts of the scheduling process into discrete components that can easily be replaced and interchanged for new domains.

Framinan and Ruiz (2010) present a design for a general scheduling framework for manufacturing. Aurora, used in our work, is one example of an implemented scheduling framework (Kalton, 2006). Aurora distills the various operations involved in most scheduling problems into

reconfigurable modules that can be exchanged, substituted, adapted, and extended to accommodate new domains (e.g., Richards, 2015; Richards, 2010a; Richards, 2010b). The OZONE Scheduling Framework (Smith et al., 1996) is another example of a system that provides the basis of a scheduling solution through a hierarchical model of components to be extended and evolved by end-developers. Becker (1998) describes the validation of the OZONE concept through its application to a diverse set of real-world problems, such as transportation logistics and resource-constrained project scheduling.

Another difference from existing research is that the scope of the work presented in this paper extends beyond the prior work, including the Hotshot prototype, in a number of ways. The work presented in this paper is part of a deployed system that includes visualization, analysis, and integration with existing processes; is currently in use by novice planners; includes methods to identify and automatically resolve common types of modeling errors created by novice planners; and includes methods to transition the testing schedule from planning stage to execution phase.

Scheduling Framework

Aurora was designed to be a highly flexible and easily customizable scheduling system. It is composed of a number of components that can be plugged in and matched to gain different results. The scheduling system permits arbitrary flexibility by allowing a developer to specify what components to use for different parts of scheduling. Aurora has been successfully applied in a multitude of domains, including medical, manufacturing, and aerospace. (Richards, 2015; Richards, 2010a; Richards, 2010b). The steps in the scheduling process are described in detail below. All configurable elements are shown in bold. Elements that were modified for the test vehicle domain will be discussed further in later sections.

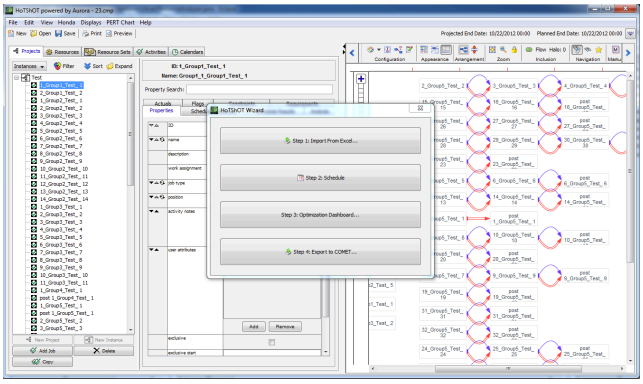


Figure 1. Aurora scheduling user interface.

Additionally, Aurora includes a default user interface that is further customized for each domain. As shown in Figure 1, a domain specific wizard to walk the user through the scheduling process is shown in the center. Behind this are parts of the standard interface: list of tasks (left), information on the selected task (center), and a network diagram (right) showing the ordering constraints between tasks. Other views include resources and calendars as well as various reports and graphs.

Scheduling Process

Schedule Initialization

1. Aurora undoes any previous post-processing (to get back to the "true" schedule result state) and applies the **Preprocessor** to the schedule information.
2. Aurora uses the **Queue Initializer** to set up the queue that will be used to run the scheduling loop. A standard Queue Initializer puts some or all of the schedulable elements—activities, flows, and resources—onto the queue.
3. The queue uses the **Prioritizer** to determine the priority of each element. Depending on the execution strategy, these priorities may be used to periodically sort the queue or to schedule the element with the highest priority at each stage.
4. The Schedule Coordinator triggers the scheduling of the elements on the queue by starting the Scheduling Loop.

Scheduling loop

1. A schedulable element (task, project, or resource) asks the **Scheduler** to schedule it.
2. The Scheduler calls constraint propagation on the schedulable so as to be sure that all of its requirements and restrictions are up to date.
3. The Scheduler looks at the element, considers any **Scheduling Method** that is associated with it (e.g., Forward, Backward). A Scheduling Method determines how the system goes about trying to schedule an element. The Scheduler also selects which **Quality Criterion** to associate with the selected scheduling method; the Quality Criterion determines what makes an assignment "good."
4. The Scheduler calls the Schedule Method on the schedulable. The process depends a great deal on the Schedule Method, but the result is that the schedulable element is assigned to a time window and has resources selected to satisfy any resource requirements. It also returns a list of the conflicts resulting from the given assignment.
5. The Scheduler calls constraint propagation on the schedulable (again) in order to update all of the neighbors so that they are appropriately restricted by the newly scheduled element. This process may result in additional

conflicts; if so, these are added to the list of conflicts from scheduling.
6. The Scheduler adds the conflicts to the **Conflict Manager** and asks the manager to attempt to resolve those conflicts.

Schedule Finalization
1. When the queue is empty, Aurora goes through a final conflict management step, this time at the global level.
2. Aurora calls the **Postprocessor** on the schedule, so that any additional analysis may be done before Aurora returns the schedule results.
3. Aurora sends the schedule results to the GUI for display.

Domain-Specific Customization

Two different types of modifications were made to the Aurora framework to create the Hotshot tool. First, the user interface front end was modified to import the testing model, display and edit domain-specific properties, perform the optimization to minimize the number of required vehicles, and to search for additional resources that could be added to shorten the project schedule. Second, components in the scheduling back end were updated specifically for this domain.

User Interface Customization
There are seven features added to the general scheduling user interface that are specific to the vehicle test domain: import an Excel model of the testing problem, edit the build pitch, edit the vehicles and build order, determine how to handle irregular tasks, minimize the number of vehicles required, search for additional resources that could be used to shorten the schedule, and export the schedule to a client-specific format. Each of these features will be described in greater detail, highlighting new features and changes made to existing features as the user base of the system grew.

The starting point of the Aurora customization for the vehicle testing domain is importing the testing tasks, task constraints, calendars, resources (vehicles, vehicle types, personnel, facilities), and build pitch information from a set of Excel spreadsheets. These Excel spreadsheets represent a model of the overall testing problem. Once imported, the general user interface supports graphically viewing and editing the model elements such as tasks, resource requirements, resources, resource sets, constraints, and calendars. For the deployment, the primary additions were in model error checking. In practice, multiple team leaders specify portions of the overall model. This leads to situations where models are defined that are logically impossible to solve. One example is when there are more days of work indicated than there are days of vehicles available given the build pitch and end date. The deployed system looks for common mistakes made with the prototype and alerts the user to the problem using familiar terms that they can easily understand.

The Build Pitch (Figure 2) dialog was added for viewing and editing the general build pitch per week (number of vehicles that can be built) as well as a maximum for each vehicle type per week. For example, 10 test vehicles per week can be built, but only 5 all-wheel-drive can be built in a week. Additionally, each vehicle type has a first available date, which determines the earliest date that a vehicle of that type could be available. The Manage Vehicles dialog is related to Build Pitch, allowing the user to edit the vehicles to be built and their build order. The build order is assigned automatically during the optimization process. Build dates are assigned based on the build order, moving from 1 to n and selecting the first available date that meets two criteria: number of vehicles/week is not exceeded and vehicle type per week is not exceeded.

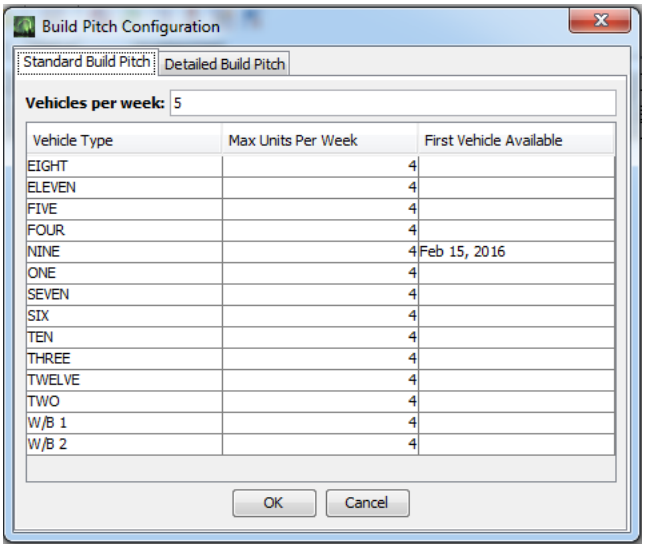


Figure 2. Build pitch configuration.

While the general build pitch can be used at the start of the scheduling process, part-way through the test process a more detailed build pitch is released that gives specific dates in which vehicles will be available, for example, 3 vehicles on Monday the 1st. Later in the test process, the information is even more detailed, with specific dates given to each test vehicle to be built. The deployed version of Hotshot supports the existing planning process by including all three of these modes and updating the schedule as the model transitions from most general to most specific build pitch.

Greater use of the system also indicated that an additional dialog to handle tasks with irregular dates was needed. The Set Task End Dates dialog was added to support tasks that were allowed to extend beyond the

desired project end date. Examples of this include tasks with a given end date later than the project end date, individual tasks that are longer than the entire project duration, and chains of tasks linked with ordering constraints such that the combined duration exceeds the entire project duration. In all these cases, the user determines if the entire project should be extended or if the task(s) will be allowed to complete at a date past the project end date.

The Optimization Dashboard (Figure 3) is used to minimize the number of vehicles required to schedule the testing tasks. In Hotshot, optimization is accomplished by using the scheduling engine to perform a search through the space of schedules to find a valid schedule that requires fewer vehicles. The Optimization Dashboard helps visualize the setup and search process for the user. The upper left area of Figure 3 summarizes the present state of the current schedule, showing the number of vehicles required, the number of destructive and exclusive tasks, the utilization of vehicles in the testing schedule, and the actual project end date relative to the initial. The upper right shows the current status of optimization, which will change once the Start button is pressed. This portion of the dialog also provides an estimate of how long the remaining optimization will take. The central portion of the dialog contains the five parts of the optimization process. Buttons for starting and controlling the optimization are found along the bottom of the dialog.

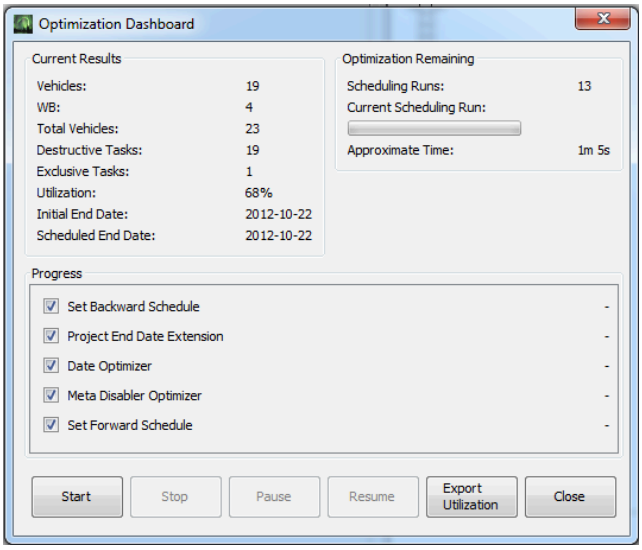


Figure 3. Optimization dashboard.

There are five steps in the optimization process. The first three steps prepare the scheduling model for search, the fourth step carries out the search, and the final step returns the schedule to the end user format:

1. **Set Backward Schedule.** Mark all tasks to be backward scheduled. This means that the schedule will be created from the end of the project to the beginning, with all tasks scheduled as close to their late end dates as possible.
2. **Project End Date Extension.** Users of the prototype often ran into problems when the desired model was too ambitious for a solution to be found. This optimization step was added to correct for overly optimistic end dates. The end date extension attempts to schedule the project with all of the constraints except build pitch. If conflicts are found, it extends the project end date to try to fix the issues for the user. If conflicts still exist, optimization is aborted. At this point the planner will need to refer to the conflicts to fix the model issues.
3. **Date Optimizer.** Once the tasks are backward scheduled, assign build order based on the earliest dates tasks are assigned to vehicles. That is, if the first task assigned to Vehicle A starts on Jan 15 and the first task assigned to Vehicle B starts on Jan 18, then A will come before B in the build order. The heuristic is that vehicles that are needed earlier should be built earlier. Note that this optimizer greatly reduces the amount of time available to test vehicles built later in the schedule. Due to the same issue with ambitious test models as seen in step 2., this optimizer will also attempt to extend the project end date if conflicts are found once build dates are applied.
4. **Meta Disabler Optimizer.** This step uses the scheduling engine as part of a greedy search for valid schedules that require fewer vehicles. Starting with the vehicles created last in the schedule, temporarily disable the vehicle and use the scheduling engine to try to create a schedule without the vehicle. If this succeeds, permanently delete the vehicle. If this fails, restore the vehicle and continue. As vehicles are removed, the Date Optimizer is used to re-order the build dates of the remaining vehicles.
5. **Set Forward Schedule.** Finally each task is returned to forward schedule mode and re-scheduled so that all tasks try to schedule as close to the project start date as possible. This is the preferred output format for downstream processes that make use of the schedule created by Hotshot.

The Resource Analysis dialog was developed to provide guidance to new users on how they could improve the schedule, either by reducing the number of vehicles required or by shortening the project duration. Starting from an optimized schedule, the Resource Analysis dialog carries out a meta-search process. For each resource (vehicle type, personnel, or facility), the dialog will perform the optimization process as if another of that resource were available. The user is shown the effect of adding each resource individually to the schedule in terms of project end date and number of cars required. This serves as a starting point for discussion on what-if scenarios for improving the schedule.

The final feature is aimed at making the scheduling results easier to use as part of the larger process of

planning for and carrying out tests. Aurora contains a variety of highly customizable displays such as the plot shown in Figure 4. In this figure the actual vehicle and tasks have been simplified and obfuscated. Vehicles are shown on the x-axis and time on the y-axis. For example, SIX~001 is the first vehicle instance of type SIX. It has the task 9_Group2_Test_9 assigned to it from August 9 to October 10. The light-purple cell on August 8 to the left of the task indicates the vehicle is not yet available for scheduling, visualizing the build pitch. The plot has also been customized to color code tasks by group and to indicate destructive tasks with a yellow tag in the upper right corner. However, an existing format of scheduling results is already in use as part of the vehicle testing process. The deployed version of Hotshot includes custom export capabilities to inject the Hotshot results into the existing, proprietary system.

Scheduling Component Customization

The main change for the deployed scheduling system was to support irregular tasks that are allowed to complete outside of the project start and end dates.

The scheduling model is based on a hierarchical structure with flows and tasks. Flows represent high-level projects made up of individual tasks that must be scheduled. One of the key constraints on flows and tasks is the *late end date*. Late end date represents that last possible date the project or task can be completed. Under normal conditions, the late end date of the flow will constrain the late end date of any of the tasks that make up the flow. For example, Task A has a defined late end date of 12/31/2015. Task A is part of Flow "Test Project," which has a late end date of 12/01/2015. The **Preprocessor** synchronizes the end dates of the tasks and the flow that contains the tasks. So in the above example, Task A will be updated to have a late end date of 12/01/2015 because it is constrained by the flow end date. In real-world conditions, the flow late end date describes when the bulk of the tasks need to be finished by, but there are some tasks that can be safely finished after the project is considered complete. To support this, tasks can be marked as *override project end date*. The updated **Preprocessor** does not change the late end dates of these tasks.

The Hotshot prototype component customization focused on three central areas: scheduling direction maintenance, special handling for vehicle testing's unusual requirements, and more standard heuristic tailoring for the domain. In this paper we focus on the **Prioritizer** component, which was not described in detail previously. See Ludwig et al. (2014) for the customization of other scheduling components: **Preprocessor**, **Scheduler quality criteria**, **Scheduler**, and **Post-processor**.

The **Prioritizer** uses a cascading series of heuristics to determine which activities should be scheduled earlier in the process. In general, if "difficult" activities in the given domain are scheduled earlier in the process, it tends to avoid subsequent conflicts in the schedule that would be difficult to repair. The heuristic prioritizer considers each heuristic in turn, until it can differentiate between two prospective activities. If two activities tie on a given heuristic (e.g. both are "exclusive use" activities), the prioritizer will consider the next heuristic ("long task"), and the next, until it can break the tie. The primary heuristics in this domain are:

- **Exclusive task**: Prefer to schedule activities that must have exclusive use of a vehicle earlier in the process.
- **Long task**: Schedule the long activities early in the process and fill in with short activities.

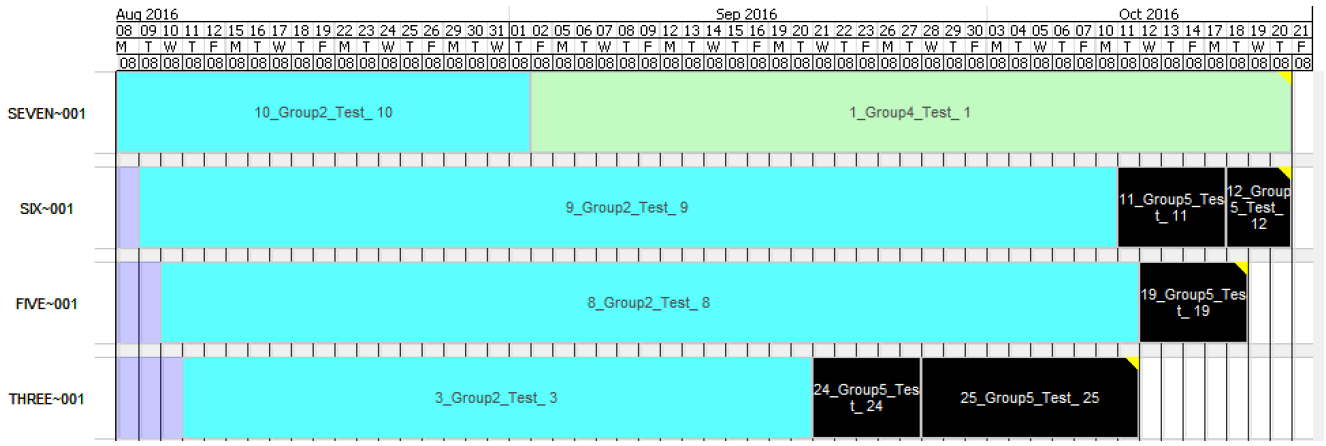


Figure 4. Aurora assignment of tasks to vehicles over time. For example, SIX~001 is the first instance of a vehicle of type SIX. It is available for tasks starting August 9th. The first task assigned to this vehicle is 9_Group2_Test_9.

- **Destructive task**: Some tests involve destroying the vehicle. This prevents any activities from subsequently making use of the vehicle, so it is important to place the destructive tasks - as late in the schedule as possible - early in the scheduling process.
- **Tight window**: This reflects the fact that activities with a short window of opportunity tend to be harder to place than those with a long window of opportunity. In this case, the "tightness" reflects the difference between task duration and the projected window size.
- **End based**: Schedule tasks that must be completed first earlier in the process for the forward-schedule phase.
- **Load-based**: Prefer activities with fewer vehicle options and/or more competition for those vehicles.
- **Subsequent duration**: Considers the amount of follow-on work after the current activity, based on ordering constraints.

Methods

Based on the prototype results, Hotshot was immediately put to work on a large project that had several challenging constraints from the start. The first challenge was that the factory in charge of this project had capacity issues and was not able to build enough vehicles to satisfy all of the testing and development requirements. As a result, a second factory was sourced to make up the shortage of vehicles the first factory was unable to produce. Developing an optimized test schedule manually for this type of build—one that included two build locations, two separate build pitches, and two different timelines—had not been attempted previously.

To tackle this challenge, the project leader followed a divide and conquer approach. Instead of treating this project as one very large schedule, it was divided into two medium-sized schedules that were individually optimized. To carry this out, the project leader separated the exclusive tasks from the non-exclusive tasks. All exclusive tasks were to be scheduled and optimized at the second factory while the non-exclusive tasks were to be scheduled and optimized at the first factory.

The second challenge centered on negotiations to create schedules that worked for both the testing team and the factories building the test vehicles. The schedule that was using the first factory, which was to build vehicles for the non-exclusive tasks, was created first. Ongoing negotiations took place with this factory with regard to build timing and build pitch. This factory had several assignments it was balancing and had to make changes and requests in real time during the schedule optimization process. These requests were fed back to the project team, which utilized Hotshot to update the test schedule. Most of the change requests involved when to build certain vehicles and how many vehicles to build per week. The project team was able to honor the factory's requests as well as counter-propose options that would help further optimize vehicles and schedules.

After the non-exclusive tasks' schedule was created, the exclusive schedule was started. The second factory was an in-house fabrication department that had different requirements and constraints than those of the first factory. However, the same process of creating a test schedule was used in this case. Build pitch and build timing were considered when performing the optimizations and schedule creation.

Results

In the end, the team was able to create an optimized schedule with a larger number of vehicles (50 – 150) that met all of the non-exclusive test groups' needs. The second exclusive task schedule was created using a smaller number (10 – 50) of test vehicles. In total, a large number of vehicles were needed to satisfy the project requirements and testing needs, drawing from 30 – 50 different vehicle types. This included 4042 days of testing, with over 340 testing tasks. These tasks were constrained by the completion of preceding tasks, by requiring the use of the same resources as preceding tasks, and by the availability of vehicles, personnel, and testing facilities. In addition to successfully scheduling a suite of tests that would have been very difficult previously, Hotshot also supported the negotiation process and minimized the number of test vehicles required.

Working with the initial prototype, planners demonstrated the ability of generating a schedule in under two minutes, as opposed to this task requiring days of labor. This capability enabled the planners to generate numerous "what-if" scenarios. Planners could quantify the effect of compressing or extending the schedule in terms of how many cars would be required. Planners also demonstrated the effects that steeper and shallower build pitches have on the number of cars required for a given set of tasks and project end dates. Planners were also able to negotiate about the vehicle types required by tests. For example, a vehicle type requested by one test but not usable by other tests stands out in plot as a vehicle with very low utilization. The planner can then go back to the person in charge of the test and see if a more commonly used vehicle type could be substituted. The ability to quickly run "what-if" scenarios held true with the larger models as well. The ability to quickly examine these types

of effects enabled a more efficient negotiation process to take place between the test and vehicle production teams than during previous challenging projects.

Hotshot was also used to minimize the number of test vehicles required for this large project. Unfortunately, there is no direct comparison to the previous method because no manual model was attempted given the complexity of the test schedule. The only estimate we can give for the number of vehicles saved is based on the combined judgment from several members of the project team. They estimated that Hotshot created at least six fewer vehicles than would have been created with the previous method. This represents a 6% reduction in the number of vehicles required and a significant cost savings in the millions of dollars.

Note that this estimate was purposely conservative. The prototype version (Ludwig et al., 2014) demonstrated a 12% reduction in vehicles when directly compared with the manually created schedule on a much smaller model.

Conclusion

This paper described a complex, real-world scheduling problem in automotive vehicle testing prototype management. To address this problem, we added domain-specific heuristics to a general intelligent scheduling software framework to create the custom Hotshot scheduling software.

Hotshot helped solve a very complex scheduling challenge in the presented use case. Solving this challenge with the previous, manual method would have been almost impossible. As deployed, Hotshot enabled the schedules to be created in an efficient manner while also building fewer vehicles than the manual method would have needed.

Due to the reduction in required vehicles, this use case also demonstrates the cost-effective development of a customized scheduling system. The savings from the reduced vehicles alone in the presented use case greatly outweighs development cost, and additional savings are generated with each new project. Hotshot has already saved the end user millions of dollars in prototype costs while increasing transparency of the entire process from the implementation level to the executive level.

Ongoing work is aimed at scaling Hotshot, and its optimization capabilities, to multiple simultaneous projects. Currently, Hotshot is used to optimize a single project. The functionality around build pitch allows constraints caused by vehicle availability for multiple projects to be factored into the schedule. However, the schedule does not take into account delays that could be introduced due to conflicts in using limited personnel and testing facilities for different projects being run at the same time. The next step in development will assist planners in creating a combined schedule for all of the active testing projects at any given time.

References

Bartels, J.-H., & Zimmermann, J. (2007). Scheduling tests in automotive R&D projects. European Journal of Operational Research, 193(3), 805–819.

Becker, M.A. (1998). Reconfigurable Architectures for Mixed-Initiative Planning and Scheduling. Ph.D. diss., Robotics Institute and Graduate School of Industrial Administration, Carnegie Mellon university, Pittsburgh, PA.

Framiñan, J.M., & Ruiz, R. (2010). Architecture of manufacturing scheduling systems: Literature review and an integrated proposal. *European Journal of Operational Research* 205(2): 237-246.

Kalton, A. (2006). Applying an Intelligent Reconfigurable Scheduling System to Large-Scale Production Scheduling. *International Conference on Automated Planning & Scheduling (ICAPS) 2006*. Ambleside, The English Lake District, U.K. June 6-10, 2006.

Limtanyakul, K., & Schwiegelshohn, U. (2012). Improvements of constraint programming and hybrid methods for scheduling of tests on vehicle prototypes. *Constraints*, 17, 172-203.

Limtanyakul, K., & Schwiegelshohn, U. (2007). Scheduling tests on vehicle prototypes using constraint programming. In *Proceedings of the 3rd multidisciplinary international scheduling conference: Theory and applications* (pp. 336–343).

Ludwig, J., A. Kalton, R. Richards, B. Bautsch, C. Markusic, J. Schumacher (2014). A Schedule Optimization Tool for Destructive and Non-Destructive Vehicle Tests. *Proceedings of the Twenty-Sixth Annual Conference on Innovative Applications of Artificial Intelligence (IAAI 2014)*

Richards, R. (2010a). Critical Chain: Short-Duration Tasks & Intelligent Scheduling in e.g., Medical, Manufacturing & Maintenance. *Proceedings of the 2010 Continuous Process Improvement (CPI) Symposium.* Cal State University, Channel Islands. August 19-20, 2010.

Richards, R. (2010b). *Enhancing Resource-Leveling via Intelligent Scheduling: Turnaround & Aerospace Applications Demonstrating 25%+ Flow-Time Reduction. 2010 PMI College of Scheduling Conference PMICOS. Calgary, Canada. May 2-5, 2010.*

Richards, R. (2015). Packaging Line Scheduling Optimization. *Pharmaceutical Manufacturing Vol 14 no 8 pp 13-15, Oct 2015.*

Schwindt, C. & Zimmermann, J. (Eds.). (2015). *Handbook on Project Management and Scheduling Vol. 2*. Springer International Publishing.

Smith, S.F., Lassila, O. and Becker, M. (1996). Configurable, Mixed-Initiative Systems for Planning and Scheduling. In: Tate, A. (Ed.). *Advanced Planning Technology*. Menlo Park, CA: AAAI Press.

Zakarian, A. (2010). A methodology for the performance analysis of product validation and test plans. International Journal of Product Development, 10(4), 369–392.

Exploring Organic Synthesis with State-of-the-Art Planning Techniques

Rami Matloob and **Mikhail Soutchanski**
Dept. of Comp. Science,
Ryerson University, 245 Church Street, ENG281, Toronto, ON, M5B 2K3, Canada

Abstract

We explore different techniques to solve the computationally challenging, but practically important organic synthesis problem. This problem requires finding a sequence of reactions producing the target molecule from a set of given initial molecules. This problem is often used on exams to test the problem solving skills of students who study generic reactions in organic chemistry courses. This problem is also relevant in industry. In a quest to find a more efficient way to solve a set of benchmark problems, we start by explaining how the organic synthesis problem can be formulated as a planning problem in PDDL. We then demonstrate how state-of-the-art planners such as SASE, Madagascar and SatPlan – that reduce a bounded planning problem to satisfiability – can only encode a fraction of the benchmark problems. We also explore the recently developed action schema splitting syntactic transformation that splits each action into several sub-actions with a shorter interface thereby alleviating somewhat the grounding problem. We assess experimentally the performance of the Fast Downward planning system for different splitting values and compare the results with our original un-split domain. For the unique finest split, where the modified action schemas have the smallest number of arguments, we investigate performance of Fast Downward with different heuristics. We propose organic synthesis as the new challenge for planning.

1 Introduction

We are involved into a long-term project related to developing an online automated tutoring system for undergraduate students who study Organic Chemistry. The goal is to create a system that will provide many functionalities. In particular, the main feature will be helping the chemistry students with solving the multi-step organic synthesis problem. The organic synthesis problem is a common chemistry problem that requires finding a sequence of reactions that produces a target molecule from given initial molecules. The students who take Organic Chemistry courses are introduced to many generic reactions. These reactions are generic in the sense that they are applicable to multiple classes of molecules. The students study details of each reaction including how atoms in participating molecules change bonds. To test the students knowledge it is common to give them a few organic synthesis problems so the students can discover the right combination of reactions for a given target molecule and initial molecules. These exam problems can vary in difficulty from relatively simple problems that can be solved in 2 or 3 steps to more complex problems that may require finding a sequence of 10-12 reactions. The students can improve their knowledge about reactions if they can practice solving numerous organic synthesis problems before the exam. We would like to develop a tool that will be verifying whether a solution proposed by a student is correct, and if not, after a number of unsuccessful attempts, show a correct solution upon request from the student. More specifically, we would like to develop a tool that can be challenged with any organic synthesis problem. The problem can be either entered by a human using existing open source molecule editing tools, or it can be generated automatically. The range of organic synthesis problems is potentially unlimited, i.e., the tool should be able to solve any problem of certain level of complexity, but not just problems out of a finite library entered in advance. It is possible that this tool will be also helpful for research purposes since the organic synthesis problem is a long-standing problem with industrial applications. There has been a long history of research related to *computer aided synthesis design* (CASD). This history and the current developments are well outlined in (Judson 2009; Cook et al. 2012; Ravitz 2013; Bøgevig et al. 2015; Szymkuć et al. 2016). Reviewing this history would not be relevant for the purposes of our paper, but it is sufficient to mention that most of the systems required interaction with a human assistant, and to the best of our knowledge they are either no longer developed, or they are not publicly available for assessment.

The recent work (Heifets and Jurisica 2012) explored solving the organic synthesis problem using AND/OR graph search with a modified proof number search (this approach is popular when solving large two-player zero-sum games such as Go or Chess). The significant contributions of their work include development of 20 benchmark problems inspired by the exam questions given to the students at the Massachusetts Institute of Technology (MIT) taking Course 5.13 Organic Chemistry II during 2001-2006. Since these are real exam questions, it is interesting to see if they can be solved by a computer program. About 50 reactions required to solve all these 20 problems were manually encoded using a specialized chem-informatics language (James, Weininger, and Delany 2011). This benchmark set is publicly available from (Heifets 2012). The search for a sequence of reactions was facilitated by a proprietary chem-informatics software JChem developed by *ChemAxon*; it reads encodings of reactions and molecules and can answer queries whether molecules match (ChemAxon 2015). Since this search requires significant computational resources, it was implemented on an IBM super-computer. Each test received up to 6 hours of CPU time and 8GB of RAM to solve one of the benchmark problems. The experimental results are reported in (Heifets and Jurisica 2012). They demonstrate that some of the 20 benchmark problems could be solved within 6 hours limit, but the problems 16, 17, 18, 19 and 20 could not be solved.

This previous research begs naturally the question whether the organic synthesis problems can be solved using any of the modern AI planning techniques while relying only on standard hardware. To explore this, (Masoumi, Antoniazzi, and Soutchanski 2015) has developed representation of reactions in the Planning Domain Definition Language (PDDL). PDDL has been designed to standardize Artificial Intelligence (AI) planning languages. It is extensively used by all systems competing at the International Planning Competitions. More specifically, (Masoumi, Antoniazzi, and Soutchanski 2015) has developed PDDL representation of generic reactions in PDDL 2.2 that includes derived predicates (Edelkamp and Hoffmann 2004). Before any of the planning problems could be solved, both initial and goal states should be encoded in PDDL. This has been done manually. Additionally, the reactions from the benchmark (Heifets 2012) should be somehow translated from the specialized chem-informatics language into PDDL. Automatic translation was impossible because some effects of the reactions in (Heifets 2012) are left unspecified. This was not an obstacle for research reported in (Heifets and Jurisica 2012) since their program searched from target to input molecules. However, in PDDL encoding, all the effects must be stated explicitly. For this reason, all the required reactions have been manually entered and saved using *MarvinSketch* reaction editing software provided by *ChemAxon* (ChemAxon 2015). This software allows saving the reactions in a RXN format that is popular in chem-informatics industry (Accelrys 2011). Subsequently, the reactions have been automatically translated from RXN into PDDL using an in-house developed computer program. Experimental results reported in (Masoumi, Antoniazzi, and Soutchanski 2015) are mostly negative because the evaluated AI planners suffered from the grounding problem. Each planning instance involves a few dozen atoms, such as carbon, hydrogen or oxygen, while some of the actions have more than 5 arguments, and as a consequence, when actions are instantiated with available atoms the size of a transition system constructed in the process of grounding exceeds the available memory. As reported in (Masoumi, Antoniazzi, and Soutchanski 2015), the computer memory with 128GB of RAM was not enough even if only 23 actions remained in the PDDL domain.

Several promising research directions have been identified in (Masoumi, Antoniazzi, and Soutchanski 2015). Our paper reports experimental results collected while pursuing some of the identified research directions. This is the first contribution of our paper. A close inspection of the previously developed PDDL domain with reactions has revealed that some of the reactions and/or planning instances with initial and goal states had inaccuracies, e.g., types mismatches. Some of the errors occurred because the reactions have been entered into *MarvinSketch* manually by a Computer Science undergraduate student. For this reason, before we could proceed, we have carefully debugged a set of reactions. The resulting clean set of 24 reactions is the PDDL domain that we use in our research: see Table 1. This domain can be used to solve some of the benchmark problems: see Table 2. For each of these problems, we have verified that it can actually be solved using the reactions from a smaller sub-domain that includes only required reactions. Thanks to this verification, we are confident that all problems we are investigating in this paper actually have a solution that can be potentially computed. Elucidating this clean PDDL sub-domain is our second contribution.[1]

Reaction Name	#VARs	PRE	ADD	DEL
alcoholAndPBr3	7	8	4	4
aldolCondensation	15	22	10	10
alkeneAndWater	5	5	6	4
alkylHalideAndCyanide	5	4	4	4
anhydrideReduction	17	46	16	20
aromaticNitration	15	25	6	6
carboxylicAcidAndThionylChloride	9	10	6	8
catalyticHydrogenationOfNitroGroup	6	6	6	6
dediazoniation	6	6	2	4
diazotization	10	13	4	10
dielsAlder	6	19	12	8
enolSN2attackOnAlkylHalide	6	8	8	8
grignard	6	4	6	4
grignardAdditionToAcidChloride	7	7	4	4
grignardReagentFormation	3	1	4	2
imineFormation	8	9	6	6
imineReductionToAmine	14	22	8	6
ketoneAndLDA	13	29	8	8
ketoneReduction	10	17	6	6
michaelAdditionWithKetones	9	15	6	4
michaelAddition	7	14	6	4
oxidationOfAlcoholsWithPCC	16	30	8	10
nitrileReductionToAmine	9	14	10	12
sandmeyerReaction	6	5	4	6

Table 1: 24 verified reactions and their parameters.

Pr	TotalObj	Hydrogen	Carbon	Oxygen	PlanLength
2	40	16	8	11	4
4	68	24	9	21	8
5	62	28	20	8	3
6	32	10	4	5	7
10	59	31	14	7	3
14	74	40	17	5	5
17	94	46	27	12	5
20	58	22	9	6	11

Table 2: Number of objects in each problem and plan length

The rest of our paper is structured as follows. First, we review relevant work from planning and provide introduction to organic chemistry to facilitate understanding of our paper. Second, we report results from our experimental evaluation. They include attempts to solve the benchmark problems using programs that reduce planning to satisfiability. Subsequently, we have attempted to transform the domain syntactically using splitting approach developed in (Areces et al. 2014). We report results collected with Fast Downward (FD) software (Helmert and et al 2015) that we use to solve benchmark problems with the domains produced by splitting software. As a conclusion, we discuss some of the research directions that can be further explored. It remains to be seen whether modern AI planning techniques are mature enough to compete with undergraduate students when solving exam problems from Organic Chemistry.

[1]We are going to share publicly the PDDL encoding of all reactions and all 20 benchmark problems once we have completed debugging and verified that there are no errors left.

2 Background

The following papers investigate how reducing a bounded planning problem to satisfiability (SAT) can be an efficient approach to solving planning problems. In 1992, (Kautz and Selman 1992) proposed reduction of planning to satisfiability and explained how effects and preconditions can be encoded into SAT. This paper also explains how restrictions can be added in SAT so encodings can be more compact.

The details and the advantages of linear encodings (with operator splitting and explanatory frame axioms) and parallelized encodings are explained in (Kautz, McAllester, and Selman 1996). This is also the first paper that introduces lifted causal encodings.

The idea of reducing classical planning problems efficiently to SAT is further investigated in (Ernst, Millstein, and Weld 1997). This is the first paper that discusses the features of a fully-implemented compiler that can generate SAT encodings for STRIPS planning problems.

SatPlan's technical guide (Kautz, Selman, and Hoffmann 2006) discusses the different versions of the SatPlan system and how it has improved over the years.

A new encoding scheme based on the SAS+ formalism is introduced in (Huang, Chen, and Zhang 2010). Their encoding reduces the number of clauses in the problem by exploiting the structural information in the SAS+ formalism, where each variable can vary over a finite domain, while PDDL has only boolean variables. This paper also introduces a new SAT-based planner named SASE.

Madagascar is a modern SAT-based planner that was implemented to improve scalability of large SAT problems (Rintanen 2014; 2015).

Action schema splitting automated domain transformation approach is introduced in (Areces et al. 2014). It transforms an action schema with a big interface (many parameters) into several properly coordinated sub-actions with smaller interfaces. This automated transformation helps reducing the number of ground actions, but makes the length of a plan longer. The grain of split (course split vs fine split) can be controlled in an implementation using a numerical parameter γ. The smaller γ is, the fewer variables the generated sub-actions will have, and the more sub-actions will be produced for each given action schema. We use an implementation of this approach downloaded from the author's Web site on September 19, 2014.

3 Preliminaries

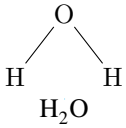
In this Section, we explain how a chemical molecules and reactions can be encoded in PDDL. Consider molecules as graphs with edges that have four different labels representing four common types of bonds between atoms. We model them using the predicate $bond(?x, ?y)$ – it represents a single bond between atoms $?x$ and $?y$ – and the predicates $doublebond(?x, ?y)$, $triplebond(?x, ?y)$ and $aromaticbond(?x, ?y)$, where the latter bond is for the case when several atoms share electrons as in the benzene ring molecule. Furthermore, in graphs representing molecules, vertices are labeled by the name of a chemical atom from the periodic table, such as carbon (C), oxygen (O), hydrogen (H), nitrogen (N), sodium (Na), chromium (Cr), chlorine (Cl), and so on. Since molecules can be described as graphs, it is convenient to consider reactions as graph transformations that break some of the existing bonds and form some new bonds between atoms in participating molecules. Each chemical atom has valence, which can be determined by the number of connections it can form. For example, hydrogen has a valence of 1, oxygen has a valence of 2, and therefore oxygen can form one double bond, or two single bonds. Each carbon atom has a valence of 4, and for this reason, it can form all four kinds of bonds as explained above, e.g., one triple bond and one single bond, or one double bond and two single bonds, and so on. When drawing graphs representing molecules with aromatic bonds, it is common to draw a circle to portray these bonds for atoms around the circle, e.g., see the pyridine molecule C_5H_5N that occurs before and after reaction arrow in Figure 1.

To explain our encoding of reactions in PDDL, we use problem 5 as an example. One of molecules participating in reactions solving problem 5 is water or H_2O. We define the atoms in H_2O as follows.

```
o51 - oxygen
h51 - hydrogen
h52 - hydrogen
```

where we write first names of the constants and then their types. To describe bonds between the atoms in the molecule H_2O we have to write each predicate twice to characterize bonds in both directions (for clarity, we do not show numerical indexes in the illustrations).

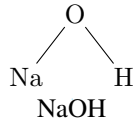
(bond h51 o51) ;
(bond o51 h51) ;
(bond h52 o51) ;
(bond o51 h52) ;



Another participating molecule sodium hydroxide or NaOH can be represented as follows.

```
na - sodium
o50 - oxygen
h50 - hydrogen
```

(bond na o50) ;
(bond o50 na) ;
(bond h50 o50) ;
(bond o50 h50) ;



The problem 5, abbreviated subsequently as p5, can be solved using combination or two reactions; first, $oxidationOfAlcoholsWithPCC$ is executed twice with different molecules, and then $aldolCondensation$ is executed last to produce the target molecule. Figures 1 and 2 with molecules participating in the reactions are shown below. In RXN files all atoms are consecutively numbered. In the figures, the numbers are adjacent to each atom. Thanks to these numbers, it is possible to identify the changes in bonds between atoms during the reactions. The complete PDDL code for $aldolCondensation$ can be found in Appendix, but below we show and discuss a snippet that includes a few preconditions and selected effects, for brevity.

We encode generic reactions as action schemas because reactions apply to large classes of molecules not just to specific molecules. For example, many molecules in organic chemistry use alkyls. The simplest alkyls are methyl $-CH_3$ and ethyl $-CH_2-CH_3$. Each alkyl has a *key carbon atom*

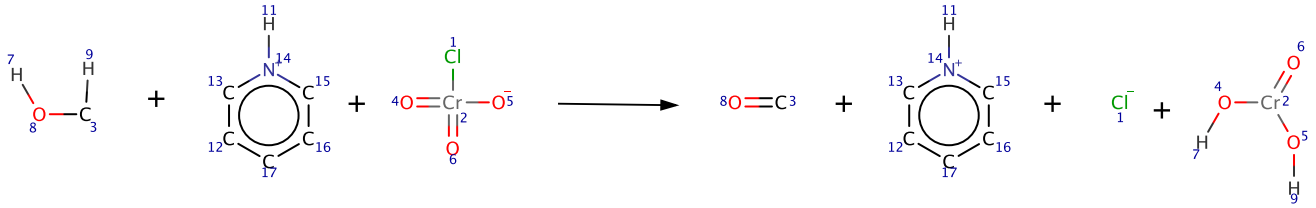


Figure 1: Oxidation of alcohols with PCC reaction

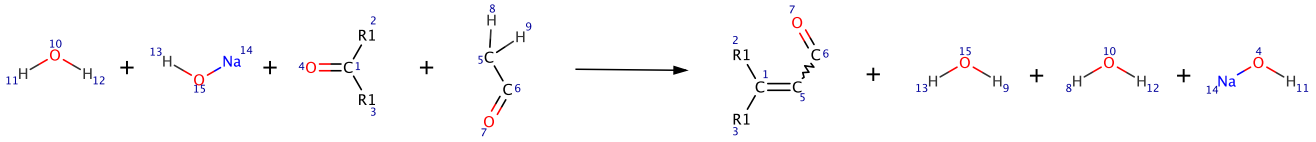


Figure 2: Aldol condensation reaction

which bonds with an external atom to form a molecule. An alkyl is a molecule with the general formula C_nH_{2n+1}, where n is a positive integer. The alkyls are arbitrary branching trees of single bonded carbon atoms such that the remaining carbon valences are filled with hydrogens. In Figure 2, alkyls are denoted with $R1$, but each $R1$-instance can be a different alkyl. Therefore, $aldolCondensation$ is a schema covering a class of individual reactions. In our program, alkyls can be represented as r-group atoms or can be simply represented by a single key carbon atom in the PDDL code. In (Masoumi, Antoniazzi, and Soutchanski 2015), it was attempted to use PDDL derived predicates to represent alkyls and other functional groups common in organic chemistry, but this did not work since grounding of the domain with derived predicates could not fit in memory.

In $aldolCondensation$, the most important change involves two bonds. On the left hand side, the double bond between the carbon atom with number 1 and the oxygen atom with the number 4 cleaves, and subsequently, the 4th atom forms new bonds with potassium Na (number 14) and hydrogen (number 11). On the right hand side, the new double bond between the 1st and 5th carbon atoms forms, while the previous bonds of the 5th carbon with 8th and 9th hydrogen atoms cleave. In the PDDL code snippet we reflect only the main changes, while skipping minor details. The indexes of PDDL variables correspond to the numbers assigned to atoms in the figure. We can describe each bond once when we write preconditions because bonds are symmetrical in the input molecules, but to specify effects we write each bond relation in both directions to maintain its symmetry.

```
(:action aldolCondensation
    :parameters  /* skip for brevity */
    :precondition (and (not (= ?c_1 ?c_5))
(not (= ?o_10 ?o_4))  (not (= ?o_10 ?o_15))
(not (= ?o_4 ?o_15)) (not (= ?r1_2 ?r1_3))
 (bond ?r1_2 ?c_1) (bond ?r1_3 ?c_1)
(doublebond ?o_4 ?c_1) /* skip a few */
(not (= ?h_9 ?h_8)) (not (= ?c_6 ?c_5))
(bond ?c_6 ?c_5) (bond ?h_9 ?c_5)
(bond ?h_8 ?c_5) (doublebond ?o_7 ?c_6))
    :effect (and /* skip some effects */
(not(doublebond ?o_4 ?c_1))(not(doublebond ?c_1 ?o_4))
 (not (bond ?h_8 ?c_5)) (not (bond ?c_5 ?h_8))
 (not (bond ?h_9 ?c_5)) (not (bond ?c_5 ?h_9))
            )
)
```

In the $oxidationOfAlcoholsWithPCC$ reaction, the large middle pyridinium molecule C_5H_5NH is part of pyridinium chlorochromate (PCC) reagent $C_5H_5NH[CrO_3Cl]$. Its purpose is transforming the left-most alcohol molecule into the carbonyl molecule C=O formed by the the 3rd atom (carbon) and the 8th atom (oxygen). This carbon atom stands for an alkyl that before reaction bonds with hydroxyl -OH thereby forming an alcohol molecule. In the products, we see that the 7th atom (hydrogen from hydroxyl) cleaves from 8th atom (oxygen) and forms a new bond. However, we skip this and other minor changes in the PDDL code snippet below to focus on the purpose of this reaction.

```
(:action oxidationOfAlcoholsWithPCC
    :precondition (and (not (= ?h_9 ?h_7))
(not (= ?o_8 ?o_5)) (not (= ?o_8 ?o_4))
(not (= ?o_5 ?o_4)) (bond ?c_3 ?h_9)
 (bond ?c_3 ?o_8) (bond ?o_8 ?h_7)
  /* skip other preconditions */
                )
    :effect (and /* skip some effects */
(not (bond ?c_3 ?h_9)) (not (bond ?h_9 ?c_3))
(not (bond ?c_3 ?o_8)) (not (bond ?o_8 ?c_3))
(doublebond ?c_3 ?o_8)  (doublebond ?o_8 ?c_3)
(bond ?o_5 ?h_9) (bond ?h_9 ?o_5)
(not (bond ?o_8 ?h_7)) (not (bond ?h_7 ?o_8))
(bond ?o_4 ?h_7) (bond ?h_7 ?o_4)
            )
)
```

As described in Table 1, $oxidationOfAlcoholsWithPCC$ involves 16 parameters, while $aldolCondensation$ has 15 parameters. Since the initial state of the problem 5 has 62

atoms (see Table 2), instantiation of these 2 reactions requires significant memory. Therefore, when we took only 2 reactions required to solve p5, FD was not able to solve this problem on a computer with 32GB of memory, but could solve it when we eliminated from a reaction one of the hydrogen atoms that has been non-essential for this planning instance. Subsequently, we call this engineered version as a "p5 with missing hydrogen" problem.

4 Experiments

4.1 SAT-Based Planners

We use 3 different planners, SASE (Huang, Chen, and Zhang 2010), the version released on August 30, 2011, SatPlan (Kautz, Selman, and Hoffmann 2006), the version released in 2006, and Madagascar, version M (with default parameters), (Rintanen 2014; 2015), publicly available as a C code with the last modifications from February 2015, to encode and solve problems. For brevity, we refer to these 3 programs as "encoders" because they encode a given planning instance as SAT. The produced encodings were compared based on the number of variables, denoted as NoV, and number of clauses, denoted as NoC. However, we did not try to feed encodings to any of the SAT solvers.

Note that SatPlan gives the option to use 4 different encodings and they are: 1)action-based encoding, 2)gpstyle-action-based encoding, 3) gp-based encoding and 4) thin-bp-based encoding.

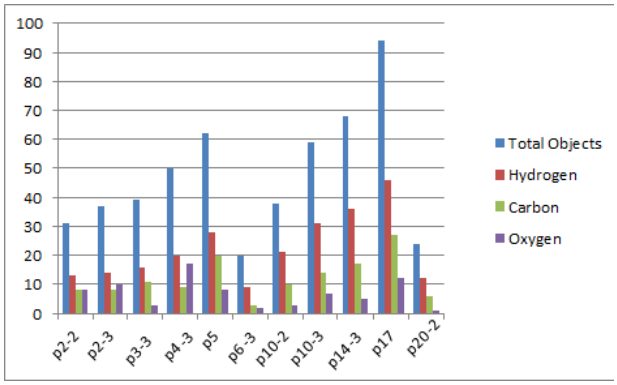


Figure 3: Diagram for number of atoms

Encoding Directly as CNF We started gathering results from Madagascar, SASE and SatPlan by generating a domain file that contained all the actions needed for the problems mentioned in Table 2. All experiments were performed on a server running a virtual machine with 2.80 GHz CPU, Ubuntu 14.04 and 128GB of RAM. The results were not very promising for the full problems. We were able to only get CNF encodings of some of the problems. All three encoders, either produced a very large CNF encoding or ran out of memory (Madagascar) when trying to generate an encoding. Table 3 and Table 4 show more information about the encodings generated by SASE and SatPlan, respectively.

Since full versions of some planning instances turned out to be too complex, 2 or 3 step versions were manually engineered to get better insights to why the full version was not solvable. These simpler versions include only atoms needed by the initial 2 or 3 reactions extracted from the full problem. Figure 3 displays the number of oxygen, hydrogen, carbon atoms and the total number of objects in each of the 2-3 step sub-problems. All 3 encoders, SASE, SatPlan and Madagascar were able to find a CNF encoding for problem p6-3 (3 step version of p6) and problem p20-2 (2 step version of p20). These encodings are promising since they were generated very quickly (under 0.1 seconds). Moreover, the number of variables and the number of clauses were also small as shown in Figure 4 which is based on the raw data from Table 5. Here, and subsequently, we only show encoding 1 for SatPlan since the other 3 encodings were very similar in both number of clauses and number of variables.

Problem	NoV	NoC
P2-Full	10079	782197
P4-Full	9784	742043
P5-Full-Hydrogen missing	460189	101711688
P6-Full	3148	95580
P10-1Step	132	340
P14-Full	2856	121267
P17-Full	out of memory	out of memory
P20-Full	15331	610997

Table 3: CNF encodings generated by SASE

Problem	NoV	NoC	Encoding
P2-Full	Not solved	Not solved	
P4-Full	Not solved	Not solved	
P5-Missing-H	Not solved	Not solved	
P6-Full	4318	879100	1
P6-Full	3958	1976336	2
P6-Full	4928	2027910	3
P6-Full	4928	713534	4
P10-1Step	12	50	1
P10-1Step	12	50	2
P10-1Step	44	242	3
P10-1Step	44	242	4
P14-Full	Not solved	Not solved	
P17-Full	Not solved	Not solved	
P20-Full	11244	12913007	1
P20-Full	9396	16728930	2
P20-Full	10494	16768636	3
P20-Full	10494	10978125	4

Table 4: CNF encodings generated by SatPlan

	P6-3	P20-2
SASE, NoV	445	264
SASE, NoC	2102	1221
Madagascar, NoV	511	176
Madagascar, NoC	3364	1212
SatPlan enc1, NoV	105	70
SatPlan enc1, NoC	866	652

Table 5: Metrics for encodings of 2-3 step problems

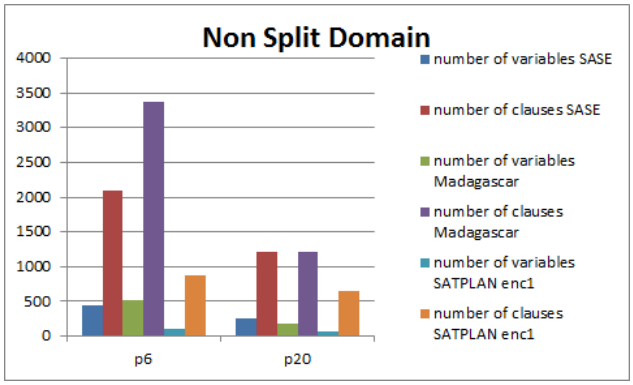


Figure 4: NoV and NoC for p6-3 and p20-2

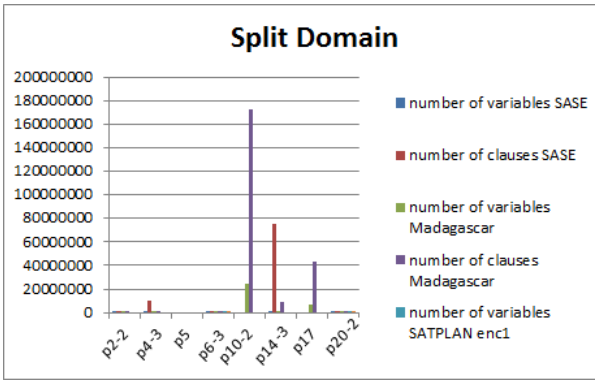


Figure 5: Metrics for encoding split domains

Splitting Domain then Encoding as CNF

	p2-2	p4-3	p6-3	p10-2	p14-3	p17-full	p20-2
SASE (NoV)	25166	221510	3145	1146178	7634		
SASE (NoC)	421664	10016148	15179	75020254	67342		
Madagascar	5112	45200	2756	24399420	1116294	7322944	598
Madagascar	79907	495665	24436	172364287	8671435	43172172	6593
SatPlan enc1	862	499					
SatPlan enc1	5710	38589					

Table 6: The odd line is NoV and the even line is NoC

Domain splitting was another option that we decided to explore. We decided to take all the actions needed for the problems mentioned in Table 1 and use action schema splitting mentioned in (Areces et al. 2014) to create a modified domain with $\gamma = 0.4$. This process splits an action with n parameters into many sub-actions. The minimum number of parameters for the sub-action is 2 when $\gamma = 0$ and n when $\gamma = 1$. According to (Areces et al. 2014), the advantage of this process is an exponential reduction of the number of ground actions when actions are instantiated. This also produced interesting results. The participating programs were able to generate CNF encodings for more problems. However, the encodings were very large and it took several hours in some cases to generate them. Figure 5 shows a graph based on the raw data for simplified 2-3 step problems from Table 6. The empty entries mean the problem could not be encoded within 128GB of RAM. When comparing the results for problems 6-3step and p20-2step, it is clear that the number of variables was much larger for the split domain with $\gamma = 0.4$ than it was for the un-split domain. Similarly, the number of clauses produced by the encoders is much larger when using the split domain with $\gamma = 0.4$. Apparently, the CNF encodings from the split domain had been larger, since each action was split into multiple sub-actions.

The only full problem that was actually solved by all 3 planners, SASE, Madagascar and SatPlan, was problem 6 (both with the split and unsplit domains). Note that the other full problems discussed above were only encoded as CNF.

Table 7 shows the number of variables and number of clauses in CNF generated when encoding full p6 by SASE, SatPlan and Madagascar. When comparing the data from Table 7 with Table 3 and Table 4, it is clear how much larger are the encodings produced for the exact same problem. Note that both tables are based on a domain containing all 24 actions from Table 1. The only difference is that in Table 7 the encodings were generated based on the split domain with $\gamma = 0.4$ while in Tables 3 and 4 the encodings were based on the un-split domain. The last column in Table 7 includes the total time it took for the planners to encode and then to actually compute a plan. Observe that Madagascar is faster thanks to its specialized SAT solver. The program from (Heifets and Jurisica 2012) took 31 seconds to solve p6-full on an IBM supercomputer. No other full problems could be encoded using the split domain.

CNF encodings	NoV	NoC	Time (sec)
SASE	54881	490548	12.441
Madagascar	30660	382723	0.829
SATPLAN enc1	25348	2062809	29.846

Table 7: Problem 6-full with the full split domain $\gamma = 0.4$

This set of our experiments led us to several hypotheses. First, encoding split domains as CNF is not beneficial, since CNF encodings of un-split domains are more compact. Second, for un-split domains, the evaluated planners based on reducing planning to SAT can encode only the simplest planning instances. There is little indication they can scale up to solve all of the benchmark problems. Apparently, the number of objects in benchmark problems has an adverse effect on the abilities of SAT-based planners to scale up. Only p6-full with the lowest number (32) of atoms could be solved.

4.2 Fast Downward on the Split Domains

The split domain did however improve FD's results when running over the same problems. Previously, when we were debugging benchmark problems, FD took many hours on smaller unsplit domains that contained only actions needed to solve the problem. The domains used here have been preprocessed so that they have the $distinct(?x, ?y)$ predicate instead of negation of equality $(not\ (= ?x\ ?y))$ since splitting software requires STRIPS domains before it can run. In this set of experiments, we run a version of FD released on October 9, 2015. FD was doing lazy greedy best-first search with FF heuristic. Table 8 shows the total **T**ime (in

seconds) and peak **M**emory (in KB) that FD used to solve the problems using split smaller domains extracted manually for each problem from the larger split domain. For example, the sub-domain used for the 2 step version of p2 consisted of only the 2 actions needed to solve this simplified problem.

Problem	T $\gamma=0.4$	M $\gamma=0.4$	T $\gamma=0.0$	M $\gamma=0.0$
2-2Step	0.0616	5180	0.072	5616
4-3Step	0.6739	31448	0.357	13684
6-3Step	0.0061	3348	0.006	3340
10-2Step	58.6161	538952	0.867	34928
14-3Step	10.1912	62720	0.989	16956
20-2Step	0.0109	3476	0.011	3472

Table 8: FD's Time and Memory results when using smaller split sub-domains with $\gamma=0.4$, $\gamma=0.0$

The results in Table 8 demonstrate that for smaller domains and short, 2-3 step, planning problems splitting was effective and the finer split domains ($\gamma=0.0$) consumed less time and memory than the coarser split domain ($\gamma=0.4$).

To investigate the full problems, we decided to look carefully into the full domains using different γ values to see if different values improved the time it takes to solve a problem. We produced 2 more split domains using $\gamma=0.0$ and $\gamma=0.2$, in addition to the domain with $\gamma=0.4$ that we had before. The original full unsplit domain has 24 action, while the split domains with a gamma values of 0.0, 0.2 and 0.4 have 424, 336 and 228 actions, respectively. To see whether splitting would improve the time it took for a problem to be solved by FD, we obtained the following data for the full problems 4,6 and 14 based on the full split domains.

P4	Total Time (Sec)	Peak Memory(KB)
non-split domain	Not Solved	Not Solved
split domain $\gamma=0.4$	Not Solved	Not Solved
split domain $\gamma=0.2$	45.7	78112
split domain $\gamma=0.0$	18.4	62521
P6	Total Time (Sec)	Peak Memory(KB)
non-split domain	49.3	275322
split domain $\gamma=0.4$	42.5	223080
split domain $\gamma=0.2$	39.1	115232
split domain $\gamma=0.0$	34.9	74256
P14	Total Time (Sec)	Peak Memory(KB)
non-split domain	Not Solved	Not Solved
split domain $\gamma=0.4$	Not Solved	Not Solved
split domain $\gamma=0.2$	22546.4	2262432
split domain $\gamma=0.0$	4306.6	2486320

Table 9: FD: problems 4,6 and 14 split/unsplit full domains (domains containing 24 actions)

Table 9 shows some advantages of the finer split domains, since problems 4, 6 and 14 took less time for smaller γ values. However, the other full problems, i.e. 2,3,5,10,17 and 20 were not solved for any of the full split domains. All these problems need more than 128GB of RAM. Therefore, splitting was not completely successful even for $\gamma=0.0$.

The advantages of splitting were not always demonstrated. P20 (full, 11-step version) was not solved by FD with neither the split nor the unsplit full domains due to its complexity. We decided to create 2 smaller sub-domains, where only the actions needed for problem 20 are present, one based on the split domain with a $\gamma=0.4$ and one based on the split domain with a $\gamma=0.0$. Table 10 shows the time, memory and the number of initial candidates FD produced.

P20	Total Time (Sec)	Peak Memory(KB)	Initial Candidates
non-split	1.31	23368	6
$\gamma=0.4$	2750	3698780	74
$\gamma=0.0$	38.8	80076	93

Table 10: Smaller split and unsplit sub-domains: p20

The number of initial candidates is a metric that is produced by FD based on the domain and problem given. The split domain has many more initial candidates that FD needs to explore than the unsplit domain. This is the reason why FD takes more time and needs more memory to find the answer when solving the long planning problem using the split domain. As you can see in Table 10, FD needs to only check 6 initial candidates when using the unsplit domain which consumes less time and memory. Also, for $\gamma=0.4$, the domain has more initial candidates than the unsplit domain, but more complex (more parameters) reactions than for $\gamma=0.0$, so it is has the worst of both worlds. It is mixing the worst things of the other two cases into one. Notice also that in the split domain the length of the plan is much longer than in the un-split domain because a greater number of sub-actions should be executed sequentially. Therefore, in the split domain, finding a plan can take more time. Overall, splitting increases the number of initial candidates, but simplifies the actions. This example demonstrates shows the limitations of splitting in some situations.

Simplifying the domain to improve time and memory usage. It was interesting to investigate how much performance of FD would improve, if a few atoms are removed from the domain reactions thereby alleviating in part the grounding problem. In some reactions, there are hydrogen atoms that remain invariant, i.e, the reactions have no effects on their bonds with carbon atoms. Since they do not really participate in reactions, removing a few of them is harmless, in a sense that the planning instances still should be solvable after doing this minor modification. The data that we collected in Table 11 show that by removing hydrogen atoms and their bonds with carbons, the time and memory improved, but with varying efficiency.

P4	Total Time (Sec)	Peak Memory(KB)
FullDomain $\gamma=0.0$	18.4	62521
Domain $\gamma=0.0$ with 1 H removed	16.7	40632
P6		
FullDomain $\gamma=0.0$	34.9	74256
Domain $\gamma=0.0$ with 1 H removed	33	72772
Domain $\gamma=0.0$ with 2 H removed	16.7	39528
P14		
FullDomain $\gamma=0.0$	4306.6	2486320
Domain $\gamma=0.0$ with 1 H removed	773	513524

Table 11: Removing hydrogens to improve time/memory

Heuristics All data generated by FD above are collected from the lazy_greedy(ff) search. We experimented with a

Heuristic	P4	P6	P14
eager_greedy, ff	95 s	82 s	36.4 h
lazy_greedy, add	33.8 s	19 s	69946 s
lazy_greedy, ipdb	–	–	–
lazy_greedy, lmcount	–	163 s	–
lazy_greedy, max	–	–	–
lazy_greedy, merge&shrink	–	–	–
astar, add	37 s	19 s	73591 s
astar, ipdb	–	–	–
astar, lmcount	–	3.2 h	–
astar, max	–	–	–
astar, merge&shrink	–	–	–

Table 12: Time for P4, P6 and P14 using different search

Heuristic	P2-2	P6-3	P10-2	P14-3	P20-2
eager_greedy, ff	0.12	0.52	0.998	1.05	0.0117
lazy_greedy, add	0.08	0.02	0.93	1.04	0.010
lazy_greedy, ipdb	–	0.052	–	–	–
lazy_greedy, lmcount	–	0.10	–	–	0.071
lazy_greedy, max	–	–	–	–	–
lazy_greedy, m&s	–	11.7	–	–	–
astar, add	0.09	0.4	0.9	1.2	0.011
astar, ipdb	–	0.05	–	–	–
astar, lmcount	–	0.19	–	–	0.08
astar, max	–	–	–	–	–
astar, m&s	–	–	–	–	–

Table 13: Time (sec) for 2-3 step problems per heuristic

few different heuristics and search combinations using the finest split domain ($\gamma = 0.0$). As you can see in Table 12, eager_greedy(ff) was only able to solve problems 4 and 6 in a reasonable time but took over 36 hours to solve problem 14. The astar(add) optimal search performed well: FD solved problems 4, 6, 14 in 37, 19, 73591 seconds, respectively. However, it was still slower than lazy greedy with ff: see Table 9 above. The A^* search with ipdb, max and the merge and shrink heuristics produced negative results; "–" means FD was not able to solve the problem in a reasonable time. FD doing A^* search with lmcount took over 3 hours to solve problem 6, and was not able to solve problems 4 and 14. In comparison, lazy_greedy performed better. For example, when using lmcount as the heuristic, P6 was solved in 163 seconds, which is a huge improvement in time when compared to 3.2 hours for A^*. A similar pattern can be seen for the 2-3 step problems: see Table 13. Only eager greedy with ff and astar(add) were able to solve the 2-3 step problems, but lazy greedy with ff did the best when compared with the other options. These data are somewhat preliminary, but they show that this new domain can serve as a challenge for researchers developing heuristics.

5 Gamer and L-RPG

Gamer (Kissmann and Edelkamp 2011; Kissmann 2012) is a symbolic planner based on binary decision diagrams. Unfortunately, it was not possible to build Gamer on a 64 bit machine. However, Gamer did run on a 32 bit machine with 4GB of RAM. It was able to solve only P2-2step, P4-3step and P6-3step using sub-domains containing only the actions needed for the problems. The actions in the domains were split with $\gamma = 0.0$ for best results. The same 3 problems were not solved when a non split domain was used. Furthermore, none of the full problems were solved using split and non split domains. It is possible that the reason for not solving full problems is the limited available RAM. However, the fact that the 2-3step problems were not solved when using the non split domain does not support that.

L-RPG (Ridder 2015; 2014; Ridder and Fox 2014) is a forward-chaining planner that does not rely on grounding. The planner was developed as a solution to memory constraints issues that other state-of-the-art planners have. L-RPG looks like a promising solution to the benchmark problems. However, it was not possible to solve any of the planning instances due to bugs found in the code of L-RPG.

6 Conclusion and Future Work

FD's results for problems 4, 6 and 14 are somewhat comparable in terms of time with data reported in Table 1 from (Heifets and Jurisica 2012), despite differences in hardware. FD was able to solve problems 4 and 6 using the finest split domain slightly slower than the proof-number search on an IBM supercomputer that took 15sec and 31sec, respectively. However, FD was able to solve problem 14 faster using the split domain ($\gamma = 0.0$); the proof number search on an IBM super-computer took 5138sec vs FD's 4306sec as reported in our Table 9 above. However, FD could not solve problems 2,3,5,10 within 128GB of memory even using split domains.

We can see from the results of the experiments that encoding the problems as CNF did not generate good results. The produced encodings were very large due to grounding. However, we have demonstrated that actions schema splitting transformation is practically useful since splitting did improve FD's results when compared to unsplit domains, at least for some problems. At the same time, splitting helped only with solving a few of the full benchmark problems. Other ideas should be explored before a planner can reliably compete with undergraduate students in solving the benchmark problems. The proposed benchmark remains a serious challenge for modern AI planners.

There are several future work possibilities. The research in LRPG(Ridder 2015) can help because lifted planning can be promising with the proposed domain. Similarly, lifted encodings (Kautz, McAllester, and Selman 1996) should be explored. Moreover, reducing planning to QBF may be beneficial for this planning domain (Cashmore, Fox, and Giunchiglia 2013; Cashmore 2013).

7 Acknowledgement

R.M. (the first author) would like to thank the Undergraduate Program of the Computer Science Department at Ryerson University for providing the opportunity to take part in the undergraduate thesis course. We would also like to thank the Department of Computer Science for providing access to a cloud based server, which enabled us to run our experiments on a Linux-based virtual machine that has been allocated 128GB of RAM.

References

Accelrys. 2011. *CTfile Formats*. Accelrys. http://download.accelrys.com/freeware/.

Areces, C.; Bustos, F.; Dominguez, M. A.; and Hoffmann, J. 2014. Optimizing planning domains by automatic action schema splitting. In Chien et al. (2014).

Bøgevig, A.; Federsel, H.-J.; Huerta, F.; Hutchings, M. G.; Kraut, H.; Langer, T.; Löw, P.; Oppawsky, C.; Rein, T.; and Saller, H. 2015. Route design in the 21st century: The IC-SYNTH software tool as an idea generator for synthesis prediction. *Organic Process Res. & Developm.* 19(2):357–368.

Cashmore, M.; Fox, M.; and Giunchiglia, E. 2013. Partially grounded planning as quantified boolean formula. In Borrajo, D.; Kambhampati, S.; Oddi, A.; and Fratini, S., eds., *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling, ICAPS 2013, Rome, Italy, June 10-14, 2013*. AAAI.

Cashmore, M. 2013. *Planning as Quantified Boolean Formulae*. Ph.D. Dissertation, University of Strathclyde, Dept. of Computer and Information Sciences.

ChemAxon. 2015. JChem software. http://www.chemaxon.com.

Chien, S. A.; Do, M. B.; Fern, A.; and Ruml, W., eds. 2014. *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2014, Portsmouth, New Hampshire, USA, June 21-26, 2014*. AAAI.

Cook, A.; Johnson, A. P.; Law, J.; Mirzazadeh, M.; Ravitz, O.; and Simon, A. 2012. Computer-aided synthesis design: 40 years on. *Wiley Interdisciplinary Reviews: Computational Molecular Science* 2(1):79–107.

Edelkamp, S., and Hoffmann, J. 2004. PDDL2.2: The Language for the Classical Part of the 4th Intern. Planning Competition. Technical Report 195, Universität Freiburg, Institut für Informatik.

Ernst, M.; Millstein, T.; and Weld, D. 1997. Automatic sat-compilation of planning problems. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, IJCAI 97, Nagoya, Japan, August 23-29, 1997, 2 Volumes*, 1169–1177.

Heifets, A., and Jurisica, I. 2012. Construction of new medicines via game proof search. In Hoffmann, J., and Selman, B., eds., *AAAI*, 1564–1570. AAAI Press.

Heifets, A. 2012. *Benchmark problems, 2012*. Available at http://www.cs.toronto.edu/~aheifets/ChemicalPlanning/.

Helmert, M., and et al. 2015. *The Fast Downward Planning System*.

Huang, R.; Chen, Y.; and Zhang, W. 2010. A novel transition based encoding scheme for planning as satisfiability. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*.

James, C.; Weininger, D.; and Delany, J. 2011. *Daylight Theory Manual Ver. 4.9 (08/01/11)*. http://www.daylight.com/dayhtml/doc/theory/index.html.

Judson, P. 2009. *Knowledge-Based Expert Systems in Chemistry: Not Counting on Computers*. RSC Theoretical and Computational Chemistry. Royal Society of Chemistry.

Kautz, H. A., and Selman, B. 1992. Planning as satisfiability. In *ECAI*, 359–363.

Kautz, H. A.; McAllester, D. A.; and Selman, B. 1996. Encoding plans in propositional logic. In *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning (KR'96), Cambridge, Massachusetts, USA, November 5-8, 1996.*, 374–384.

Kautz, H.; Selman, B.; and Hoffmann, J. 2006. SATPLAN: Planning as Satisfiability. In *Abstracts of the 5th International Planning Competition, 2006*. http://www.cs.rochester.edu/users/faculty/kautz/satplan/.

Kissmann, P., and Edelkamp, S. 2011. Gamer, a general game playing agent. *KI* 25(1):49–52.

Kissmann, P. 2012. *Symbolic Search in Planning and General Game Playing*. Ph.D. Dissertation, Universität Bremen, Germany. http://nbn-resolving.de/urn:nbn:de:gbv:46-00102863-15.

Masoumi, A.; Antoniazzi, M.; and Soutchanski, M. 2015. Modeling organic chemistry and planning organic synthesis. In *Proceedings of the 1st Global Conference on Artificial Intelligence (GCAI 2015)*, volume 36, 176–195.

Ravitz, O. 2013. Data-driven computer aided synthesis design. *Drug Discovery Today: Technologies* 10(3):e443 – e449. http://www.chemplanner.com.

Ridder, B., and Fox, M. 2014. Heuristic evaluation based on lifted relaxed planning graphs. In Chien et al. (2014).

Ridder, B. C. 2014. *Lifted Heuristics: Towards More Scalable Planning Systems*. Ph.D. Dissertation, Kings College, Department of Informatics, London, UK.

Ridder, B. 2015. *L-RPG: Introducing a Lifted Forward-Chaining Planner*. Available at https://www.assembla.com/spaces/MyPOP/subversion/source.

Rintanen, J. 2014. Madagascar: Scalable Planning with SAT; an overview of the techniques in the Madagascar (M, Mp, MpC) planners. In *International Planning Competition*. https://users.ics.aalto.fi/rintanen/papers/Rintanen14IPC.pdf.

Rintanen, J. 2015. *Madagascar software*. http://users.ics.aalto.fi/rintanen/MADAGASCAR.TAR.

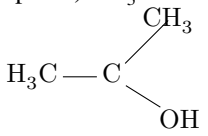
Szymkuć, S.; Gajewska, E. P.; Klucznik, T.; Molga, K.; Dittwald, P.; Startek, M.; Bajczyk, M.; and Grzybowski, B. A. 2016. Computer-assisted synthetic planning: The end of the beginning. *Angewandte Chemie International Edition* 55(20):5904–5937.

Appendix

In organic chemistry, many molecules are compounds of bonded carbon and hydrogen atoms and each carbon atom has the valence 4. To simplify the figures, it is usual to assume that all unnamed nodes are carbon atoms with the appropriate number of hydrogen atoms attached by default.

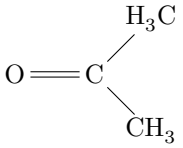
To explain what happens in p5, we show main molecules participating in the solution to p5. We skip here PCC (two

of them are present in the initial state of p5) and other small molecules since they have been already discussed in Preliminaries section. In the initial state of p5, one of the molecules participating in oxidation reaction is isopropyl alcohol[2] (isopropanol) $CH_3CH{-}OH{-}CH_3$

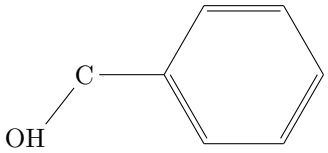


Isopropyl alcohol

The product of the first reaction is acetone $(CH_3)_2CO$, or propanone, a compound used in many households.[3]



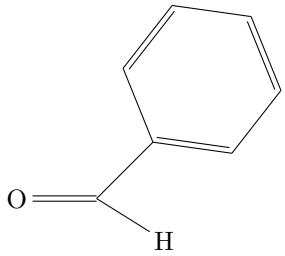
Acetone



Benzyl alcohol

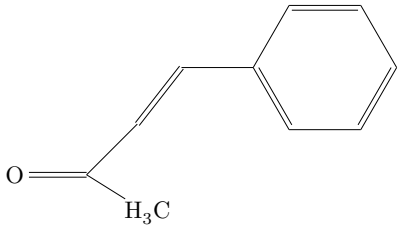
The second oxidation reaction takes benzyl alcohol (known also as phenylmethanol) molecule $C_6H_5CH_2OH$ and produces a benzaldehyde[4] molecule C_6H_5CHO. This molecule is called benzyl alcohol because it has a hydroxyl group $-OH$ attached to the carbon atom that also has a single bond with the phenyl ring. The phenyl ring can be viewed as a benzene ring, minus a hydrogen. It is often called aromatic ring because all (the assumed, present by default) carbons around the ring have aromatic bonds with each other. It is common to draw this aromatic ring with a sequence of alternating single and double bonds, while in reality they are aromatic bonds. The carbon attached to the hydroxyl group has two other assumed single bonds with hydrogen atoms present by default.

The name benzaldehyde can be explained by the presence of an aromatic phenyl ring that is attached to a carbon atom. This carbon has a single bond with a hydrogen atom, and a double bond with an oxygen atom thereby forming the carbonyl group C=O. Carbonyl containing compounds with the bond to hydrogen are known as aldehyde. Benzaldehyde is simplest aromatic aldehyde; it is industrially useful.



Benzaldehyde

The last 3rd reaction, $aldolCondensation$ reaction,[5] takes acetone, which is a simplest ketone, takes benzaldehyde, which is an aldehyde, takes sodium hydroxide NaOH, also known as lye and caustic soda, and produces the target molecule, named 4-phenylbut-3-en-2-one, also known as benzalacetone, benzylideneacetone,[6] which is a ketone.



Benzylideneacetone

The following PDDL code provides detailed and complete description of this last reaction.

```
(:action aldolCondensation
    :parameters (?o_10 - oxygen ?o_4 - oxygen
?h_11 - hydrogen ?na_14 - sodium ?c_1 - carbon
?c_5 - carbon ?o_15 - oxygen ?h_8 - hydrogen
?h_9 - hydrogen  ?h_12 - hydrogen
?h_13 - hydrogen ?r1_2 - object
?r1_3 - object ?o_7 - oxygen ?c_6 - carbon)
    :precondition (and (not (= ?c_1 ?c_5))
 (not (= ?h_11 ?h_8)) (not (= ?h_11 ?h_9))
 (not (= ?h_8 ?h_9)) (not (= ?o_10 ?o_4))
 (not (= ?o_10 ?o_15)) (not (= ?o_4 ?o_15))
(not (= ?h_11 ?h_12)) (bond ?o_10 ?h_11)
 (bond ?o_10 ?h_12) (bond ?na_14 ?o_15)
(bond ?h_13 ?o_15) (not (= ?r1_2 ?r1_3))
 (bond ?r1_2 ?c_1) (bond ?r1_3 ?c_1)
(doublebond ?o_4 ?c_1) (not (= ?h_9 ?h_8))
 (not (= ?c_6 ?c_5))
(bond ?c_6 ?c_5) (bond ?h_9 ?c_5)
(bond ?h_8 ?c_5) (doublebond ?o_7 ?c_6))
    :effect (and (not (bond ?o_10 ?h_11))
(not (bond ?h_11 ?o_10)) (bond ?o_10 ?h_8)
 (bond ?h_8 ?o_10) (bond ?h_11 ?o_4)
(bond ?o_4 ?h_11) (not (bond ?na_14 ?o_15))
(not (bond ?o_15 ?na_14)) (bond ?na_14 ?o_4)
 (bond ?o_4 ?na_14) (bond ?h_9 ?o_15)
(bond ?o_15 ?h_9) (not (doublebond ?o_4 ?c_1))
 (not (doublebond ?c_1 ?o_4))
 (doublebond ?c_1 ?c_5)(doublebond ?c_5 ?c_1)
 (not (bond ?h_8 ?c_5))(not (bond ?c_5 ?h_8))
 (not (bond ?h_9 ?c_5))(not (bond ?c_5 ?h_9) )
)
)
```

[2] https://en.wikipedia.org/wiki/Isopropyl_alcohol

[3] https://en.wikipedia.org/wiki/Acetone

[4] https://en.wikipedia.org/wiki/Benzaldehyde

[5] https://en.wikipedia.org/wiki/Aldol_condensation

[6] https://en.wikipedia.org/wiki/Benzylideneacetone

Planning Machine Activity Between Manufacturing Operations: Maintaining Accuracy While Reducing Energy Consumption

Simon Parkinson, Mauro Vallati
Lukas Chrpa
Department of Informatics
School of Computing and Engineering
University of Huddersfield, UK
s.parkinson@hud.ac.uk

Andrew Longstaff, Simon Flectcher
Centre for Precision Technologies
School of Computing and Engineering
University of Huddersfield, UK

Abstract

There has recently been an increased emphasis on reducing energy consumption in manufacturing. This is largely because of fluctuations in energy costs causing uncertainty. The increased competition between manufacturers means that even a slight change in energy consumption can have implications on their profit margin or competitiveness of quote. Furthermore, there is a drive from policy-makers to audit the environmental impact of manufactured goods from cradle-to-grave. The understanding, and potential reduction of machine tool energy consumption has therefore received significant interest as they require large amounts of energy to perform either subtractive or additive manufacturing tasks.

One area that has received relatively little interest, yet could harness great potential, is reducing energy consumption by optimally planning machine activities while the machine is not in operation. The intuitive option is to turn off all non-essential energy-consuming processes. However, manufacturing processes such as milling often release large amounts of heat into the machine's structure causing deformation, which results in deviation of the machine tool's actual cutting position from that which was commanded, a phenomenon known as thermal deformation. A rapid change in temperature can increase the deformation, which can deteriorate the machine's manufacturing capability, potentially producing scrap parts with the associated commercial and environmental repercussions. It is therefore necessary to consider the relationship between energy consumption, thermal deformation, machining accuracy and time, when planning the machine's activity when idle, or about to resume machining.

In this paper, we investigate the exploitability of automated planning techniques for planning machine activities between subtractive manufacturing operations, while being sufficiently broad to be extended to additive processes. The aim is to reduce energy consumption but maintain machine accuracy. Specifically, a novel domain model is presented where the machine's energy consumption, thermal stability, and their relationship to the overall machine's accuracy is encoded. Experimental analysis then demonstrates the effectiveness of the proposed approach using a case study which considers real-world data.

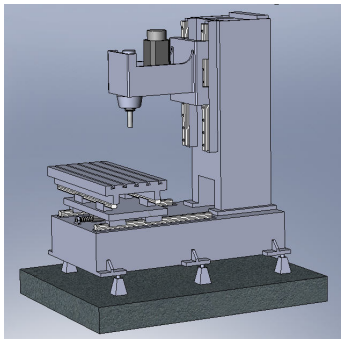


Figure 1: Example C-Frame three-axis machine tool

Introduction

Machine tools are complex mechantronic system used in both subtractive and additive manufacturing. Much of their performance comes from their mechanical rigidity. For example, Figure 1 illustrates the structure of a three-axis machine tool. Machine tools come in a large variety of sizes and configurations, but a common feature is their ability to position their tool in a three-dimensional space relative to the workpiece either to remove (cut, grind, etc.) or add material. Accuracy is often a primary commercial driver in the advancement of machine tools for precision, high-value manufacturing to micrometre-level tolerances. However, maintaining such high levels of accuracy requires strict control of the many factors which can cause a change in accuracy. For example, the effect of temperature change on the machine's structure can have a dramatic impact on the accuracy of the tool. Energy efficiency is also becoming an increasingly important factor in machine tool development both to reduce manufacturing costs (Draganescu *et al.* 2003; Diaz *et al.* 2011), as well as reducing environmental impact (Diaz *et al.* 2010). However, the relationship between the improvement in energy efficiency and possible reduction in machine accuracy resulting from rapid temperature change is less well explored. This is surprising considering the amount of heat generated from electrical devices and mechanical subsystems during the machining process.

The use of machine tools has been identified as the largest consumer of energy during the manufacturing of parts. It has been established that machine tools use 63% of the to-

tal energy required to manufacture a part (Hesselbach and Herrmann 2011). Additionally, energy consumption occupies over 20% of the operating costs of machine tools per year, in excess of £10,000. While it is difficult to state how much of the 20% is consumed between manufacturing operations, it is likely that the machine will be stationary for many periods during the working-day as new parts are loaded, etc. Many researchers have investigated the potential of reducing energy consumption during the manufacturing process itself (Vijayaraghavan and Dornfeld 2010; Liu *et al.* 2014; 2015). For example, reducing energy usage during milling (Diaz *et al.* 2011). These works have largely been motivated by the fact that large forces are required to cut material, and any reduction at this stage can therefore be significant. However, one area that has received less attention is the consumption of energy between manufacturing operations, when the machine is not cutting and therefore is nominally idle. In the first instance it may appear that if the machine is idle it will be consuming no energy. However, it is often the case that many electrical components of a machine tool will continue to use energy. Furthermore, once the machine is required to operate once again, an energy-intensive warm-up cycle is often required to bring the subsystem (e.g spindle motor) into a suitable (stable) state for actual machining.

Such warm-up cycles are often required since the heat generated from the machine components during manufacturing will transfer to the machine tool's structure and cause deformation. This thermal deformation is, in the simplest case, a first-order response to the temperature step input. Heating the subsystem prior to manufacturing means that much of the deformation will take place before cutting begins, helping to reduce in-process change and increasing the accuracy of the component. A warm-up cycle is usually energy intensive, but will only be necessary should the heat-generating subsystem and surrounding structure decrease below an identified temperature. This creates an interesting possibility where keeping the machine subsystems active, at a reduced level, whilst not manufacturing can generate sufficient heat to maintain the thermal stability of the machine tool's structure and remove the need for a warm-up cycle, thus reducing the overall energy consumption. For example, Figure 2 illustrates, through Finite Element Analysis (FEA), the deformation of the machine tool's structure resulting from the release of heat generated by the spindle motor and friction in the moving mechanical elements. In the diagram, the nominal tool position and orientation are shown superimposed on the actual location; the difference between the two, caused by temperature effects, leads to a displacement at the tool tip, which is known as the thermal error. If this deformation were to take place during the manufacturing process then the resultant manufactured component would display the results of this error, leading to a requirement for rework or even scrapped parts. However, if this deformation were to occur before the manufacturing process, then the thermal error of the machine can remain stable during the process, and therefore the accuracy of the produced part is largely unaffected.

Energy consumption information for many machine tool

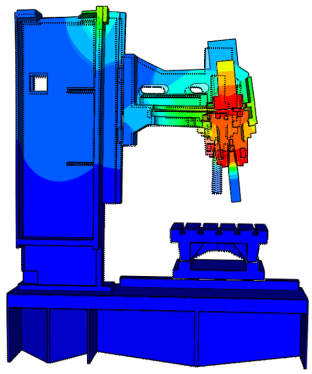


Figure 2: Deformation of the machine tool's structure due to heat generated by the spindle motor

electrical subsystems is widely available, but that from mechanical interaction (friction) is often less well defined. However, in both cases the amount of heat released into the machine's structure and its affect on machine accuracy needs to be established. This can be acquired by recording the temperature of the machine tool's structure while monitoring the deviation of the machine tool's cutting point. During analysis, each subsystem will typically be run at different speeds to establish the relationship between the different levels of energy consumption and heat generation, and also the relationship between heat generation at the subsystem's location and the effect on machine accuracy. Once all the data has been acquired, FEA can be used to computationally model the relationship between heat generation and deformation of the machine tool (Mian *et al.* 2011; 2013). This model can then be used to derive a series of coefficients that describe the generation of heat with increased energy consumption, and the change in machine tool accuracy from the resulting different thermal gradients.

The number of electrical subsystems, the different operational levels, the current state of the machine tool, and the required initial state of the next manufacturing operation make it challenging to consider all possible options and minimise energy consumption whilst maintaining a desired level of accuracy. This creates an interesting and novel possibility to utilise Automated Planning to automate the process, removing the requirement for expert knowledge, minimise energy consumption, and maintain the required level of accuracy. While the exploitation of planning techniques for planning machine activities between manufacturing operations has never been investigated, previous works demonstrated the potential of using automated planning for optimising different aspects of using machine tools. For example, the non-productive time (downtime) of a machine tool during calibration has been reduced through automatically constructing calibrations plans, reducing reliance on expert knowledge (Parkinson *et al.* 2012a; 2012b). Further work of encoding mechanisms to calculate measurement uncertainty (Parkinson *et al.* 2014b) created the potential to perform multi-objective optimisation (Parkinson *et al.* 2014a).

This paper is organised as follows: first, the importance of planning for activity between manufacturing operations

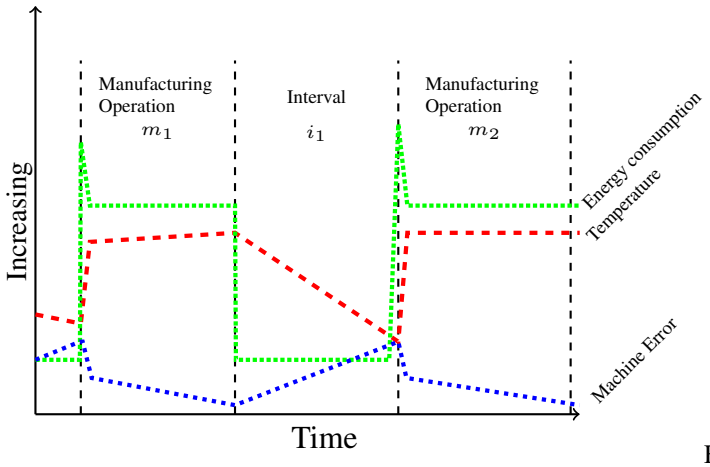


Figure 3: Illustrating how the machine tool energy consumption, structural temperature, and accuracy is changing during manufacturing and interval periods.

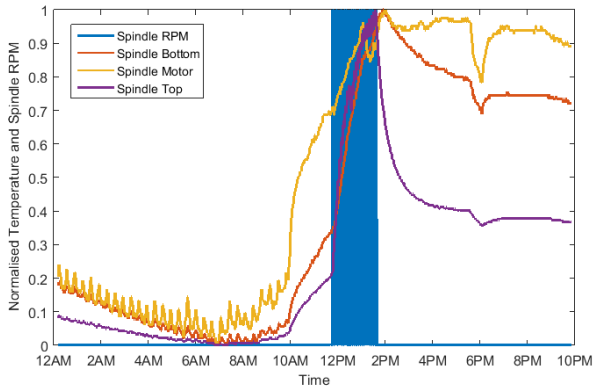


Figure 4: Heat generated by spindle motor during a two-hour heating and cooling cycle. Spindle bottom, top and motor indicate the normalised temperature for three surface temperature sensors mounted around the spindle. Spindle RPM reports the normalised spindle speed in RPM (0 to 9000).

(named *interval activity* herein) is described and motivated. Second, we provide a discussion on the importance of planning interval activity, and a domain model is provided, encoded using the Planning Domain Definition Language (PDDL) (Fox and Long 2003). Then, the effectiveness of automated planning is demonstrated using a real-world case study. Finally, conclusions are given.

Importance of Interval Activity

In this section, the importance of considering interval activity is motivated. In particular, the relationship that is of interest in this paper is that between energy consumption, generation of temperature profile, and the affect on machining accuracy. Prior knowledge of this relationship creates the potential to optimise machine tool use between manufacturing operations. For example, in some situations, it may be advantageous to keep the electronic components in use to maintain energy consumption, generate heat, and thus maintain machine accuracy.

Figure 3 provides a graphical illustration of two manufacturing operations with an interval between. The figure illustrates the relationship between increasing energy consumption (green), heat generation (red), and increasing machine error (blue) through a simplified representation. Note that although the figure is for illustration purposes, the data is a realistic, if simplified, representation of what occurs. In the figure, it can first be seen that energy consumption is at its lowest when the machine is idle, and its highest when a new manufacturing job is started, This is because a dedicated warm-up cycle is required to stabilise the machine's structure and avoid thermal change during manufacturing. It is then noticeable in the figure that as the energy consumption increases, so does the temperature of the machine's structure. The final relationship presented is that the error of the machine tool increases to a steady-state value and maintained when the temperature is stable. In practice, the number of different operations that occur during machining mean that the energy profile, and resulting temperature and error trends, will display somewhat more complex behaviour. In the remainder of this section, a more detailed analysis of each element (energy consumption, temperature and accuracy) is presented.

The relationship between energy consumption, the generation of heat and its dissipation into the machines structure is different for every subsystem: a high-speed spindle motor uses significantly more power than a linear axis servo motor. For example, consider manufacturing an aluminium housing with the dimensions of 150mm $\times$ 50mm $\times$ 20mm (Heidenhain 2010). The total energy needed for the machine tool to produce the part is 20.4kW. A total of a 4.8kW for the machine tool spindle, and 0.5kW for the three axes' feed drives. Other electrical subsystems (e.g controller, coolant pump, etc.) make up the remainder. As both these components have different levels of energy consumption, they generate different amounts of heat. Each component will have different modes of operation. For example, a common spindle motor might be capable of speeds in excess of 9,000 revolutions per minutes (RPM). Figure 4 demonstrates the heat generated as the spindle speed increases on a three-axis machine tool. The figure shows the normalised spindle speed in RPM (0 to 9000), and normalised temperature for three surface temperature sensors mounted around the spindle. There are: (1) spindle bottom (21.8°C to 27.4°C), (2) spindle motor (21.6°C to 26.2°C), and (3) spindle motor (21.7°C to 33.1°C). The graph illustrates that when the spindle is used at its higher speed, the temperature of the machine tool's structure surrounding the spindle increases rapidly in temperature. Once high speed usage has finished, it can be seen that the structure of the machine tool begins to reduce in temperature.

The next relationship of interest is that of changing machine tool temperature and its effect on structural deformation of the machine tool. The heat generated by machine subsystems transfers into the machine tool's structure causing distortion. The severity of the effect of changing tem-

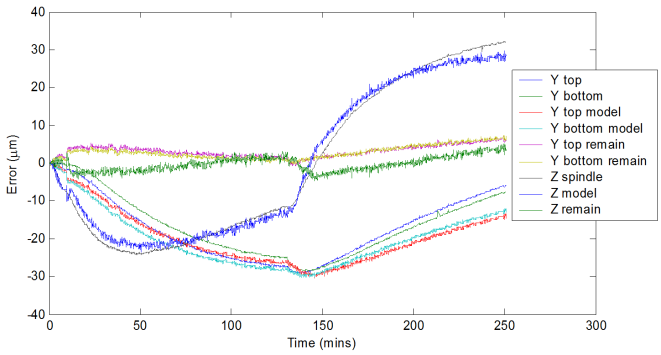


Figure 5: Error of a three axis machine tool generated during a two hour spindle heating and cooling cycle.

perature is dependent on the material from which it is constructed. For example, steel has a high coefficient of thermal expansion (~12µm per °C) compared to carbon fibre (~2µm per °C), though much less than aluminium (~22µm per °C). Considering the previous example of a spindle motor, Figure 5 illustrates the effect of changing temperature on the machine's structure. In this figure the spindle of a three-axis machine tool was running at 9,000 RPM (70% utilisation) for 120 minutes, and then left to cool for a further 120 minutes. From this experiment, it is noticeable that error for each of the three axes is changing throughout the heating cycle in the first 120 minutes, and then once the spindle is disabled, the errors continue to increase as the heat is still transferred into the machine's structure.

The examples discussed in this section demonstrate the importance of planning for machine activity between manufacturing operations. However, planning in this context is not a trivial task as any action can impact on machine accuracy and energy usage, both of which can have significant financial implications. Currently it is up to the machine operator to make the correct decision in an ad-hoc manner where they determine machine activity by knowing future manufacturing operations, as well as the energy saving policies of their manager. However, this planning for the machine operator is complicated by the large number of different machine activity actions that can be performed and their potential implications on machine accuracy and energy consumption. For example, each axis and spindle can be moved at different speeds sequentially or concurrently for different periods of time. Moving a single linear axis will transfer heat in the machine's structure surrounding the axis and would result in thermal distortion from that location, whereas moving all three axes simultaneously would transfer heat into more of the machine's structure and potentially result in more symmetrical expansion.

Domain Modelling

In this section, a PDDL model is developed and discussed to describe the domain of interval planning. In the presented model, the two following equations are used to determine energy consumption as well as machine accuracy. These equations require machine-specific data acquired through performing an error mapping and energy monitoring audit.

$$total\ error = total\ error + duration \times (effect\ on\ error \times energy\ consumption) \quad (1)$$

$$total\ energy = total\ energy + (duration \times energy\ consumption) \quad (2)$$

Equation 1 is used for updating the error fluent by a quantity of time in minutes, multiplied by the the effect on error in micrometres of deviation per minute of energy consumption. Here, there is a different effect on error value for each different mode of operation. Equation 2 updates the energy consumption fluent by the same duration (in minutes) multiplied by the a fluent storing the energy use of a particular component when being used in a predefined mode of operation.

The use of predefined modes has been adopted to reduce the size of the domain model, in terms of number of operators, and make it easier to be handled by state-of-the-art planning engines. Many machine subsystems, such as the spindle motor, can be run at any speed between stationary and their maximum RPM. This continuous behaviour could be encoded in PDDL+ (Fox and Long 2006); however, this would dramatically increase domain complexity as the number of heat-generating machine components increases. In addition, the number of planning systems able to handle PDDL+ is limited (see, e.g. (Coles and Coles 2014; Della Penna *et al.* 2009)), especially when compared with those capable of handling different versions of PDDL. Even more restricted is the number of solvers able to support the entire function set of PDDL+. Therefore, for the preliminary work undertaken to determine the feasibility of using automated planning in this domain, PDDL2.2 (S. Edelkamp and J. Hoffmann 2004) –an extension of PDDL2.1 (Fox and Long 2003)– is used. The International Planning Competition[1] has resulted in the existence of a significant number of planners able to solve PDDL2.2 planning problems.

Initial and Goal State

The initial state specifies energy consumption and effect on machine accuracy for each predefined level of operation through the use of numeric fluents. For example, `energy_idle ?c` and `error ?c` represent the energy consumption and the effect on accuracy for a component `?c`. In addition `time_unit` fluent is introduced to specify a predetermined duration of an action that should occur to bring about a change in accuracy and energy consumption. The `total_error` and `total_energy` fluents are used in the initial state to encode information regarding the machine's current state after finishing manufacturing. In addition, timed initial literals are also used to encode the duration of the interval. Using timed initial literals restricts the makespan to the duration of the interval, overcoming some planner's inability to handle concurrency in durative actions.

[1] http://www.icaps-conference.org/index.php/Main/Competitions

```
(:durative-action normal
    :parameters (?c - component)
    :duration(= ?duration (time_unit ?c))
    :condition
    (and
      (over all (in_interval))
      (at start (not(in_use ?c)))
    )
    :effect
    (and
      (at start (in_use ?c))
      (at end (not(in_use ?c)))
      (at end (increase(total_error)
        (*(time_unit ?c)
        (*(error_normal ?c)
        (/(energy_normal ?c)60)))))
      (at end (increase(total_energy)
        (*(time_unit ?c)
        (/(energy_normal ?c)60))))
      )
    )
)
```

Figure 6: Durative action representing the fact that the machine component `?c` is planned to remain in a normal state of operation for a time unit.

The goal state makes use of four optional numeric conditions to impose a tolerance window on the total error and energy consumption. The tolerance window for the total error creates the possibility to specify the manufacturing requirements of the next job and to ensure the interval activity correctly prepares the machine's state. For example, using `(< (total_error) 10)` and `(< (total_energy) 4)` in the goal state would ensure that the error must be less than 10µm(at the end of the modelled interval, thus ready for the next job) and the overall energy usage must be less than 4Wm^{-1}.

Operators

In our domain there are machine component objects, and four operators representing different levels of operation: off, idle, normal and high. Each operator is similar apart from the equation to update both error and energy fluents. Figure 6 details the normal durative action where the machine error and energy usage are adjusted based on a normal mode of operation. The action will execute for the `time_unit`, while the `in_interval` predicate is true. The full PDDL source is available from the authors on request.

Plan Metric

In this paper the following three different metrics are used:

1. `(:metric minimize (total_error))`
2. `(:metric minimize (total_energy))`
3. `(:metric minimize (/(+(total_error)(total_energy))2))`

Electrical Item	**Off** Wm^{-1}, µm	**Idle** Wm^{-1}, µm	**Normal** Wm^{-1}, µm	**High** Wm^{-1}, µm	**Multiplier**
X Servo	0, 1	0.1, 0.1	0.7, 0.5	2.8, 2	1.2E-005
Y Servo	0, 1	0.1, 0.1	0.7, 0.5	2.8, 2	1.2E-005
Z Servo	0, 3	0.2, 0.3	1.0, 1.5	4.2, 6	2.4E-005
Spindle Motor	0, 6	1.3, 0.6	6.5, 3.1	26, 12	7.7E-006

Table 1: Case study data demonstrating the different energy consumption (in Wm^{-1}) and the effect on error (in µmper minute)

The first two aim to minimise the values held in the total error and total energy fluent, whereas the third metric is used to minimise the arithmetic mean of both. This creates the potential to perform multi-objective optimisation where both error and total energy consumption are minimised for a given weighting.

Experimental Analysis

In this section, a case study is provided where interval planning is performed for a single machine tool when considering different interval scenarios. The data presented in Table 1 details the energy consumption of the machine tool, as well as the relationship between energy consumption and machine error. These values have been extracted from a similar machine tool as presented in earlier sections of this paper. As interval duration is in minutes, the data presented in Table 1 has been converted into time units. These are Watts per minute (Wm^{-1}) and the positional error in micrometres per Wm^{-1}. This is calculated using a multiplier derived from dividing the deviation in micrometres per minute by Watts per minute (µmWm$^{-1}m^{-1}$). For example, to calculate the micron error resulting from 60 minute high use of the spindle motor would be 12 µm by using Equation 1 where $duration = 12$, $energy\ consumption = 26{,}000$, and $effect\ on\ error = 7.7E-006$.

The machine-specific data presented in Table 1 is now used in the creation of several PDDL problem files to simulate the following interval scenarios. First, problem definitions are created with a duration of 30, 60, and 120 minutes. Following this, three variations of each problem are created with three different requirements on machine error. These are: tight (<20µm), medium (<50µm), and large (>50µm). These requirements are synthetically generated; however, they do provide an adequate description of different energy and machine tool accuracy requirements in a manufacturing environment. Considering the combination of each of these scenarios results in the creation of 9 different problem instances. In addition, each problem instance will be solved using each of the three metrics stated in the domain modelling section, resulting in a total of 27 different PDDL problem definition files. LPG-td (Gerevini *et al.* 2006) is

Instance	Metric: Error		Metric: Energy		Metric: $\overline{Er + En}$	
	En(Wm^{-1})	Er(μm)	En(Wm^{-1})	Er(μm)	En(Wm^{-1})	Er(μm)
T-30	138	10	138	10	138	10
T-60	260	16	220	19	232	20
T-120	520	19	380	20	410	20
M-30	0	21	0	21	0	21
M-60	800	26	374	44	670	36
M-120	1351	45	984	47	1263	46
L-30	-	-	-	-	-	-
L-60	2578	52	-	-	-	-
L-120	3951	58	1641	61	2584	60

Table 2: Experimental results detailing both error and error values for each of the nine scenarios when using three different metrics. Entries marked with a dash (-) were not solved within the 5 minute cut-off time. Problem instances are in the format of a character to represent the scenario and the interval duration in minutes. The characters are: T= tight, M = medium, L = large.

used to find the best solutions (in terms of the specified metric) to the problem definitions within a 5 minute time-frame. LPG-td has been used in "anytime" configuration; it keeps increasing the quality of plan, for a given problem instance, until the available CPU-time is over.

Table 2 provides the error (Er) and energy (En) values for each problem instance and the use of the three metrics. From the Table it is evident that the majority of the problem instances were solved within the 5 minute cut-off time. After examination, it is noticeable the problem instances requiring a large error (L) are not solved within the allowed time. It is worthy reminding that, while T and M benchmarks require that the initial error value is lower than a given value, in L benchmarks the accuracy requirement is to have a value higher than a given threshold. Therefore, plans which are suitable for T and M, are not valid for the L scenario. The fact that some instances are not solvable is not because the planning problem requires more time to identify a solution, rather there is no suitable sequence of actions capable of taking the error beyond that specified in the goal state during the allocated interval duration. However, this should be seen as a useful piece of information rather than an issue: it would not be detrimental for a manufacturer to manufacture a part on a machine with a smaller error than required to satisfy the tolerance constraints of the part.

From analysing the results presented in Table 2, it can also be seen that for both the tight and medium 30 minute interval problem instances, optimising for all three metrics results in the identification of the same plan. In addition, in some instances the energy consumption is 0. Interestingly, this is because the planner is able to identify a plan where the machine is switched-off and the slow deterioration in accuracy over the 30 minute period does not take the accuracy beyond the value set in the initial state ($<$20μmand $<$50μm). In addition, it is also possible to identify that optimising for a single metric is often at the expense of the other. For example, in T-60 it can be seen how the error is reduced to 16μmby using 260$Wm^{-1}$when optimising for error, whereas when optimising for energy, the error increases to 19μmand the energy usage decreases to 220Wm^{-1}. Optimising for the arithmetic mean of both metrics also results in a plan where both metrics are at their lowest.

This experimental analysis has demonstrated that the technique is useful for stabilising the machine error whilst reducing estimated energy consumption. It has also demonstrated that planning for large machine error is unnecessary as it will result in high energy consumption over short durations to generate large amounts of heat and cause structural deformation to reduce accuracy. Conversely, when planning for tight tolerances, the plan will contain actions with low energy consumption which will result in gradual heat generation, leading to thermal stabilisation and improved accuracy. For example, in the following plan excerpt it can be seen how, over a 30 minute period, the spindle is initially turned off and then switched back on and left to idle to maintain accuracy.

```
0:  (OFF SPINDLE_MOTOR) [10.0000]
10:  (IDLE SPINDLE_MOTOR) [10.0000]
20:  (IDLE SPINDLE_MOTOR) [10.0000]
```

Conclusion

This paper presents the exploitative use of automated planning to plan for machine tool activity between manufacturing operations. This is a novel application with potential to aid machine tool operators prepare their machine for the next manufacturing task considering overall energy consumption and the effect on manufacturing accuracy. Research has previously been undertaken in the area of reducing energy consumption and improving the error of machine tools during manufacturing. However, to the best of the authors' knowledge, little work has considered the intervals between manufacturing.

The paper provides a discussion detailing that subsystems of a machine tool (in particular servo motors) consume a large quantity of energy and that this energy results in the generation of heat. It was then discussed how this heat transfers through the machine tool's structure causing thermal deformation, resulting in positioning error of the ma-

chine tool's cutting point relative to the workpiece. This error then transfers to the manufactured component and can result in the production of out-of-tolerance parts. It was discussed how changing the machine's activity during non-manufacturing intervals can help to stabilise the machine tool's structure, resulting in a reduction in error. An experimental PDDL domain model was then developed to enable planning for manufacturing intervals using available energy and error information.

Experimental analysis was then performed using the developed domain model and nine different problem instances. Each instance relates to a different combination of accuracy requirements and interval duration. The exploratory experimental analysis demonstrates good potential as accuracy, energy, and the arthritic mean of both are minimised below the specified accuracy limit. It has also been identified that automated planning is capable of providing a viable mechanism for aiding manufacturing, albeit with a simplified domain model. The developed domain requires further development and testing to make it more accurately represent interval planning. One limitation of the presented domain is that the relationship between temperature and accuracy is non-linear, whereas for this initial proof-of-concept the relationship has been discretized into linear rates-of-change. Future work will include investigating alternative approaches to solve problems of a larger size to gain better results. For example, the use of PDDL+ and mixed integer programming will be considered, as well as undertaking further domain modelling work with the view of experimenting with different planning algorithms.

References

Amanda Jane Coles and Andrew Ian Coles. PDDL+ planning with events and linear processes. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling, ICAPS*, 2014.

Giuseppe Della Penna, Daniele Magazzeni, Fabio Mercorio, and Benedetto Intrigila. UPMurphi: a tool for universal planning on PDDL+ problems. In *Nineteenth International Conference on Automated Planning and Scheduling*, 2009.

Nancy Diaz, Moneer Helu, Stephen Jayanathan, Yifen Chen, Arpad Horvath, and David Dornfeld. Environmental analysis of milling machine tool use in various manufacturing environments. In *Sustainable systems and technology (ISSST), 2010 IEEE international symposium on*, pages 1–6. IEEE, 2010.

Nancy Diaz, Elena Redelsheimer, and David Dornfeld. Energy consumption characterization and reduction strategies for milling machine tool use. In *Glocalized Solutions for Sustainability in Manufacturing*, pages 263–267. Springer, 2011.

F Draganescu, M Gheorghe, and CV Doicin. Models of machine tool efficiency and specific consumed energy. *Journal of Materials Processing Technology*, 141(1):9–15, 2003.

Maria Fox and Derek Long. PDDL2.1: an extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20:61–124, 2003.

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

Alfonso Gerevini, Alessandro Saetti, and Ivan Serina. An approach to temporal planning and scheduling in domains with predictable exogenous events. *Journal of Artificial Intelligence Research*, 25:187–231, 2006.

Heidenhain. Aspects of energy efficiency in machine tools. Technical report, 2010.

Jürgen Hesselbach and C Herrmann. Globalized solutions for sustainability in manufacturing. In *Proceedings of the 18th CIRP International Conference on Life Cycle Engineering*, pages 2071–1050, 2011.

Ying Liu, Haibo Dong, Niels Lohse, Sanja Petrovic, and Nabil Gindy. An investigation into minimising total energy consumption and total weighted tardiness in job shops. *Journal of Cleaner Production*, 65:87 – 96, 2014.

Ying Liu, Haibo Dong, Niels Lohse, and Sanja Petrovic. Reducing environmental impact of production during a rolling blackout policy–a multi-objective schedule optimisation approach. *Journal of Cleaner Production*, 102:418–427, 2015.

Naeem S Mian, Simon Fletcher, Andrew P Longstaff, and Alan Myers. Efficient thermal error prediction in a machine tool using finite element analysis. *Measurement Science and Technology*, 22(8):085107, 2011.

Naeem S Mian, Simon Fletcher, Andrew P Longstaff, and Alan Myers. Efficient estimation by FEA of machine tool distortion due to environmental temperature perturbations. *Precision engineering*, 37(2):372–379, 2013.

Simon Parkinson, Andrew Longstaff, Andrew Crampton, and Peter Gregory. The application of automated planning to machine tool calibration. In *Proceedings of the twenty-second international conference on automated planning and scheduling (ICAPS)*, 2012.

Simon Parkinson, Andrew P Longstaff, Simon Fletcher, Andrew Crampton, and Peter Gregory. Automatic planning for machine tool calibration: A case study. *Expert Systems with Applications*, 39(13):11367–11377, 2012.

Simon Parkinson, Andrew P Longstaff, Andrew Crampton, and Peter Gregory. Automated planning for multi-objective machine tool calibration: Optimising makespan and measurement uncertainty. *Proceedings of the twenty-fourth international conference on automated planning and scheduling (ICAPS)*, pages 421–429, 2014.

Simon Parkinson, Andrew P Longstaff, and Simon Fletcher. Automated planning to minimise uncertainty of machine tool calibration. *Engineering Applications of Artificial Intelligence*, 30:63–72, 2014.

S. Edelkamp and J. Hoffmann. PDDL2.2: The Language for the Classical Part of the 4th International Planning Competition. Technical Report 195, Albert-Ludwigs-Universitat Freiburg, Institut fur Informatik, 2004.

Athulan Vijayaraghavan and David Dornfeld. Automated energy monitoring of machine tools. *CIRP Annals-Manufacturing Technology*, 59(1):21–24, 2010.

Managing Spacecraft Memory Buffers with Overlapping Store and Dump Operations

Gregg Rabideau[1], Steve Chien[1], Federico Nespoli[2], Marc Costa[3]
[1]Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, USA
{gregg.rabideau, steve.chien}@jpl.nasa.gov
[2]European Space Agency, Noordwijk, Netherlands / Telespazio VEGA UK Ltd, Luton, UK.
fnespoli@esa.int
[3]European Space Astronomy Center (ESAC-ESA), Villanueva de la Cañada, Madrid, 28692, Spain
marc.costa@esa.int

Abstract

Space mission planning/scheduling is determining the set of spacecraft activities to meet mission objectives while respecting mission constraints. One important type of mission constraint is data management. As the spacecraft acquires data via its scientific instruments, it must store the data onboard until it is able to downlink it to ground communications stations. Because onboard storage and communication opportunities are often limited, this can be a challenging task.

This paper describes a formulation of the overlapping Memory Dumping Problem (oMDP), which is a generalization of the Mars Express Memory Dumping Problem (MEX-MDP). We first describe the abstract problem of onboard data management for spacecraft. Then we focus on a more specific version that allows data downlink to be controlled by using either the priority or the maximum dump duration of each buffer.

Previous solutions to the MDP, including Max Flow and Linear Programming (LP) formulations, assume that data generation and downlink events do not overlap. We present a solution, called DALLOC, that uses a fast heuristic-based method to solve the more general oMDP. We then compare it to Max Flow as well as other heuristic methods using actual mission data from the European Space Agency's Rosetta mission. The ESA science operations team has been successfully using DALLOC to solve the oMDP in both strategic and tactical science planning.

1 Introduction

Spacecraft enable us to explore Earth, our solar system, and bodies beyond our galaxy to the furthest reaches of the universe. However, determining operations of these spacecraft (e.g. Mission planning and scheduling) is an extremely challenging part of these space missions. While in the space community it is termed mission planning, from an Artificial Intelligence perspective the issue is more scheduling than planning as the challenge is to find appropriate times to schedule observations to achieve mission objectives that conform to the operations constraints of the spacecraft. Space mission planning represents a fertile applications area for Artificial Intelligence-based planning and scheduling techniques with a wide range of deployed systems (for a survey see [Chien et al. 2012]).

One particular challenge for space mission planning is downlink planning. In this problem the data acquired onboard from engineering telemetry and science observations is stored onboard. This onboard storage is limited and is often pre-partitioned in an inflexible allocation. Commonly, first a schedule is negotiated between the space mission and a ground communications station provider (or providers). Once this schedule has been determined, a prior version of a mission plan is adapted to ensure that all data is preserved - determining exactly which portions of onboard storage are downlinked when so as to enable the science and engineering data to be acquired and downlinked without loss of data.

Many variants of this downlink problem exist. For example, there may be some uncertainty in the volume of acquired data, or deadlines for downlinking certain types of data, or buffers with dynamic priorities. We describe a particularly challenging downlink problem, the oMDP, in which data generation may occur over extremely long periods of time, overlapping with long downlink periods. We then describe the heuristic solution used by DALLOC, and compare it to two alternative heuristics and a Max Flow solution.

2 The Overlapping Memory Dumping Problem (oMDP)

Downlink scheduling is a sub-problem of the larger task of scheduling spacecraft activities. From a science planning/scheduling perspective, when constructing the schedule for the first time, the scheduler must decide on which observations to include, where they should occur, as

well as which downlink commands to issue to best satisfy science requests (e.g. for Rosetta [Chien et al. 2015]). When constructing and evaluating these observation schedules, the impact of the observations on spacecraft resources, such as memory, must be managed.

In this paper, we assume that a set of observations has been selected, and we focus on finding the best way to downlink data, thus freeing up memory used to store those observations. We focus on the scheduling of downlink commands only, assuming that the observation schedule cannot be changed. Fill rates from observations, and dump rates from downlinks, are all provided as inputs to the scheduler. As mentioned, this is a sub-problem of the strategic mission planning/scheduling process [Costa et al. 2016] where observation scheduling and downlink scheduling are performed either simultaneously or interleaved [Ayucar et al 2016]. In addition, this type of downlink re-scheduling is often necessary during short-term, tactical planning when certain last-minute changes must be made (e.g. due to the loss of a downlink).

Our problem was first discussed in [Rabideau et al. 2015] and is similar to the Mars Express Memory Dumping Problem (MEX-MDP) described in [Oddi and Policella 2004]. While we discuss this problem in the context of the Rosetta mission [Rosetta 2015], most space missions handle downlink/data volume scheduling similarly.

The general problem we solve is to specify an "empty" function that utilizes a fixed set of downlink periods to keep a set of onboard memory buffers well within their pre-defined limits. We formalize the data downlink problem as follows:

Given:

a time range T

a set of buffers $B = \{b_1, b_2, ... b_n\}$
where each b_j has
an initial volume state: $init\_vol_j$
a final volume requirement: $end\_vol\_req_j$
a hard volume capacity: $capacity_j$
a required margin: $margin_j$

a set of buffer fillers $F = \{f_1, f_2, ... f_n\}$
where each $f_i = <start\_f_i, end\_f_i, rate\_f_i>$

a set of downlinks $D = \{d_1, d_2, ... d_n\}$
where each $d_i = <start\_d_i, end\_d_i, rate\_d_i>$

Solve:

$\forall d_i, \forall b_j$: assign a function $empty(b_j,t) \rightarrow rate$ s.t.:

$\forall t \in [start\_d_i, end\_d_i]: \Sigma\ empty(b_j,t) \leq rate\_d_i$
(cannot empty more than the downlink capacity)

$\forall t \in T: volume(b_j, t) \leq capacity_j - margin_j$
(cannot exceed the buffer capacity minus margin)

$t = max(T): volume(b_j, t) \leq end\_vol\_req_j$
(cannot exceed the end volume requirement)

$\forall b_j: minimize\ max(peak\_percent(b_j))$
(maximize robustness)

In reality, as we will see, flight software on actual missions is not designed to allow for arbitrary downlink policies, so that our ability to control the $empty(b_j,t)$ function is not as flexible as desired.

Note that in our problem formulation, the downlinks and fill function are specified over all time. Therefore, the data generation and downlink events can occur concurrently. Indeed, in Rosetta operations, downlinks cover greater than half of all time and on average seven data generation events are occurring at any point in time. Thus Rosetta represents a case where prior problem formulation assumptions of non-overlap between data production and downlink [Cesta et al. 2007, Righini and Tresoldi 2010] most definitely do not hold.

3 Controlling Data Downlink with Priority and/or Duration

As with most spacecraft, Rosetta onboard data storage is partitioned into a set of buffers, called packet stores, for different types of science and engineering data that is accumulated from observations. Each instrument has a designated buffer with a specified hard upper volume limit that cannot be changed during routine scheduling.

The behavior of each downlink can be controlled in two ways: by setting priority or by limiting duration.

- First, a priority can be assigned to each of the memory buffers, indicating a relative downlink order.
- Second, downlink of a specific buffer can be halted at any time, effectively limiting the duration of data dump from that buffer.

Priority and duration are the only decision variables available to the scheduler for controlling the "empty" function described earlier. Therefore, in this formulation, the control variables are:

priorities $P = \{p_{1,1}, p_{1,2}, ... p_{i,j}\}$ for each $d_i \in D$ and $b_j \in B$
durations $U = \{u_{1,1}, u_{1,2}, ... u_{i,j}\}$ for each $d_i \in D$ and $b_j \in B$

To fully understand how these variables affect the "empty" function, we must examine the onboard software that controls the data downlink. We summarize the behavior of the Rosetta downlink software in the following set of rules.

- Some of the buffers (used for high-priority engineering data) have fixed priorities and cannot be halted (they must dump first, and until they are empty).
- A buffer remains "active" until a command is issued to stop it, after which no data will be downlinked regardless of priority.
- When more than one active buffer has data waiting to be downlinked, the one with higher priority will be dumped first.

- If more than one active buffer all have the same priority, data will be downlinked round-robin.
- When a buffer becomes empty, downlink for that buffer will stop, allowing downlink to start on the next highest priority buffer.
- Downlink from a buffer will be preempted when new data is added to an active, higher-priority buffer.

Using these downlink rules, and the two control variables, the primary goal of the downlink scheduler is to prevent overflow on all buffers. The secondary goal of the scheduler is to make selections that respect a minimum margin and maximum carryover. And finally, it is preferred to have margins as large as possible, making the schedules more robust to uncertainties in data collection (e.g. compression ratios) and downlink availability.

To achieve these goals, the scheduler must first model the behavior of the buffers so that volume and overflows can be accurately predicted. This is accomplished using the activities and timelines of the ASPEN scheduling system. With a model of how data is collected and downlinked, ASPEN generates a profile for each buffer that predicts the data volume at any point during the planning period. This profile can be used not only to predict overflows, but also provides information to the scheduler about when, and by how much, data will overflow. This information can then be used to make decisions about which priority values to assign at the start of each downlink, and when to stop the dump during each downlink. For example, after a given downlink, if there is one particular buffer that will overflow sooner, or exceed its limit by more than any other buffer, then that buffer should be given higher priority or more time to downlink.

Because of the serial nature of the resulting dump schedules, using stop dump commands to allocate fixed downlink volumes is brittle to changes in those downlinks. If downlink times change (to start later, end earlier, or with an interruption in the middle), stop dump commands that fall during deleted downlink periods will be ignored. Losing these commands will cause some buffers to dump much longer than needed, consuming time needed by other buffers.

Originally the Rosetta mission used a fixed set of pre-assigned buffer priorities and selected only the duration for each dump. To address the brittleness of this approach, the Rosetta mission switched to a priority-based method, which assigns different priorities to buffers and does not explicitly halt data dumps. This method offers less control over the exact amount to downlink from each buffer, but is more robust to changes in the downlink schedule. Potentially, both priority *and* duration could be used to control the dump schedule and increase control and robustness - this topic is left for future work.

In this paper, we discuss a set of value selection heuristics for the overlapping Memory Dumping Problem (oMDP), and evaluate their performance on selecting either dump priorities or dump durations.

4 Downlink Parameter Value Selection Heuristics

In Rosetta operations, the DALLOC software tool is used to assign buffer priorities or durations based on the number of downlinks that exist before the first overflow of that buffer. Roughly speaking, this "downlink count" heuristic used by DALLOC will assign higher priorities or longer durations to buffers with earlier overflows. This ensures that more downlink time is given to the buffers with more urgent need.

For comparison purposes, we have implemented three alternative heuristics/methods for selecting either downlink duration or priority within the DALLOC framework. In all, we have:

1. Downlink count
2. Random
3. Percent full
4. Max flow

In the "random" heuristic, values were independently selected at random to create a lower bound for comparison. When assigning priorities, one of the available priority levels is randomly selected. When assigning durations, a set of random numbers for the buffers is normalized across the total available downlink duration.

The "percent full" heuristic assigns priorities (or durations) by normalizing the peak volume percentages across the available priority values (or across the downlink duration). In other words, the relative priority or duration assigned is proportional to the relative percent full for the peak of that buffer.

Finally, we compare our local heuristics against "max flow" which uses flow values that result from running the Edmond-Karp max-flow algorithm on the network constructed for the overlapping Memory Dumping Problem (oMDP). The MEX-MDP very closely matches the oMDP, allowing us to use a similarly constructed network. Here, "flow" represents data flowing into the buffers, out via downlinks, and carrying over to the next downlink (or end of the planning period). When the max-flow algorithm completes successfully, dump durations for each buffer can be extracted from the resulting graph. If selecting priorities, the "flow" value is converted to a priority by looking at it as a percent of the downlink available.

One distinction with Rosetta and the oMDP, however, is that buffer store and dump activities can occur over long periods of time (e.g. hours) and often overlap. In the max-flow formulation, these activities must be modeled as instantaneous events. The resulting flow values, therefore, are not guaranteed to prevent overflow when the buffer profile is created.

In "max flow", the entire schedule is evaluated to compute control variables (dump durations or priorities) for all downlinks at once. This can help ensure that selected values for one downlink do not adversely impact what can be done in a later downlink (i.e. prevents "painting into a corner"). However, the run-time for such a global evaluation can be significantly longer than local methods. All other heuristics use more local methods, selecting parameters for a downlink

without much consideration for other downlinks. However, because we update buffer volumes after scheduling a downlink, this new information can be used when applying the heuristic to subsequent downlinks. While this lack of global information may lead to sub-optimal heuristics, it makes the computation very fast. Efficiency is important when downlink scheduling must be performed repeatedly during the construction of observation schedules, as done in the strategic planning phase.

All heuristics are compared in the empirical evaluation section of this paper.

5 Schedule Robustness and Iterative Leveling

As in [Oddi and Policella 2004], we are interested in producing schedules that are robust to unpredictable events that occur after committing to the schedule (e.g. after uplink). For comparison purposes, we use the same approximation to schedule robustness, which uses the maximum percent full that any buffer is predicted to be at any time.

In [Oddi and Policella 2004], max-flow is used to find a solution for *all* downlinks. This does not maximize the flow through any *individual* downlink, which can produce solutions that contain buffers that are near capacity at specific times. These solutions are considered brittle, and an "Iterative Leveling" technique is presented to improve robustness. Here, more robust solutions are generated by assigning an epsilon smaller capacity to the brittle buffer after each iteration.

We present a variant of iterative leveling that reduces all capacities to the same level instead of one-at-a-time, and iterates using binary search instead of epsilon reduction. First, we recognize that limiting one buffer to a lower percent does not help robustness if other buffers are allowed to increase above the previously identified maximum. For example, if one buffer is limited to 90%, all should be limited to 90%. Therefore, binary search can find a more robust solution by using an artificial capacity that is reduced when a solution is found, or increased when the solution results in overflows.

6 Estimated Computational Complexity

Figure 1 contains high-level pseudo-code for downlink scheduling using a global max-flow formulation, using local heuristics, and finally the outer loop that adds iterative leveling with binary search. We use the following variables to analyze the computational complexity of these downlink scheduling methods:

D = downlinks
B = buffers
C = capacities
F = fill rate changes

First, we compute max-flow values using the Edmond-Karp implementation, which is $O(EV^2)$ where E is the number of edges and V is the number of nodes. In the MDP flow network, there are only a few nodes and edges for each buffer dump [Oddi and Policella 2004]. Therefore, E and V are each approximately equal to D*B, making the overall complexity of solving the MDP with max-flow $O(D^3B^3)$.

```
scheduleWithMaxFlow(D, B, C, F)
  M = computeMaxFlow(D, B, C, F)
  for each d in D
    for each b in B
      assignValue(d, b, M[d][b])

scheduleWithHeuristic(D, B, C, F)
  sort(D)
  for each d in D
    for each b in B
      c = C[d][b]
      heuristicallyAssignValue(d, b, c)
    for each f in F
      recalculateVolumeAt(f)

iterativeLeveling(D, B, C, F)
  Cdelta = 100
  Cprev = 0
  while(Cdelta > 1)
    Cdelta = abs((C – Cprev) / 2)
    Cprev = C
    if(USE_MAX_FLOW)
      r = scheduleWithMaxFlow(D, B, C, F)
    else
      r = scheduleWithHeuristic(D, B, C, F)
    if(r)
      C -= Cdelta
    else
      C += Cdelta
```

Figure 1: Scheduling with max-flow, with local heuristics, and iterative leveling

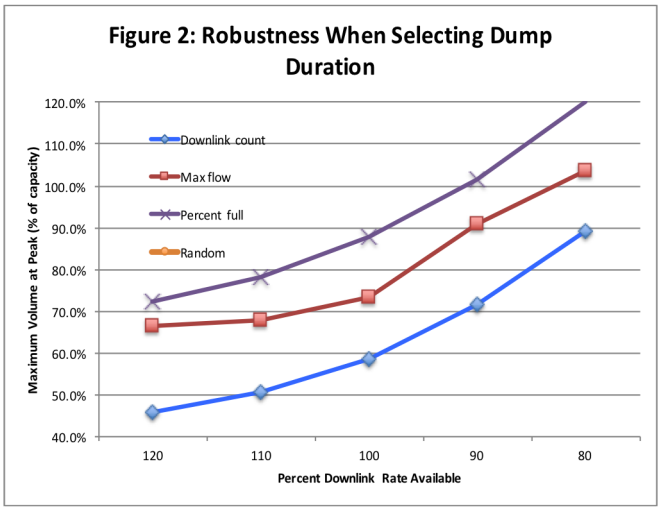
For all but "random", the local heuristics require an initial sort of the downlinks, and a propagation of volumes after each downlink assignment. Sorting the downlinks is O(DlgD). Because the downlinks are scheduled forward in time, recalculating volumes after the last scheduled downlink is simply proportional to the number of fill rate changes. The resulting complexity of scheduling using a local heuristic is O(DlgD + D*F).

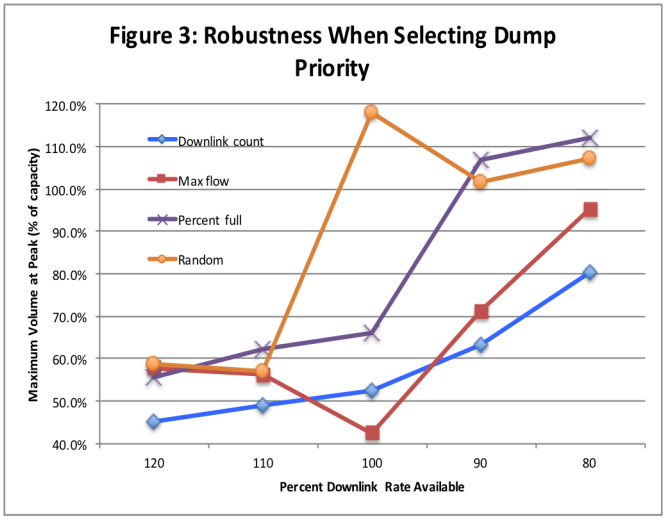
For any of the heuristics, performing iterative leveling with binary search will add a constant multiplier. This is because our implementation works on an integer percentage between 1 and 100, which will loop at most lg100 times (about 7 times).

For the Rosetta mission, downlink planning is typically processed over a "Medium Term Plan" or MTP, which is generally 4 weeks in length. For Rosetta there are 16 buffers, and for one MTP, there are typically 30+ downlinks and hundreds of fill rate changes.

7 Empirical Evaluation

We have conducted an empirical evaluation of the scheduling algorithm using the four previously mentioned heuristics for assigning dump durations or priorities. Performance of the





heuristics based on run-time and schedule robustness. We use data from four medium-term planning (MTP) periods during the comet escort phase of the Rosetta mission. The data collected during each MTP is roughly the same as the downlink available. This is because the fill rate data we use was taken from an archive of the tactical planning process, where the strategic selection of observations has already completed. In addition, the data collected from these observations is typically about 2x to 3x the total capacity of all buffers.

First, in order to evaluate the performance over a range of constrainedness, we varied the downlink rates from 80% to 120% of the true rate used in operations. Then, we look at how each heuristic compares when selecting either dump duration (Figure 2) or dump priority (Figure 3).

When selecting dump duration, the relative performance of the heuristics does not change as the problem becomes more or less constrained. As expected, selecting random durations results in schedules that are the most brittle (resulting volumes were greater than 200% and therefore do not appear on the graph in Figure 2). Surprisingly, using the max-flow

	No Leveling	Leveling
Downlink count	14.6	92.6
Max flow	337.5	2190.0
Percent full	11.0	77.3
Random	11.1	76.8

Table 1: Average run-times (seconds)

	No Leveling	Leveling
Downlink count	64.2%	55.6%
Max flow	85.4%	57.9%
Percent full	76.4%	77.1%
Random	162.9%	175.9%

Table 2: Average robustness for the actual (100%) downlink rate

values does not produce the most robust solutions. The "downlink count" heuristic, which selects duration based on the number of downlinks before the first overflow, consistency outperforms max-flow.

Next we look at the performance of heuristically assigning buffer priorities without changing dump durations (Figure 3). Recall that this will simply control the order in which buffers are downlinked. Because max-flow solutions contain volume assignments, we first convert flow values to priorities based on the flow volume as a percent of capacity. Using the actual downlink rate (100%), when fill and downlink volumes are about the same, max-flow outperforms all other methods by at least 10%. But when over- (<100%) or under-constrained (>100%), we see that the "downlink count" heuristic outperforms max-flow by about 10%.

Results from max-flow were surprising since we had expected a global solution to consistently outperform any of the local heuristics. First, we must consider that max-flow was not designed to assign priorities, which could explain the results in Figure 3. For under-constrained problems, one possibility is that max-flow is relying more heavily on iterative leveling to keep the peaks low. For over-constrained problems, max-flow is more likely to fail to find a solution, which could contribute to the drop in robustness. In all cases, max-flow models the filling and downlinking events as instantaneous, while in reality, these activities have durations and even overlap. This modeling inaccuracy may be producing suboptimal solutions.

In addition to comparing the robustness of these various methods, we are also interested in run-times in order to determine whether the benefits outweigh the costs. Table 1 reports the run times (real CPU time in seconds) of max-flow and each of the local heuristics, with and without iterative leveling. Table 2 reports the robustness (max peak volume percent) obtained, averaged from both priority and duration assignment, using actual (100%) downlink rates. As

expected, compared to using local heuristics, the cost of running max-flow is high (roughly 20x slower). Moreover, solutions are not much better, and in some cases worse than those produced by the best local heuristics. Next, we consider the cost-benefit tradeoff of using iterative leveling. As shown in the complexity analysis, the cost is a multiplier of lg100, or about 6.6x. This is consistent with the observed data in Table 1. In Table 2, we see that iterative leveling, as expected, is more important for max-flow where no peak minimization is attempted in a single iteration.

In our experience, the best combination is the use of iterative leveling with a local heuristic. This allows the tool to be more responsive during tactical planning, but also enables its use as part of the strategic planning process, where it may need to be run hundreds of times to generate a single MTP schedule.

8 Discussion

The downlink scheduling algorithms described in this paper were originally deployed as part of the Rosetta early science planning operations tool that also scheduled science activities [Chien et al. 2015]. Later in operations the data downlink scheduling software was modularized and extracted to also be used in mid to late tactical science planning. It is this separated scheduling software and associated algorithms that we describe in this paper. In some form or other this downlink scheduling software has been in use to schedule over 18 Medium Term Plans (each ~1 month of Rosetta Orbiter operations).

Automated downlink planning is in operational use for the Mars Express mission [Cesta et al. 2007]. However, they model observations and downlinks as non-overlapping (or equivalently instantaneous) data producers and consumers. In many space missions, including Rosetta, this assumption does not hold. Their robustness metric is similar to our margin requirement.

Onboard downlink management [Pralet et al. 2014] is proposed in order to address challenges of uncertainty in data generation (due to the uncertainty of effectiveness of content-dependent compression schemes). This formulation of the problem adds even several more complexities such as antenna pointing, multiple channels, data latency, and encoding table time. Again for a typical earth imager, the data production is effectively instantaneous, in contrast to the Rosetta problem.

Most other deployed automated planners must also solve some version of the downlink planning/scheduling problem however in most cases it is not the focus of the overall scheduling problem (e.g. Hubble Space Telescope [Johnston and Miller 1994], Earth Observing One [Chien et al. 2005, 2010] or Orbital Express [Knight et al. 2013]).

The Philae Lander for the Rosetta Mission has a science scheduling with downlink problem [Simonin et al. 2012]. They use ILOG-scheduler in a system called MOST to solve for most of the scheduling constraints except data management. They examine the problem of scheduling science experiments with fixed science experiment storage and downlink buffer storage but with a fixed priority downlink strategy. This problem is analogous to the full Rosetta scheduling problems [Chien et al. 2015]. However, one key difference is that MOST does not have the ability to re-program buffer priorities dynamically as we have on the Rosetta Orbiter (and described here in this paper).

9 Summary

We have described the downlink scheduling problem, a well defined subproblem within the overall space mission planning and scheduling problem. While this problem can be and often is solved in isolation, it is also addressable concurrently with the overall scheduling problem.

We then described the Rosetta downlink scheduling problem as a specific instantiation of the general downlink scheduling problem with overlapping data creation and downlinks. We describe heuristic solutions to this new problem that are in operational use for ESA's Rosetta mission and show that they outperform prior max-flow methods that do not handle overlapping effects on Rosetta mission data.

Acknowledgments

Portions of this work were performed at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

References

[Cesta et al., 2007] A. Cesta, G. Cortellessa, S. Fratini, A. Oddi, and N. Policella. "An Innovative Product for Space Mission Planning: An A Posteriori Evaluation." Proc Intl Conf on Automated Planning and Scheduling, pp. 57-64. 2007.

[Chien et al., 2005] S. Chien, R. Sherwood, D. Tran, B. Cichy, G. Rabideau, R. Castano, A. Davies, D. Mandl, S. Frye, B. Trout, S. Shulman, D. Boyer, "Using Autonomy Flight Software to Improve Science Return on Earth Observing One, Journal of Aerospace Computing, Information, & Communication, April 2005, AIAA.

[Chien et al., 2010] S. Chien, D. Tran, G. Rabideau, S. Schaffer, D. Mandl, S Frye, "Timeline-based Space Operations Scheduling with External Constraints," International Conference on Automated Planning and Scheduling, Toronto, Canada, May 2010.

[Chien et al., 2012] S. Chien, M. Johnston, N. Policella, J. Frank, C. Lenzen, M. Giuliano, A. Kavelaars, A generalized timeline representation, services, and interface for automating space mission operations, Space Operations (SpaceOps 2012). Stockholm, Sweden. June 2012.

[Chien et al., 2015] S. Chien, G. Rabideau, D. Tran, J. Doubleday, D. Chao, F. Nespoli, M. P. Ayucar, M. Costa, C. Vallat, B. Geiger, N. Altobelli, M. Fernandez, F. Vallejo, R. Andres, M. Kueppers, Using Constraint-based Search to Schedule Science Campaigns for Rosetta

Orbiter, Invited Talk, Proc. International Joint Conference on Artificial Intelligence, Buenos Aires, Argentina, July 2015.

[Costa et al., 2016] M. Costa, M. Perez-Ayucar, M. Almeida, M. Ashman, R. Hoofs, S. Chien, J. Beteta, M. Kueppers, “Rosetta: Rapid Science Operations for a Dynamic Comet, Space Operations Symposium, Daejon, Korea, 2016.

[Johnston and Miller, 1994] M. D. Johnston and G. Miller. "Spike: Intelligent scheduling of hubble space telescope observations." Intelligent Scheduling (1994): 391-422.

[Knight et al., 2014] R. Knight, C. Chouinard, G. Jones, D. Tran, Leveraging Multiple Artificial Intelligence Techniques to Improve the Responsiveness in Operations Planning: ASPEN for Orbital Express, AI Magazine, Vol 35, No 4, 2014.

[Oddi and Policella, 2004] A. Oddi and N. Policella, A Max-Flow Approach for Improving Robustness in a Spacecraft Downlink Schedule, International Workshop on Planning and Scheduling for Space (IWPSS-04). Darmstadt, Germany. June 2004.

[Perez-Ayucar et al., 2016] M. Perez-Ayucar, M. Almeida, M. Ashman, S. Chien, M. Costa, J. Garcia, R. Hoofs, M. Kueppers, D. Merritt, J. Marin, F. Nespoli, G. Rabideau, E. Sanchez, “Science Data Volume management for the Rosetta Spacecraft, Space Operations Symposium, Daejon, Korea, 2016.

[Pralet et al., 2014] C. Pralet, G. Verfaillie, A. Maillard, E. Hébrard, N. Jozefowiez, M.-J. Huguet, T. Desmousceaux, P. Blanc-Paques, J. Jaubert. Satellite Data Download Management with Uncertainty about the Generated Volumes. Proc. of the 24th International Conference on Automated Planning and Scheduling (ICAPS-14), Portsmouth, NH, USA, 2014.

[Rabideau et al., 2015] G. Rabideau, F. Nespoli, S. Chien. Heuristic Scheduling of Space Mission Downlinks: A Case study from the Rosetta Mission. International Workshop on Planning and Scheduling for Space (IWPSS-15). Buenos Aires, Argentina, 2015.

[Righini and Tresoldi, 2010] Righini G, Tresoldi E. A mathematical programming solution to the Mars Express memory dumping problem. Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on. 2010 May;40(3):268-77.

[Rosetta Mission, 2015] http://rosetta.esa.int/, retrieved 16 November 2015.

[Simonin et al., 2012] C. Simonin, C. Artigues, E. Hebrard, and P. Lopez, "Scheduling Scientific Experiments on the Rosetta/Philae Mission," "Scheduling scientific experiments on the Rosetta/Philae mission." In *Principles and Practice of Constraint Programming*, pp. 23-37. Springer Berlin Heidelberg, 2012.

Efficient High Quality Plan Exploration for Network Security

Anton V. Riabov Shirin Sohrabi Octavian Udrea Oktie Hassanzadeh
IBM T.J. Watson Research Center
1101 Kitchawan Rd, Yorktown Heights, NY 10598, USA
{riabov, ssohrab, udrea, hassanzadeh}@us.ibm.com

Abstract

We consider the application of planning in network security. In this application, plans represent possible attacks on the network, and network administrators need tools that would allow them to explore the plan space quickly and efficiently. Multiple aspects of this problem require generating and inspecting more than one plan, primarily due to limited information about the possible actions of the attacker, and a variety of possible attacks. This problem can be modeled as diverse planning, with the caveat that high quality (or, equivalently, low cost) plans must be prioritized, since those plans typically represent the most efficient attacks that are of highest importance to the administrators. Hence, there is a need for a systematic approach to finding such plans. We propose a new technique based on a top-k planner that finds k optimal or near-optimal plans, followed by plan consolidation, for generating diverse high quality plans. Comparing to existing diverse planners, we show that it is able to meet the high quality and plan diversity requirements efficiently, and therefore we can recommend it for this application.

1 Introduction

Multiple security-related challenges arising in administration and management of computer networks have been successfully tackled using classical planning in prototypes and production systems. In this paper, we propose to address a specific requirement of finding relevant but sufficiently different attacks, which we believe to be common for all proposed planning-based systems, by developing an efficient technique that allows A^*-based planners to produce a diverse set of high quality plans.

In network security applications of planning, domain models are developed by capturing and formalizing expert knowledge. Domain models include the actions of attacker, such as network scans and exploitation of vulnerabilities to infect network hosts with malware, possibly complemented by actions representing actions of users that expose new vulnerabilities, actions that help account for network connectivity, installed software, operating system and other configuration [Boddy *et al.*, 2005; Roberts *et al.*, 2011; Lucngeli *et al.*, 2010]. The individual valid plans then represent possible attacks, and the space of valid plans represents attack graphs, therefore allowing network administrators to explore and analyze the space of possible attacks, for example, by altering goals or initial state configuration.

Another class of domain models, which uses planning-based diagnosis techniques [Sohrabi *et al.*, 2010], allows augmenting the planning problem with a sequence of ambiguous and unreliable observations from network monitoring systems to generate plans of attacks that may be in progress [Sohrabi *et al.*, 2013].

Planners can indeed be very effective in network security applications, even compared to more traditional penetration testing tools and analysis techniques based on attack graphs, because planning provides efficient search in huge attack graphs induced by domain models that are exceedingly small by comparison, and therefore are significantly easier to design and maintain [Lucngeli *et al.*, 2010].

Nevertheless, this still leaves open the question of selecting the relevant portions of attack graphs to bring to the attention of the network administrators (or automated penetration testing tools). One of the planning systems discussed above proposed generating top high quality plans to address this issue [Sohrabi *et al.*, 2013]. While finding multiple high quality plans is beneficial and necessary in order to focus the investigation on the efficient and hence relevant attacks, by itself it is not sufficient. If generated attack plans are too similar to each other in actions and states, administrators can be easily overwhelmed with small variations of the same pattern. In addition, planner performance is important for timely response to ongoing attacks.

We believe this combination of requirements has not been addressed in prior work. The problem we are considering is closely related to diverse planning, but plans found by the diverse planners are not necessarily all of high quality [Nguyen *et al.*, 2012; Sroka and Long, 2012]. On the other hand, top-k planning [Riabov *et al.*, 2014] provides an efficient technique for finding a set of top quality plans, by applying K^* algorithm [Aljazzar and Leue, 2011] to augment A^* search, but it does not guarantee plan diversity.

We propose to address these challenges by using a top-k planner to create a large set of high-quality plans, followed by clustering based on a similarity metric, and finally outputting a set of cluster representatives that are both high quality and

diverse. In the rest of the paper, we briefly describe the top-k planning and clustering techniques we use, and evaluate the end-to-end performance of our approach, comparing to existing diverse planners and a top-k planner.

2 Network Security: Domain Models

Rather than develop a new domain model for the network security application, our goal is to show via experiments how our approach can benefit previously proposed models. We generated planning task instances for two classes of domain models: attack graphs, as discussed in [Boddy *et al.*, 2005; Roberts *et al.*, 2011; Lucngeli *et al.*, 2010] and hypothesis exploration based on unreliable observations and a model of a dynamical system, as described in [Sohrabi *et al.*, 2013].

For the attack graph model, we followed an approach similar to [Lucngeli *et al.*, 2010]. We obtained a list of products and known vulnerabilities from The U.S. National Vulnerability Database, allowing us to define up to 3554 network exploit actions, each annotated with low, medium or high complexity, and configure the hosts with a selection from 21107 products. We also generated a random hierarchical network topology of up to 2000 nodes on up to 150 networks separated by firewalls, with several VPN connections cutting across the hierarchy. We defined the following actions: *connect-tcp*, which checked network connectivity between hosts, *exploit-net-V*, which represented remote exploitation of a known vulnerability *V* for a compatible product, with higher costs for more complex attacks, and *exploit-local-V*, with similarly assigned costs, which was invoked after the remote exploitation succeeded, to achieve escalation of privilege and fully compromise the host and use it for subsequent attacks. All network hosts, including gateway nodes between networks, were configured to have at least one locally exploitable product and one remotely exploitable product, which ensured a large number of possible attacks, and therefore a large space of plans for exploration.

Our hypothesis exploration domain model, following [Sohrabi *et al.*, 2013], uses two manually defined state transition systems: a malware detection system with 25 states described in that work, and an extended version of it representing two hosts and interactions between them, with 221 states total. To generate problem instances, we generated random observation sequences of varying length, and combined them with the transition systems. In this case, the planning task models the states of the host, transitions between states, and the corresponding observations from network traffic monitoring sensors, with the goal of generating hypotheses explaining the observations. For example, a malware infection state of a host can have a corresponding observation of downloading an executable file, and a possible transition to a command-and-control rendezvous state which has a new Internet Relay Chat (IRC) session observation. The executable download may also be observable in other states, such as remote software installation by administrator, which makes this observation ambiguous, also making multiple hypotheses necessary. Note that due to this structure of the model, and since the same transition system state can be reached through different explanation paths, one state of the modeled system corresponds to many possible planning states. Consequently, this model generates a very large space of plans, as in the attack graph domain.

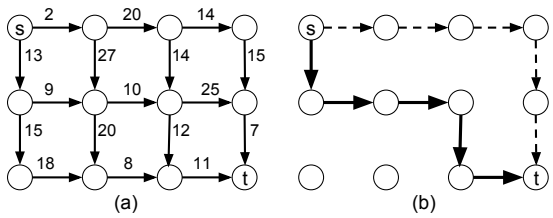


Figure 1: (a) shows a graph with source node s and terminal node t with edge lengths specified on the edges; (b) shows the shortest path in bold arrows and the second shortest path in dashed arrows.

3 Finding Diverse High Quality Plans

To measure plan quality, we assume that action costs can be assigned such that cost of a plan is the sum of costs of actions included in the plan, and the lower the cost of the plan, the higher its quality. The first step of our approach is to generate a fixed number k of lowest-cost plans, and then cluster similar plans based on a plan similarity metric. We require k to be sufficiently large, so that when similar plans are clustered, we obtain the necessary number of clusters. Finally, one lowest-cost representative plan from each cluster is selected to form the resulting set of diverse high quality plans.

In this section, we first introduce what we call a top-k planning problem. Then, we describe how to compute top-k plans using a k shortest paths algorithm. Finally we describe the clustering algorithm and the plan similarity metric used for clustering.

3.1 Tok-k Planning Problem

Definition 1 *We define the top-k planning problem as* $R = (F, A, I, \mathcal{G}, k)$*, where* F *is a finite set of fluent symbols,* A *is a set of actions with non-negative costs,* $I \subseteq F$ *defines the initial state,* $\mathcal{G} \subseteq F$ *defines the goal state, and* k *is the number of plans to find. Let* $R' = (F, A, I, \mathcal{G})$ *be the planning problem with action costs that has* n *plans (*n *can be infinite). The set of plans* $\Pi = \{\pi_1, ..., \pi_m\}$*, where* $m = k$ *if* $k \leq n$*,* $m = n$ *otherwise, is a solution to* R *if an only if each* $\pi_i \in \Pi$ *is a plan for* R' *and there does not exists a plan* π' *for* R'*,* $\pi' \notin \Pi$*, and a plan* $\pi_i \in \Pi$ *such that* $cost(\pi') < cost(\pi_i)$ *.*

The solution to the top-k planning problem is a set of low-cost plans, and are not necessary all optimal. If k is less than the number of optimal plans for R', then Π will contain all of the optimal plans. However, if k is larger than the number of optimal plans for R' then Π will contain some suboptimal plans in addition to all optimal plans. Also note that if $k > n$, Π contains all n valid plans, otherwise it contains k plans

3.2 Background: K Shortest Paths Problem

K shortest paths problem is an extension of the shortest path problem where in addition of finding one shortest path, we need to find a set of paths that represent the k shortest paths [Hoffman and Pavley, 1959].

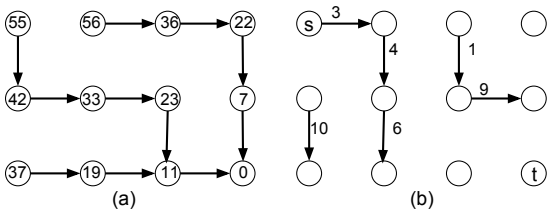


Figure 2: (a) shows the shortest path tree T and distance to destination t; (b) shows the side edges with their associated detour cost.

The following is a formal definition taken from Eppstein [Eppstein, 1998].

Definition 2 (K Shortest Path Problem) *k shortest path problem is defined as 4-tuple $Q = (G, s, t, k)$, where $G = (V, E)$ is a graph with a finite set of n nodes (or vertices) V and a finite set of m edges E, s is the source node, t is the destination node, and k is the number of shortest paths to find. Each edge $e \in E$ has a* length *(or* weight *or* cost*), which is denoted by $l(e)$. The length of a path p, $l(p)$, is consequently defined by the sum of its edge lengths. The* distance *$d(u, v)$ for any pair of nodes u and $v \in V$ is the length of the shortest path between the two nodes. Hence, $d(s, t)$ is the length of the shortest path for the problem Q. Let n = size of the set of all s-t paths in graph G. Then, the set of paths $P = \{p_1, p_2, ..., p_m\}$, $m = k$ if $k \leq n$, $m = n$ otherwise, is the solution to the k shortest paths problem Q if and only if each $p_i \in P$, is a s-t path in graph G and there does not exist a s-t path p' in graph G, $p' \notin P$ and a path $p_i \in P$ such that $l(p') < l(p_i)$.*

Note that if $k > n$, then P contains all s-t paths, otherwise P contains k shortest paths from node s to node t. It follows from the definition that at least one shortest path with length $d(s, t)$ is in the set P if $m > 0$. Figure 1 shows an example from [Eppstein, 1998] to illustrate the terminology. The distance $d(s, t) = 55$, is the length of the shortest path shown in bold; the length of the second shortest path is 58.

The K^* algorithm [Aljazzar and Leue, 2011] is an improved variant of the Eppstein's k shortest paths algorithm [Eppstein, 1998] and hence uses many of the same concepts as in the Eppstein's algorithm (which we refer to as EA). Here, we first outline the EA algorithm, and then discuss K^*.

Given a k shortest paths problem Q, the EA algorithm first computes a single-destination shortest path tree with t as the destination (or the reversed single-source shortest path tree) by applying Dijkstra's algorithm on G. The edges in the resulting *shortest path* tree, T are called the *tree* edges while all the missing edges (i.e., the edges in $G - T$) are called the *sidetrack* edges. Each edge in G is assigned a number that measure the detour cost of taking that edge. Consequently, the detour cost of the tree edges is 0, while the detour cost of the sidetrack edges is greater than 0. Figure 2 shows the shortest path tree T and the sidetrack edges along with their detour cost of our earlier example.

The EA algorithm then constructs a complex data structure called *path graph* $P(G)$ that stores the all paths in G, where each node in represents a sidetrack edge. This is followed by the use of Dijkstra search on $P(G)$ to extract the k shortest paths. An important property is that given a sequence of sidetrack edges representing a path in $P(G)$ and the shortest path tree T, it is possible to uniquely construct a s-t path in G. This can be done by using sub-paths from T to connect the endpoints of sidetrack edges. Given this property and the special structure of $P(G)$, it is ensured that the i-th shortest path in $P(G)$ results in a sidetrack sequence which can be mapped to the i-th shortest path in G. By construction, $P(G)$ provides a heap-ordered enumeration of all paths in G, and since every node of $P(G)$ has limited out-degree (at most 4), the complexity of enumerating paths in increasing cost order is bounded. The worst-case runtime complexity of the EA algorithm is $O(m+n \log n+kn)$. This complexity bound depends on a compact representation of the resulting k paths, and can be exceeded if the paths are written by enumerating edges.

Although EA could be used for top-k planning, the K^* algorithm is preferable, because it does not require the complete state transition graph G. Instead, K^* can create G dynamically using $A*$ search, driven by a heuristic toward the goal, an approach commonly used in planners (e.g., Fast-Downward [Helmert, 2006]). In short, the K^* algorithm works as follows. The first step is to apply a forward A* search to construct a portion of graph G. The second step is suspending A* search, updating $P(G)$ similarly to EA, to include nodes and sidetracks discovered by A*, applying Dijkstra to $P(G)$ to extract solution paths, and resuming the A* search. The use of A* search to dynamically expand G enables the use of heuristic search and also allows extraction of the solution paths before G is fully explored. While K^* algorithm has the same worst-case complexity as the EA algorithm, it has better performance in practice because unlike the EA algorithm, K^* does not require the graph G to be completely defined when the search starts.

3.3 Top-k Planning Using K^*

In the implementation of the planning algorithm we follow the algorithm structure imposed by K^*, as follows.

```
 0. Read planning problem R = (F, A, I, G, k).
 1. Expand the state graph G by using A*
    and applying actions to compatible  states
    starting from I, and until  G is reached.
 2. Continue applying A* to expand G
    until 20% increase in links or nodes.
 3. Update P(G) based on new links in G.
 4. Apply Dijkstra step
    to extract the next path from P(G).
 5. If k paths are found
 6.      Goto step 10.
 7. If K* scheduling condition is reached
 8.      Goto step 2.
 9. Goto step 4.
10. Return at most k plans (one plan per path).
```

We expect that with some work this approach can be integrated into planners that use A* search, enabling those planners to solve top-k problems. Also note that the soundness and completeness of planning follows directly from the soundness and completeness of the K^* algorithm.

3.4 Clustering Algorithm

Given the set of top-k plans, in this section, we will discuss how to group the similar plans using clustering techniques. In practice, many of the generated top-k plans are only slightly different from each other. That is, they do seem to be duplicates of each other, except for one or more states or actions that are different. This may be the result of the underlining AI planner which tries to generate all alternative low-cost plans, and while this generates distinct low-cost plans, it does not always mean that these plans are significantly different from each other. Hence, instead of presenting large number of plans, some of which could be very similar to each other, with the help of clustering, we can present clusters of plans, where each cluster can be replaced by its representative plan.

Clustering has been a topic of interest in several areas of research within several communities such as Information Retrieval (e.g, [Aslam *et al.*, 2004]), machine learning, and Data Management as part of the data cleaning process (e.g., [Hassanzadeh and Miller, 2009]). Many survey papers exist on clustering algorithms (e.g, [Xu and Wunsch, 2005; Filippone *et al.*, 2008]). While most, if not all, clustering algorithms share a common goal of creating clusters that minimize the intra-cluster distance (distance between members of the same clusters) and maximize the inter-cluster distance (distance between members of different clusters), the assumptions and inputs for these clustering algorithm are often different. For example, several of these approaches assume some given input parameters such as the number of clusters or a cluster diameter. To consolidate similar plans produced by the top-k planner, we apply a clustering algorithm that must satisfy the requirements stated below. One representative plan from each cluster is selected to be included in the final set of diverse plans.

Definition 3 (Clustering Requirements) *Given a set of k sorted plans, Π, create clusters of plans $\mathcal{C} = \{c_1, ..., c_o\}$ where the value of o is unknown ahead of time. Further, for each two clusters c, $c' \in \mathcal{C}$, $c \cap c' = \emptyset$ and $\forall \pi \in \Pi$, $\exists c \in \mathcal{C}$ such that $\pi \in c$. Hence, the clusters are disjoint and each plan belongs to one cluster.*

We propose the use of the following three non-hierarchical clustering algorithms. Each of these algorithms require visiting each plan only once in order to decide to which cluster they belong to; hence, are called single-pass algorithms.

Center-Link

Center-Link clustering algorithm iterates over the top-k plans starting with the lowest-cost plan. For each plan, it computes the similarity to a representative of each cluster created in previous iterations. If there are no clusters that have a representative similar to the plan (i.e., their similarity score is above the threshold θ), a new cluster is created and the plan becomes the representative of that cluster. Otherwise the plan is added to the first cluster whose cluster representative is similar to this plan. Cluster representatives are chosen to be the lowest-cost plans in each cluster. Due to the order of iteration, stating from the lowest-cost plans, the cluster representative is always the first added plan to the cluster. This algorithm is similar to the CENTER algorithm in [Hassanzadeh and Miller, 2009], however, the sorted input is different (i.e., plans, as opposed to records in a database). The Center-Link algorithm could result in small number of similarity comparisons because each plan is only compared to the representative plan of each cluster.

Single-Link

Single-Link clustering algorithm is an extension of the Center-Link algorithm, where instead of comparing only with the representative of a cluster, each plan is compared with all members of a cluster, and if the plan is found to be similar to any of the members of that cluster, then it is assigned to that cluster. Single-Link algorithm is a non-hierarchical variation of single-linkage algorithm [Xu and Wunsch, 2005]; the node joins a cluster as long as there is a single link with one of the members of the clusters. This algorithm could result in the smallest number of clusters.

Average-Link

Average-Link algorithm is a simple extension of the Single-Link algorithm, where each plan is compared with all the members of a cluster and the average similarity score is used to determine if the plan belong to that cluster or not. This algorithm results in many similarity comparisons, and could result in large number of clusters. Note, Average-Link clustering is a non-hierarchical variant of hierarchical average-linkage clustering [Xu and Wunsch, 2005]. In the experiments we show we have used this algorithm because it produced more clusters and more diverse plans.

3.5 Plan Similarity Metric

Finding if two plans are similar has been studied mainly under two categories: plan stability for replanning (e.g., [Fox *et al.*, 2006]) and finding diverse plans (e.g., [Nguyen *et al.*, 2012]). We follow prior work on diverse planning and compute plan similarity during clustering as Jaccard similarity between actions of the plan, thereby grouping similar plans together and increasing the diversity between the clusters.

Jaccard similarity is a score between 0 and 1, which measures the ratio of the number of actions that appear in both plans to the total number of actions appearing in at least one plan. Let A(π) be the set of actions in π, then:

$$\text{sim}_{\text{Jaccard}}(\pi, \pi') = \frac{|A(\pi) \cap A(\pi')|}{|A(\pi) \cup A(\pi')|} \quad (1)$$

We note that Jaccard similarity is the inverse of the plan distance defined in [Nguyen *et al.*, 2012].

4 Experimental Evaluation

To compare planner performance, we configure the planners so that approximately 50 diverse plans are generated. We measure plan diversity using stability and uniqueness metrics. We also compare plan cost and planning time.

We measure stability and uniqueness using the following formula from [Roberts *et al.*, 2014]. Note, we modified these formula to make it a number between 0 and 1. Let $\Pi = \{\pi_1, ..., \pi_m\}$ be the set of plans. If $|\Pi| = 1$, $\text{Diversity}_{\text{stability}}(\Pi) = 1$, and $\text{Diversity}_{\text{uniqueness}}(\Pi) = 1$, otherwise for $|\Pi| \geq 1$:

Domain	Top-k + clustering					LPG-d					Div				
	Time	**#Plans**	**Cost**	**D**	**U**	**Time**	**#Plans**	**Cost**	**D**	**U**	**Time**	**#Plans**	**Cost**	**D**	**U**
Malware-5	1	50	1502	0.51	1	1	10	3513	0.80	1	1	9	1789	0.36	0.37
Malware-10	1	50	1586	0.41	0.99	59	10	8426	0.84	1	1	5	3861	0.44	0.54
Malware-20	3	50	1492	0.20	0.99	384	10	16520	0.87	1	1	6	7262	0.46	0.53
Malware2-5	1	50	2005	0.66	0.96	-	-	-	-	-	1	3.2	1454	0.40	0.98
Malware2-10	2	50	2441	0.47	1	-	-	-	-	-	5	6	6092	0.68	0.97
Malware2-20	5	50	2105	0.28	1	-	-	-	-	-	3	6.8	4910	0.35	0.95
AttackGraph-1	3	50	54	0.59	1	1.89	2	59	0.95	0.43	-	-	-	-	-
AttackGraph-2	194	50	65	0.30	1	-	-	-	-	-	-	-	-	-	-

Table 1: Comparisons of planning time, plan diversity and average plan cost.

$$\text{Diversity}_{\text{stability}}(\Pi) = \frac{\sum_{\pi_i,\pi_j \in \Pi, i \neq j} [1 - \text{sim}_{\text{Jaccard}}(\pi_i, \pi_j)]}{|\Pi| \times (|\Pi| - 1)} \quad (2)$$

$$\text{Diversity}_{\text{uniqueness}}(\Pi) = \frac{\sum_{\pi_i,\pi_j \in \Pi, i \neq j} \begin{cases} 0 & \text{if } \pi_i \setminus \pi_j = \emptyset \\ 1 & \text{otherwise} \end{cases}}{|\Pi| \times (|\Pi| - 1)} \quad (3)$$

4.1 Experiment Results

We use a family of malware detection planning problems described in [Sohrabi *et al.*, 2013]. We varied the size of the problem by changing the number of observations, with Malware2 problem also supporting more system states. All planning problems share a planning domain description containing 6 actions and 8 predicates. In this domain, low costs were assigned to actions used in perfect explanations of observations, and high costs to actions representing exceptions, such as unexplained observations or state transitions without observations. For the attack graph model we are using the approach discussed in domain model section.

Table 1 summarizes the experiment results with two domain models. In all experiments we used a dual 16-core 2.70 GHz Intel(R) Xeon(R) E5-2680 processor with 256 GB RAM.The results presented in each row, corresponding to a planning domain. For Malware domains, the results are averages over 5 instances of each size. We have terminated planners after the time limit of 30 minutes was reached.

We compare the performance to two planning systems, LPG-d [Nguyen *et al.*, 2012] and Div [Roberts *et al.*, 2014], with our system that combines top-k and clustering.For both planners we have set the number of plans to 10, with the intent to produce a small number of attack scenarios to review.

The column **Time** in Table 1 contains planning time in seconds. The **#Plans** column contains the number of plans produced. The **Cost** column is the average cost of those plans. The **D** column is the average $\text{Diversity}_{\text{stability}}(\Pi)$, and the **U** column is the average $\text{Diversity}_{\text{uniqueness}}(\Pi)$. The closer these two diversity metrics are to 1, the more different are the plans.

We have selected LPG-d because it creates plans that maximize diversity, and that was confirmed by our experiments. Even though we have set parameter d to 0.1, to be equal to θ for our clustering algorithm, $\text{Diversity}_{\text{stability}}(\Pi)$ is above 0.8 in all experiments for this planner.

Div places greater emphasis on plan cost, and indeed average plan cost is lower than for LPG-d. However it sometimes produces multiple copies of the same plan, resulting in very poor diversity metrics. There are no AttackGraph results for Div, due to a crash without an error message.

The Top-k planner produced 1000 plans, which were later clustered. The number of clusters is determined by the similarity threshold θ, which was set to a low value 0.1 to ensure sufficient number of clusters, further bounded at maximum 50. It was the only planner that could solve the largest AttackGraph-2 instance with 4000 vulnerabilities and 1950 hosts. Overall this simple approach performs well in these domains in terms of planning time, and plan quality, but generally has lower plan stability metric that measures plan diversity. This is expected since we are trading off plan diversity for plan quality. In the network application, prioritizing cost is a desirable property because it eliminates from consideration attacks that are different from previously considered, but too inefficient to be carried out in practice. Of course, the quality and the applicability of the obtained solutions ultimately depends on knowledge engineering, and specifically on how action costs map to relevant attacks.

5 Related Work

Generating a plan set rather than just one plan has been a subject of interest in several recent papers in the context of generating diverse plans (e.g., [Roberts *et al.*, 2014; Coman and Muñoz-Avila, 2011]). Several plan distance measures most of which are domain-independent have been proposed to both guide the search and evaluate the set of diverse of plans (e.g., [Srivastava *et al.*, 2007; Bryce, 2014]). Given some partial preferences or multiple dimensions of quality, such as cost or time, the problem becomes a multi-objective optimization problem where diverse plans should form a Pareto-optimal set [Nguyen *et al.*, 2012]. Sroka and Long 2012 argue that the previous work will not find good-quality plans as they are more focused on finding diverse plans since it is "easier to find diverse sets father away from optimal". The work we presented in this paper falls in between. While we are given some notion of quality as measured by cost, the cost function itself is imperfect, and we are not given other objective functions besides costs. So finding one min-cost plan is not enough, nor is finding a diverse set of plans without taking into consideration the cost function. Hence, finding a set of diverse low-cost plans is required.

6 Conclusions and Future Work

In this paper we propose to address the plan space exploration problem arising in network security applications by generating high-quality diverse plans. We find that the existing work on diverse planning does not address this problem directly, and we propose a new approach specifically for this task, by combining top-k planning and plan clustering. Experimental evaluation shows that our new technique provides significant improvements in both plan quality and planning time. Although the primary focus of this work is to facilitate planning-assisted attack graph exploration carried out by network administrators, the techniques we are using are domain-independent, and future work may involve studying the applicability and the potential benefits of this approach in other applications, as well as integration with network security systems and evaluation via user studies.

References

[Aljazzar and Leue, 2011] Husain Aljazzar and Stefan Leue. K*: A heuristic search algorithm for finding the k shortest paths. *Artificial Intelligence*, 175(18):2129–2154, December 2011.

[Aslam *et al.*, 2004] J. A. Aslam, E. Pelekhov, and D. Rus. The Star Clustering Algorithm For Static And Dynamic Information Organization. *Journal of Graph Algorithms and Applications*, 8(1):95–129, 2004.

[Boddy *et al.*, 2005] Mark S. Boddy, Johnathan Gohde, Thomas Haigh, and Steven A. Harp. Course of action generation for cyber security using classical planning. In *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 12–21, 2005.

[Bryce, 2014] Daniel Bryce. Landmark-based plan distance measures for diverse planning. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 56–64, 2014.

[Coman and Muñoz-Avila, 2011] Alexandra Coman and Hector Muñoz-Avila. Generating diverse plans using quantitative and qualitative plan distance metrics. In *Proceedings of the 25th National Conference on Artificial Intelligence (AAAI)*, pages 946–951, 2011.

[Eppstein, 1998] David Eppstein. Finding the k shortest paths. *SIAM Journal on Computing*, 28(2):652–673, 1998.

[Filippone *et al.*, 2008] M. Filippone, F. Camastra, F. Masulli, and S. Rovetta. A Survey of Kernel and Spectral Methods for Clustering. *Pattern Recognition*, 41(1):176–190, 2008.

[Fox *et al.*, 2006] Maria Fox, Alfonso Gerevini, Derek Long, and Ivan Serina. Plan stability: Replanning versus plan repair. In *Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 212–221, 2006.

[Hassanzadeh and Miller, 2009] Oktie Hassanzadeh and Renée J. Miller. Creating Probabilistic Databases from Duplicated Data. *VLDB Journal*, 18(5):1141–1166, 2009.

[Helmert, 2006] Malte Helmert. The Fast Downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

[Hoffman and Pavley, 1959] Walter Hoffman and Richard Pavley. A method for the solution of the *n*th best path problem. *Journal of the ACM*, 6(4):506–514, 1959.

[Lucngeli *et al.*, 2010] Jorge Lucngeli, Carlos Sarraute, and Gerardo Richarte. Attack planning in the real world. In *Workshop on Intelligent Security (SecArt 2010)*, 2010.

[Nguyen *et al.*, 2012] Tuan Anh Nguyen, Minh Binh Do, Alfonso Gerevini, Ivan Serina, Biplav Srivastava, and Subbarao Kambhampati. Generating diverse plans to handle unknown and partially known user preferences. *Artificial Intelligence*, 190:1–31, 2012.

[Riabov *et al.*, 2014] Anton Riabov, Shirin Sohrabi, and Octavian Udrea. New algorithms for the top-k planning problem. In *Proceedings of the Scheduling and Planning Applications woRKshop (SPARK) at the 24th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 10–16, 2014.

[Roberts *et al.*, 2011] M. Roberts, A. Howe, I. Ray, M. Urbanska, Z. S. Byrne, and J. M. Weidert. Personalized vulnerability analysis through automated planning. In *Working Notes of IJCAI 2011, Workshop Security and Artificial Intelligence (SecArt-11)*, 2011.

[Roberts *et al.*, 2014] Mark Roberts, Adele E. Howe, and Indrajit Ray. Evaluating diversity in classical planning. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 253–261, 2014.

[Sohrabi *et al.*, 2010] Shirin Sohrabi, Jorge Baier, and Sheila McIlraith. Diagnosis as planning revisited. In *Proceedings of the 12th International Conference on the Principles of Knowledge Representation and Reasoning (KR)*, pages 26–36, 2010.

[Sohrabi *et al.*, 2013] Shirin Sohrabi, Octavian Udrea, and Anton Riabov. Hypothesis exploration for malware detection using planning. In *Proceedings of the 27th National Conference on Artificial Intelligence (AAAI)*, pages 883–889, 2013.

[Srivastava *et al.*, 2007] Biplav Srivastava, Tuan Anh Nguyen, Alfonso Gerevini, Subbarao Kambhampati, Minh Binh Do, and Ivan Serina. Domain independent approaches for finding diverse plans. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2016–2022, 2007.

[Sroka and Long, 2012] Michal Sroka and Derek Long. Exploring metric sensitivity of planners for generation of pareto frontiers. In *Proceedings of the 6th Starting AI Researchers' Symposium (STAIRS)*, pages 306–317, 2012.

[Xu and Wunsch, 2005] R. Xu and I. Wunsch. Survey of Clustering Algorithms. *IEEE Transactions on Neural Networks*, 16(3):645–678, 2005.

Using Operations Scheduling to Optimize Constellation Design

Steve Schaffer, Andrew Branch, Steve Chien, Stephen Broschart, Sonia Hernandez, Konstantin Belov, Joseph Lazio, Loren Clare, Philip Tsao, Julie Castillo-Rogez, E. Jay Wyatt

Jet Propulsion Laboratory, California Institute of Technology
Pasadena, CA, 91109-8099
firstname.lastname@jpl.nasa.gov

Abstract

Space mission design is a challenging task. Many factors combine to influence overall mission return, and it is extremely difficult a priori to predict which factors in concert will most influence mission return. These challenges are even greater for constellation missions, in which a potentially large number of spacecraft are used in concert to achieve mission goals, because constellations have additional design choices of number of spacecraft, orbit combinations, and constellation topology.

We describe efforts to use automated operations scheduling to assist in the design and analysis of a family of radio science constellation missions. Specifically, we work to produce a model-based approach to evaluating mission return based on key design variables of: target catalogue selection, constellation topology, size of the science constellation, size of the relay support network, orbit mix, communications capability, communications strategy, ground station configuration, onboard processing and compression, onboard storage, and other elements of operations concept.

In our design methodology, choices on the design dimensions are evaluated by producing mission plans using automated scheduling technology and these resultant plans are evaluated for science return. By this approach we intend to enable evaluation of large numbers of mission configurations (literally 10^6 configurations) with manual assessment of only a small number of the best of these configurations.

Introduction

Space mission design involves concurrent engineering on multiple disciplinary fronts in an effort to produce an overall configuration of spacecraft, orbit, and operations concept, to best achieve overall science objectives.

One of the challenges in space mission design is correctly accounting for a large number of design dimensions that may interact in subtle and hard to predict ways. We address this difficulty by adopting an operations-based approach to evaluating mission designs. We in effect partially simulate the missions, applying any and all operations constraints we can to derive results as realistic as possible. We then characterize the science measurements possible and use these as a proxy for mission return. By performing these simulations and calculations, we hope to estimate mission return and therefore enable devoting resources to the most promising early mission designs.

We investigate the use of this approach in the context of a large scale constellation to perform low frequency radio science measurements. Such a constellation would be placed well beyond Earth orbit – potential locations would be Lagrange points, Earth trailing, or a lunar orbit. Figure 1 shows a screen snapshot of a Cosmographia [NAIF] visualization of a constellation in a lunar orbit. This would enable the constellation to measure signals from beyond the interference of the Earth's ionosphere which restricts Earth-based arrays. The constellation would consist of a number of spacecraft: 16-128 spacecraft constitute a design range under study. Because the purpose of using multiple spacecraft is to synthesize a signal as if measured by a larger virtual antenna, ideally the spacecraft would be spread out at a range of distances from each other (from several km to several 1000 km) in a diverse spatial distribution. This dispersion presents challenges for communications – as communications rates decline with the square of the distance. Additionally – ground-based antenna arrays use antennae each of which weighs many tons and produces a Gigabit of data per second. In order for this constellation to be feasible the launch mass must be reduced to small spacecraft (< 10 kg per spacecraft) to reduce the expense of the mission. Additionally, the data volume must be reduced in order to be brought back to the Earth – bringing back 128 Gb/s from lunar orbit would require an extremely powerful communications setup and small spacecraft have very limited power.

In this paper we discuss the high level approach of the design process, the design dimensions of the constellation

mission, the details of our implementation, and some preliminary results.

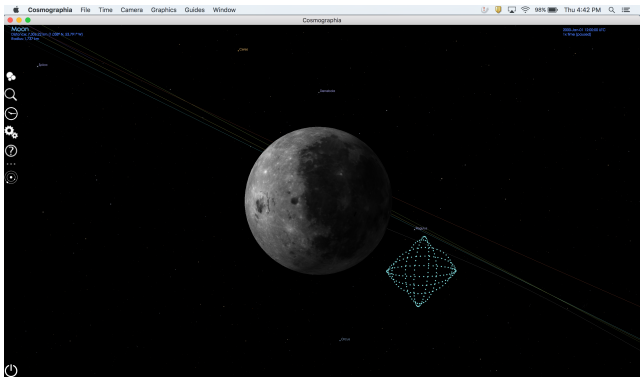


Figure 1: Cosmographia visualization of constellation of spacecraft in lunar orbit.

Mission Design

We formulate of the mission design problem is as follows.

Set of mission design dimensions:

$D_1 \ldots D_i$

For each design dimension D_j there are a set of k alternatives:

$D_{j,1} \ldots D_{j,k}$

Indeed there can be a continuous range of alternatives, for simplicity we restrict to finite discrete alternatives here.

A mission design can therefore be a choice of a single alternative for each of the design dimensions:

$D_{proposed} = <D_{1,a}, D_{2,b}, D_{3,c} \ldots D_{M,m}>$

We also presume a set of mission constraints:

$C_1 \ldots C_o$ where $C_j(D_{eval}) \rightarrow$ {True, False}

We also presume a mission score function $F(D_{eval}) \rightarrow$ integer.

The goal of the mission design process is to determine a mission design:

$D_{good} = < D_{1,a}, D_{2,b}, D_{3,c} \ldots D_{M,m} >$

Such that Forall constraints $C_{i=1} \ldots C_{i=o}$ $C_i(D_{good})$ = TRUE

(e.g. the mission passes all of the constraints)

and $F(D_{eval})$ is maximized.

Generate and Test Operations Evaluation-based Mission Design

Our overall approach is to enumerate a large proportion of the design space by enumerating a large number of design vectors (e.g. the design vectors D_{good} listed above). For each of these candidate designs that satisfies all mission constraints $C_1 \ldots C_i$, we automatically construct an operations model for the mission design and the use this operations model to generate a mission plan for the mission. This mission plan is automatically scored to estimate the $F(D_{eval})$.

Figure 2 below shows the flow of this general approach.

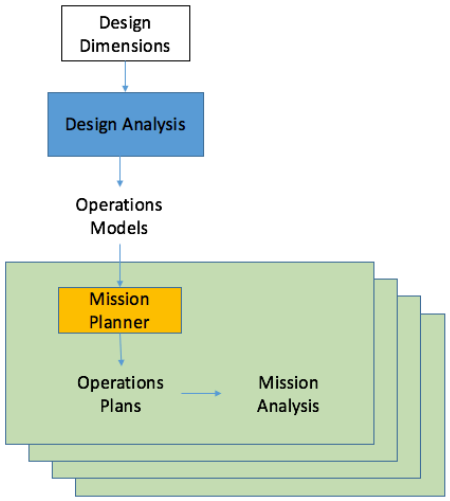


Figure 2: Generate and Test approach to spacecraft design configuration and operations analysis.

Dimensions of Constellation Mission Design Study

We now describe a number of the design dimensions we are analyzing in our constellation mission design study. These dimensions represent set D in the problem formulation.

Constellation Topology

One key aspect of constellation design is constellation topology. One possible configuration is that all of the spacecraft do not directly interact in operations and each directly transmits its data to ground communications stations. An-

other configuration uses one or more "mother ships" communicating with a larger number of daughter ships. In this setup commands are uplinked to a mother ship and relayed on to the daughter ships. Similarly, science and engineering data is cross linked from the daughter ships to a mother ship and then downlinked to Earth-based ground communications stations. This "star" configuration has the advantage that the mother ship relays can be higher powered and therefore more suited to the longer distance communication to the Earth. Additionally, the mother ship can be placed into an orbit advantageous for communications to Earth, whereas the daughter ships can be placed into orbits advantageous for science. The constellation topology also interacts with the sizing of the spacecraft onboard storage (e.g. solid state recorder). In a peer-based constellation the individual science craft must have significant SSR storage. In a star topology this storage can be concentrated at the mother ship(s).

Antenna Synthesis

As we are studying a radio science constellation, the primary science driver is the quality of radio antenna that can be synthesized. This quality is driven by several factors: antenna pattern coverage, individual antenna design and performance, and integration time.

Antenna pattern coverage for distributed antennae has been studied previously, mostly in the context of ground-based radio science arrays [Keto 1997, Boone 2001, Boone 2002]. In short, a good antenna pattern provides uniform coverage over a range of baseline lengths and orientations where a pairwise baseline length is the planar projective distance between the two receivers (e.g. spacecraft) and the orientation is the relative orientation of that baseline (all relative to the target).

Data Generation and Representation

The radio science measurements being made represent incredibly large amounts of raw data in a natural uncompressed format. For the low frequency measurement constellation we are studying, sampling and storing the data in the 30MHz regime (twice the frequency of interest) acquired at 12 bit resolution per sample at two polarizations results in a raw data rate of 0.7×10^9 bits per second per spacecraft of science data. A number of other measurements and storage options are possible at a potential reduction in science. These options are listed below. For each of these operational scenarios we evaluate the potential constellation return in terms of length, number, and qualities of science measurements possible.

Type of Data Acquired	Data Volume (bits/second) (2 polarizations, per s/c)
12bCC	0.7×10^9
12bPPCC	12×10^6
3bCC	180×10^6
3bPPCC	3×10^6
1bCC	60×10^6
1bPPCC	1×10^6
12bFx2ms	0.343×10^6

Orbit Selection and Design

The orbit of the spacecraft comprising the constellation drives many of the constellation performance factors. The orbit drives the antenna pattern coverage and therefore a good proportion of the science quality. A good combination of orbits will provide a good variation of baselines and orientations to provide good science.

Orbit selection also significantly affects communications. Communications data rate is proportional to d^{-2} where d is the distance between the two points in communications. Also occlusions by spacecraft or the moon can prevent communications. In a star mothership topology, the orbits dictate the cross link distance each spacecraft must communicate to the mother ship(s). The mothership orbit may be occluded from the Earth by the moon so we also analyze the coverage (visibility) of the mother ship(s) from the three Deep Space Network ground station locations: Canberra, Goldstone, and Madrid.

Spacecraft Design

Spacecraft design influences mission return in many ways. For the mothership relays, their communication capability is directly related to their power capability. Additionally, the mothership design may have one or multiple cross link antennae – each cross link antenna may be able to simultaneously receive a signal from a science spacecraft. There may however be a geometric constraint on the two cross linking spacecraft. Also, whether the mothership can crosslink and downlink to earth simultaneously is a major factor. As is the onboard solid state recorder capacity of the mothership (or each mothership if there are more than one).

The science spacecraft capabilities also will vary. Can each acquire science data and cross link simultaneously? How much power is available to cross link (affecting data rate)? How much onboard storage does each science craft have?

Additionally, for both the mother ship and science craft, what are the onboard maintenance activities that need to be performed and how will they impact science return?

Operations Planning

Once we have determined the above elements of the mission design we drive the design process using operations planning. The idea is that the operations model can be used to derive an estimate of the overall science return of the configuration.

Each full set of candidate design choices are semi-automatically encoded into separate domain models for the ASPEN/CASPER planning system [Chien et al. 2000a, 2000b]. ASPEN is a timeline-based scheduling framework that allows for operations, spacecraft, science, and other constraints to be incorporated in an automated scheduling environment.

The automatic scheduling algorithms then generate a proposed mission operations schedule constrained by those models. Each of the generated mission plans may then be evaluated for various metrics including science data utility, remaining resource margins, etc. The combined metrics for each design choice set can then be compared to select the best candidate mission designs for further evaluation.

The separate domain models for each point in the design space each leverage a common core of action/state models describing the entire space of available mission designs. The actions available in the common model span the entire constellation: some are executed only on individual science craft, some only on the mothership(s), and others require joint simultaneous action by multiple craft. The modeled actions include: repointing the field of view of the sensor, recording data from the sensor, crosslinking data from a science craft to a mothership, downlinking data from the mothership to earth, downlinking data directly from a science craft to earth, as well as placeholders for intermittently required engineering activities. These actions make use of various modeled states and resources: the visibility of each science target, the interferometry baseline utility of each observation window, the number of receivers on the mothership, the visibility of earth ground stations, the bandwidth of each communication link, power generation rate, remaining battery reserves, and so on.

Each complete set of concrete design choices then imposes additional constraints on the common base model. For example, the choice of sensor changes the field of view and scientific utility of observations, the choice of data storage device changes the available storage space and required power, and the selected transmitter power changes the available data bandwidth. These additional constraint inputs to the planner are generated from the mission design choices by a set of scripts dedicated to the task.

The CASPER automated operations planning system then uses the combined core model and design constraints to generate a proposed operations plan. CASPER starts from an empty mission plan and iteratively optimizes it by adding or removing actions to improve a declared utility function. The utility function is directly related to the calculated science utility of the data received at earth, and strongly inversely related to any mission constraint violations. This guides the planner to add observation, crosslink, and downlink activities while also respecting the design limits on view periods, storage space, bandwidth, power etc. As described earlier, the calculated utility of the science data is related to the total observation integration time and how well the selected interferometry baselines cover the space of distances and angles needed to characterize the structure of each radio astronomy target. The final output operations schedule from the planner includes concrete timed actions for each of the constellation craft to execute.

Critical to all of this operations planning is the geometric aspect of the problem. For all of these geometric analyses we use the SPICE package [Acton 1996]. These analyses include: spacecraft position and science target position for antenna analysis, spacecraft and mothership position for cross link calculations, and mothership and groundstation position and downlink calculation.

The operations schedule can then be evaluated versus various metrics that are interesting to the design team. These metrics represent F(D) in the problem definition. Foremost among these will be the overall predicted utility of the returned science data, as calculated by the planner's own utility function. Each of the component metrics of baseline coverage, integration time, target coverage is also reported for comparative consideration by the design team. Additional metrics such as excess unused capacity on some resources (e.g. unused power or bandwidth) are also reported to help inform which parts of each design may be over-engineered and which are the bottlenecks during actual operations.

Implementation

We are currently in early prototyping of our overall design evaluation system. The overall data flow of our prototype is shown below in Figure 3.

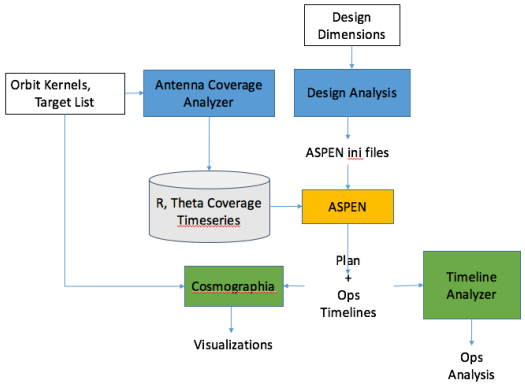


Figure 3: Design Operations Evaluation System Architecture

The overall range of design alternatives is described in an input file. This input file is used to semi-automatically generate a set of ASPEN models. Also input is a list of science targets for evaluation and a set of candidate orbits. These are analyzed by a separate code module which computes the antenna coverage pattern and baselines at each point in time for each target if the orbit were to be used and the relevant target were under observations.

In a run for each constellation configuration, ASPEN produces a plan as well as ancillary operations timelines. These indicate which spacecraft are observing which target at each point in time as well as the communications transfers required to return the data to Earth ground stations. This information can be used to analyze the operations performance of the constellation configuration and also the operations can be visualized within Cosmographia. For example, we can directly observe the state of the science spacecraft solid state recorders, or of the mothership relay solid state recorders. If these are constantly at capacity, we might infer that the bottleneck is the communications rates for either the cross link or the mothership to Earth.

The above architecture is in the process of being implemented and has already revealed some preliminary results which are being investigated further. Specifically, as we add more science spacecraft to the constellation, the number of independent measurements goes up linearly – so that the signal to noise improves linearly. However, addition of a spacecraft increases the number of pairwise observation baselines by n, for n spacecraft, i.e. the number of pairwise baselines increases as n^2 with the number of spacecraft. Therefore, we might expect the antenna coverage ratio to increase with the square of the number of spacecraft. However preliminary runs indicate that it is hard to realize even a linear increase in antenna coverage due to most orbits simply repeat coverage of already existing baselines (in distance and orientation) (Figure 4 below). As shown in this analysis, increases in the number of spacecraft show even declining (sub linear) increase in the antenna coverage pattern. However, this represents only initial results and much further study is needed. Figure 4 shows the antenna coverage ratio as random spacecraft from the orbits shown in Figure 1 are added to the constellation for 512 r bins x 512 theta bins (not equal area bins) computed over one repeat orbit cycles (about 8 hours 40 minutes) with a step size of 1 second theta range 0-110 degrees, and r range from 0 to 700 km.

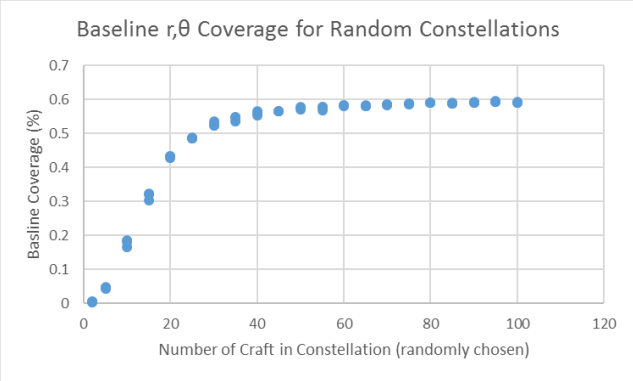


Figure 4: Plot for antenna coverage as a function of number of spacecraft

Related and Future Work

FRACSAT [Do et al 2013] uses forward state space search planning to generate all possible feasible configurations in a constellation design space. This approach is directly relevant to the first part of our methodology, i.e. generation of feasible alternatives to then simulate operations. While we currently use a hard coded approach for this phase, the FRACSAT approach is quite promising to explore and is an excellent area for future work. Also coming out of the F6 program (like FRACSAT), the work by Cornford [Cornford et al. 2012] considers more trades in the project management aspect of the design space such as when to commit to a certain design option or family of options.

The work described in this paper can be considered a continuing evolution of the planning for mission design approach previously described in [Knight et al. 2012] applied to the Desdyni mission (now called NI-SAR) and previously applied to SIM mission design [Smith et al. 2000], Europa mission Design [Rabideau et al. 2015] and Pluto Fast Flyby Misson Design [Sherwood et al. 1997]. In this approach, operations plans are generated for a range of mission configurations and these plans are evaluated with respect to mission objectives. Work on analyzing the BepiColumbo data management and downlink [DelaFuente et al. 2015] using automated downlink scheduling techniques can also be considered in the same approach – specifically using automated operations techniques to analyze and predict possible system performance prior to operations.

Work by Fukunaga [Fukunaga et al. 1997] also addresses automation of spacecraft design. This work also searches in the design space and simulates to evaluate mission performance. However, this approach does not use any planning or scheduling based operations model.

There are many areas of future work – this paper only describes very preliminary efforts towards operations-based

constellation design analysis. First, all of the models used thus far are quite primitive – using more refined accurate models would result in better results. Second, intelligent exploration of the design space rather than brute force sparse sampling would be much more effective. Third, we could introduce stochasticity in the operations model to evaluate a designs robustness to a wider range of scenarios. This stochasticity could represent either a wider range of operating scenarios or robustness to execution uncertainties.

Conclusions

We have presented preliminary work in using an operation-based planning model to evaluate design configurations for a radio science constellation mission concept. In this approach we enumerate a number of design alternatives, semi automatically generate operations models for each of these design alternatives, and use these operations models to generate baseline operations plans. These operations plan can then be analyzed to evaluate the constellation designs. This software prototype is in very preliminary stages and still undergoing evolution.

Acknowledgements

Portions of this work were performed by the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

References

Acton, C.H.; "Ancillary Data Services of NASA's Navigation and Ancillary Information Facility;" Planetary and Space Science, Vol. 44, No. 1, pp. 65-70, 1996.

Boone F. Interferometric array design: Optimizing the locations of the antenna pads. Astronomy & Astrophysics. 2001 Oct 1;377(1):368-76.

Boone F. Interferometric array design: Distributions of Fourier samples for imaging. Astronomy & Astrophysics. 2002 May 1;386(3):1160-71.

Chien S, Rabideau G, Knight R, Sherwood R, Engelhardt B, Mutz D, Estlin T, Smith B, Fisher F, Barrett T, Stebbins G. Aspen–automated planning and scheduling for space mission operations. InSpace Ops 2000 June, Toulouse, France, AIAA.

Chien S A, Knight R, Stechert A, Sherwood R, Rabideau G. Using Iterative Repair to Improve the Responsiveness of Planning and Scheduling. In Artificial Intelligence Planning Systems, 2000 Apr (pp. 300-307), AAAI Press.

Cornford S, Shishko R, Wall S, Cole B, Jenkins S, Rouquette N, Dubos G, Ryan T, Zarifian P, Durham B. Evaluating a Fractionated Spacecraft system: A business case tool for DARPA's F6 program. In Aerospace Conference, 2012 IEEE 2012 Mar 3 (pp. 1-20). IEEE.

S. de la Fuente, N. Policella, S. Fratini, J. McAuliffe, "Bepi-Colombo Science Data Storage and Downlink Optimization Tool," Intl Workshop on Planning and Scheduling for Space, Buenos Aires, Argentina, July 2015.

Do M, Feather M, Garcia D, Hicks K, Huang E, Kluck D, Mackey R, Nguyen T, Shah J, Stylianos Y, Tikidjian R. Synthesizing Fractionated Spacecraft Designs as a Planning Problem. Scheduling and Planning Applications workshop, Intl Conf on Planning and Scheduling, Rome , Italy, 2013.

A. Fukunaga, S. Chien, R. Sherwood D. Mutz, and A. Stechert. Automating the process of optimization in spacecraft design. In *Proc. of Aerospace Conference*, volume 4, pages 411–427, 1997.

E. Keto, The shapes of cross-correlation interferometers. The Astrophysical Journal. 1997 Feb 1;475(2):843.

R. Knight, D. McLaren, and S. Hu. Planning coverage campaigns for mission design and analysis: clasp for the proposed desdyni mission. In *Proc. of Intl Symposium on Artificial Intelligence, Robotics, and Automation for Space*, 2012.

Navigation ancillary information facility, Jet Propulsion Laboratory, California Institute of Technology, http://naif.jpl.nasa.gov/naif/cosmographia.html

G. Rabideau, S. Chien, E. Ferguson, Using Automated Scheduling for Mission Analysis and a Case Study Using the Europa Clipper Mission Concept. International Workshop on Planning and Scheduling for Space, (IWPSS 2015). Buenos Aires, Argentina. July 2015

R. Sherwood, S. Chien, G. Rabideau, T. Mann, Design for X (DFX) Operations Characteristic Spacecraft Design Analysis, International Workshop on Planning and Scheduling for Space 1997, Oxnard, CA.

B. Smith, B. Engelhardt, R. Knight, and D. Mutz. Automated planning for spacecraft and mission design. In *Proc. of 3rd International Symposium on Intelligent Automation and Control*, 2000.

Constructing Plan Trees for Simulated Penetration Testing

Dorin Shmaryahu and **Guy Shani**
Information Systems Engineering
Ben Gurion University, Israel

Joerg Hoffmann and **Marcel Steinmetz**
Department of Computer Science
Saarland University, Germany

Abstract

Penetration Testing (pentesting), where network administrators automatically attack their own network to identify and fix their vulnerabilities, has recently received attention from the AI community. Smart algorithms that can identify robust and efficient attack plans can imitate human hackers better than simple protocols. Current classical planning methods for pentesting model poorly the real world, where the attacker has only partial information concerning the network. On the other hand POMDP-based approaches provide a strong model, but fail to scale up to reasonable model sizes. In this paper we offer a more realistic model of the problem, allowing for partial observability and non-deterministic action effects, by modeling pentesting as a partially observable contingent problem. We suggest several optimization criteria, including worst case, best case, and fault tolerance. We experiment with benchmark networks, showing contingent planning to scale up to large networks.

1 Introduction

Penetration testing (pentesting) is a popular technique for identifying vulnerabilities in networks, by launching controlled attacks (Burns et al. 2007). A successful, or even a partially successful attack reveals weaknesses in the network, and allows the network administrators to remedy these weaknesses. Such attacks typically begin at one entrance point, and advance from one machine to another, through the network connections. For each attacked machine a series of known exploits is attempted, based on the machine configuration, until a successful exploit occurs. Then, this machine is controlled by the attacker, who can launch new attacks on connected machines. The attack continues until a machine inside the secure network is controlled, at which point the attacker can access data stored inside the secured network, or damage the network.

In automated planning the goal of an agent is to produce a plan to achieve specific goals, typically minimizing some performance metric such as overall cost. There are many variants of single agent automated planning problems, ranging from fully observable, deterministic domains, to partially observable, non-deterministic or stochastic domains. Automated planning was previously suggested as a tool for conducting pentesting, exploring the two extreme cases — a classical planning approach, where all actions are deterministic, and the entire network structure and machine configuration are known, and a POMDP approach, where machine configuration are unknown, but can be noisily sensed, and action outcomes are stochastic.

The classical planning approach scales well for large networks, and has therefore been used in practice for pentesting. However, the simplifying assumptions of complete knowledge and fully deterministic outcomes results in an overly optimistic attacker point-of-view. It may well be that a classical-planning attack has a significantly lower cost than a real attack, identifying vulnerabilities that are unlikely to be found and exploited by actual attackers.

The POMDP approach on the other hand, models the problem better, and can be argued to be a valid representation of the real world. One can model the prior probabilities of various configurations for each machine as a probability distribution over possible states, known as a belief. Pinging actions, designed to reveal configuration properties of machines are modeled as sensing actions, and a probability distribution can be defined for the possible failure in pinging a machine. The success or failure of attempting an exploit over a machine can be modeled as a stochastic effect of actions.

This approach, however, has two major weaknesses — first, POMDP solvers do not scale to the required network size and possible configurations. Second, a POMDP requires accurate probability distributions for initial belief, sensing accuracy, and action outcomes. In pentesting, as in many other applications, it is unclear how the agent can reliably obtain these distributions. In particular, how to identify an accurate probability distribution over the possible OS for the machines in the network? Prior work (Sarraute *et al.*) has devised only a first over-simplifying model of ”software updates”, which the authors admit themselves is not suitable and may adversely affect the usefulness of the pentesting result (”garbage in, garbage out”). One might consider research into obtaining better distributions, e.g. by statistics from data, but this is wide open, and in any case the scalability weakness remains.

A possible simple approach to defining such probabilities is to use a uniform distribution. However, a solution to a POMDP defined using a uniform distribution can be arbitrarily bad. Consider, for example, a case where there exists a large set of configurations that are easy to penetrate, such as

a variety of old, unupdated operating systems. All these configurations may be very rare in the network, yet still exist on some machines, and are hence represented in the model. Assuming a uniform distribution over possible configurations, an attacker may believe that these vulnerable configurations are as frequent as any other configuration, and may hence attempt a long sequence of exploits which will work only for these faulty configurations. In such a case, the performance of the agent measured over the uniform POMDP, may be arbitrarily far from its performance in practice.

As an intermediate model between classical planning and POMDPs, MDP models of pentesting have been suggested (Durkota *et al.* 2015; Hoffmann 2015). These somewhat simplify the issue of obtaining the probabilities, now corresponding to "success statistics" for exploits. Yet even this data is not easy to come by in practice, and scalability may still be problematic given that solving factored MDPs is notoriously hard (a thorough empirical investigation has yet to be conducted).

In this paper we suggest another, different, intermediate model between classical planning and POMDPs. We replace the POMDP definition with partially observable contingent planning, a qualitative model where probability distributions are replaced with sets of possible configurations or action effects (Albore *et al.* 2009; Muise *et al.* 2014; Komarnitsky and Shani 2014). Solvers for this type of models scale better than POMDP solvers, and can be used for more practical networks. As these models require no probabilities, we avoid the guesswork inherent in their specification.

Contingent planners attempt to find a plan tree (or graph), where nodes are labeled by actions, and edges are labeled by observations. This plan tree is a solution to the problem if all leaves represent goal states. In pentesting, one is also interested in finding better attacks, i.e. in ranking the set of possible plan trees by some measurable quantity. For example, an attacker may be interested in attacks that, at the worst case, take no more than a certain amount of time. An important research question is, hence, to define possible optimization criteria for attack plan trees. Then, one must design algorithms dedicated to these optimization criteria.

We focus here on the first question — possible optimization criteria for ranking contingent plan trees. We suggest a number of such criteria, including best and worst case, budget constrained plans, and fault-tolerant planning (Domshlak 2001). We also consider deadends, which arise in pentesting as some machine configurations cannot be penetrated, leaving no opportunity to the attacker to reach its goal. We discuss how to define and compare contingent plans under such unavoidable deadends.

We demonstrate empirically that different heuristics produce different plan trees, and that these plan trees can be compared using our optimization criteria, to prefer one heuristic over another. We leave the construction of optimal and approximate contingent planners for future research.

2 Networks and Pentesting

We begin by providing a short background on pentesting.

We can model networks as directed graphs whose vertices are a set M of *machines*, and edges representing connections between pairs of $m \in M$. Like previous work in the area, we assume below that the attacker knows the structure of the network. But this assumption can be easily removed in our approach. We can add sensing actions that test the outgoing edges from a controlled host to identify its immediate neighbors. From an optimization prespective, though, not knowing anything about the network structure, makes it difficult to create smart attacks, and the attacker is forced to blindly tread into the network. It might well be that some partial information concerning the network structure is known to the attacker, while additional information must be sensed. We leave discussion of interesting forms of partial knowledge to future work.

Each machine in the network can have a different *configuration* representing its hardware, operating system, installed updates and service packs, installed software, and so forth. The network configuration is the set of all machine configurations in the network.

Machine configuration may be revealed using sensing techniques. For example, if a certain series of 4 TCP requests are sent at exact time intervals to a target machine, the responses of the target machine vary between different versions of Windows (Lyon 2009). In many cases several different such methods must be combined to identify the operating system. Sending such seemingly innocent requests to a machine to identify its configuration is known as fingerprinting. Not all the properties of a target machine can be identified. For example, one may determine that a certain machine runs Windows XP, but not which security update is installed.

Many configurations have *vulnerabilities* that can be *exploited* to gain control over the machine, but these vulnerabilities vary between configurations. Thus, to control a machine, one first pings it to identify some configuration properties, and based on these properties attempts several appropriate exploits. As the attacker cannot fully observe the configuration, these exploits may succeed, giving the attacker full control of the target machine, or fail as some undetectable configuration property made this exploit useless.

The objective of penetration testing (pentesting) is to gain control over certain machines that possess critical content in the network. We say that a machine m is *controlled* if it has already been hacked into, and the attacker can use it to fingerprint and attack other machines. A *reached* machine m is connected to a controlled machine. All other machines are *not reached*. We assume that the attacker starts controlling the internet, and all machines that are directly connected to the internet are reached.

We will use the following (small but real-life) situation as an illustrative example (Sarraute *et al.*):

Example 2.1. The attacker has already hacked into a machine m', and now wishes to attack a reached machine m. The attacker may try one of two exploits: *SA*, the "Symantec Rtvscan buffer overflow exploit"; and *CAU*, the "CA Unicenter message queuing exploit". SA targets a particular version of "Symantec Antivirus", that usually listens on port 2967. CAU targets a particular version of "CA Unicenter", that usually listens on port 6668. Both work only if a protection mechanism called *DEP* ("Data Execution Pre-

vention") is disabled. The attacker cannot directly observe whether DEP is enabled or not.

If SA fails, then it is likely that CAU will fail as well because DEP is enabled. Hence, upon observing the result of the SA exploit, the attacker learns whether DEP is enabled. The attacker is then better off trying other exploits else. Achieving such behavior requires the attack plan to observe the outcomes of actions, and to react accordingly. Classical planning which assumes perfect world knowledge at planning time cannot model such behaviors.

3 Contingent Planning Model and Language

A contingent planning problem is a tuple $< P, A_{act}, A_{sense}, \phi_I, G >$, where P is a set of propositions, A_{act} is a set of actuation actions, and A_{sense} is a set of sensing actions. An actuation action is defined by a set of preconditions — propositions that must hold prior to executing the actions, and effects — propositions that hold after executing the action. Sensing actions also have preconditions, but instead of effects they reveal the value of a set of propositions. ϕ_I is a propositional formula describing the set of initially possible states. $G \subset P$ is a set of goal propositions.

In our pentesting application, P contains propositions for describing machine configuration, such as $OS(m_i, winxp)$, denoting that machine m_i runs the OS Windows XP. Similarly, $SW(m_i, IIS)$ represents the existence of the software IIS on machine m_i. In addition, the proposition $controlling(m_i)$ denotes that the attacker currently controls m_i, and the proposition $hacl(m_i, m_j, p)$ denotes that machine m_i is directly connected to machine m_j through port p.

The set A_{sense} in our pentesting model represents the set of possible queries that one machine can launch on another, directly connected machine, pinging it for various properties, such as its OS, software that runs on it, and so forth. Each such sensing action requires as precondition only that the machines will be connected, and reveals the value of a specific property. In some cases there are certain "groups" of operating systems, such as Windows XP with varying service packs and updates installed. In this case we can allow one property for the group ($OS(m_i, winxp)$) and another property for the version, such as ($OSVersion(m_i, winxp_sp1)$) which may not be observable by the attacker.

The set A_{act} in our pentesting model contains all the possible exploits. We create an action $a_{e,m_{source},m_{target}}$ for each exploit e and a pair of directly connected machines m_{source}, m_{target}. If an exploit e is applicable only to machines running Windows XP, then $OS(m_{target}, winxp)$ would appear in the preconditions. Another precondition is $controlling(m_{source})$ denoting that the attacker must control m_{source} before launching attacks from it. The effect of the action can be $controlling(m_{target})$, but we further allow the effect to depend on some hidden property p that cannot be sensed. This is modeled by a conditional effect $\langle p, controlling(m_{target})\rangle$ denoting that if property p exists on m_{target} than following the action the attacker controls m_{target}.

Belief states in contingent planning are sets of possible states, and can often be compactly represented by logic formulas. The initial belief formula ϕ_I represents the knowledge of the attacker over the possible configurations of each machine. For example $oneof(OS(m_i, winxp), OS(m_i, winnt4), OS(m_i, win7))$ states that the possible operating systems for machine m_i are Windows XP, Windows NT4, and Windows 7.

Like Sarraute et al., we assume no non-determinism, i.e., if all properties of a configuration are known, then we can predict deterministically whether an exploit will succeed. We do allow for non-observable properties, such as the service pack installed for the specific operating system. We support actions for sensing whether an exploit has succeeded. Hence, observing the result of an exploit action reveals information concerning these hidden properties.

Example 3.1. We illustrate the above ideas using a very small example, written in a PDDL-like language for describing contingent problems (Albore *et al.* 2009).

We use propositions to describe the various properties of the machines and the network. For example, the predicate *(hacl* $?m_1$ $?m_2$*)* specifies whether machine m_1 is connected to machine m_2, and the predicate *(HostOS ?m ?o)* specifies whether machine m runs OS o. While in this simple example we observe the specific OS, we could separate OS type and edition (say, Windows NT4 is the type, while Server or Enterprise is the edition). We can then allow different sensing actions for type and edition, or allow only sensing of type while edition cannot be directly sensed.

We define actions for pinging certain properties. For example, the *ping-os* action:

```
(:action ping-os
    :parameters (?src - host ?target - host ?o - os)
    :precondition (and (hacl ?src ?target)
                       (controlling ?src)
                       (not(controlling ?target))
    :observe (HostOS ?target ?o)
)
```

allows an attacker that controls host s connected to an uncontrolled host t, to ping it to identify whether it's OS is o. We allow for a similar ping action for installed software.

The *exploit* action attempts to attack a machine exploiting a specific vulnerability:

```
(:action exploit
    :parameters (?src - host ?target - host ?o - os ?sw - sw
                 ?v - vuln)
    :precondition (and (hacl ?src ?target)
                       (controlling ?src)
                       (not(controlling ?target))
                       (HostOS ?target ?o)
                       (HostSW ?target ?sw)
                       (Match ?o ?sw ?v))
    :effect (when (ExistVuln ?v ?target) (controlling ?target))
)
```

The preconditions specify that the machines must be connected, that the OS is o and the software sw is installed, and that the vulnerability v which we intend to exploit matches the specific OS and software.

The success of the exploit depends on whether the vulnerability exists on the target machine, which manifests in the conditional effect. The attacker cannot directly observe whether a specific vulnerability exists, but can use the *CheckControl* action to check whether the exploit has succeeded:

```
(:action CheckControl
    :parameters (?src - host ?target - host)
    :precondition (and (hacl ?src ?target ?p)
                       (controlling ?src))
    :observe (controlling ?target)
)
```

The initial state of the problem describes the knowledge of the attacker prior to launching an attack:

```
(:init
1:    (controlling internet)
2:    (hacl internet host0)
      (hacl internet host1)
      (hacl host1 host2)
      (hacl host0 host2)
      ...
3:    (oneof (HostOS host0 winNT4ser) (HostOS host0 winNT4ent))
      (oneof (HostOS host1 win7ent) (HostOS host1 winNT4ent))
      ...
4:    (oneof (HostSW host0 IIS4) (HostSW host1 IIS4))
      ...
5:    (Match winNT4ser IIS4 vuln1)
      ...
6:    (or (ExistVuln vuln1 host0) (ExistVuln vuln2 host0))
      ...
)
```

We state that initially the attacker controls the "internet" only (part 1). In this case the structure of the network is known, described by the *hacl* statements (part 2). Then, we describe which operating systems are possible for each of the hosts (part 3). Below, we specify that either $host0$ or $host1$ are running the software IIS (part 4). We describe which vulnerability is relevant to a certain OS-software pair (part 5), and then describe which vulnerabilities exit on the various hosts (part 6).

The above specification may allow for a configuration where no vulnerability exists on a host (machine) that matches the host OS and software. Hence, none of the exploits will work for that specific host.

4 Plan Trees and Optimization Criteria

We now formally define solutions to a contingent planning problem. We discuss deadends that arise in pentesting, and then turn our attention to a discussion of optimization criteria.

4.1 Contingent Plan Trees

A solution to a contingent planning problem is a plan tree, where nodes are labeled by actions. A node labeled by an actuation action will have only a single child, and a node labeled by an observation action will have multiple children, and each outgoing edge to a child will be labeled by a possible observation.

An action a is applicable in belief state b, if for all $s \in b$, $s \models pre(a)$. The belief state b' resulting from the execution of a in b is denoted $a(b)$. We denote the execution of a sequence of actions $a_1^n =< a_1, a_2, ..., a_n >$ starting from belief state b by $a_1^n(b)$. Such an execution is valid if for all i, a_i is applicable in $a_1^{i-1}(b)$.

Plan trees can often be represented more compactly as plan graphs (Komarnitsky and Shani 2014; Muise *et al.* 2014), where certain branches are unified. This can lead to a much more compact representation, and to scaling up to larger domains. Still, for ease of exposition, we discuss below plan trees rather than graphs.

In general contingent planning, a plan tree is a solution, if every branch in the tree from the root to a leaf, labeled by actions a_1^n, $a_1^n(b_I) \models G$. In pentesting, however, it may not be possible to reach the goal in all cases, because there may be network configurations from which the target machine simply cannot be reached. To cater for this, we need to permit plan trees that contain *dead-ends*. We define a dead-end to be a state from which there is no path to the goal, given *any* future sequence of observations. That is, any plan tree starting from a dead-end state would not reach the goal in any of its branches. For example, a dead-end state arises if no exploit is applicable for the goal machine. It is clearly advisable to stop the plan (the attack) at such states. On the other hand, if a state is not a dead-end, then there still is a chance to reach the target so the plan/attack should continue.

There is hence need to define contingent plans where some of the branches may end in dead-ends. A simple solution, customary in probabilistic models, is to introduce a give-up action which allows to achieve the goal from any state. Setting the cost of that action (its negative reward) controls the extent to which the attacker will be persistent, through the mechanism of expected cost/expected reward.

In a qualitative model like ours, it is not as clear what the cost of giving up (effectively, of flagging a state as "dead-end" and disregarding it) should be. It may be possible to set this cost high enough to force the plan to give up only on dead-ends as defined above. But then, the contingent planner would effectively need to search all contingent plans not giving up, before being able to give up even once.

We therefore employ here a different approach, allowing the planner to give-up on s iff it can prove that s is a dead-end. Such proofs can be lead by classical-planning dead-end detection methods, like relaxation/abstraction heuristics, adapted to our context by determinizing the sensing actions, allowing the dead-end detector to choose the outcome. In other words, we employ a sufficient criterion to detect dead-end states, and we make the give-up action applicable only on such states. As, beneath all dead-ends, eventually the pentest will run out of applicable actions, eventually every dead-end will be detected and the give-up enabled.

In general, this definition would not be enough because the planner could willfully choose to move into a dead-end, thereby "solving" the task by earning the right to give up. This cannot happen, however, in the pentesting application, as all dead-ends are *unavoidable*, in the following sense. Say N is a node in our plan tree T, and denote by $[N]$ those initial states from which the execution of T will reach N. If N

is a dead-end, then every $I \in [N]$ is unsolvable, i.e., there does not exist any sequence of A_{act} actions leading from I to the goal. In other words, any dead-end the contingent plan may encounter is, in the pentesting application, inherent in the initial state. Matters change if we impose a budget limit on the attack, in which case the dead-ends encountered depend on which decisions are taken. We define an according plan quality criterion as part of the next subsection.

4.2 Optimization Criteria for Contingent Plans

General contingent planning follows the satisfying planning criterion, that is, one seeks any solution plan tree. It is possible, though, to consider cases where one plan tree is preferable to another, and construct algorithms that seek better, or even the best possible plan tree.

When we assume that the environment is modeled as a POMDP, and we know all the probability distributions, an obvious optimization criterion is the expected discounted reward (or cost) from executing a plan tree in the environment, and can be estimated by running multiple trials and computing the average discounted reward (ADR). In this paper, however, we focus on cases where these distributions are unknown. Without the specified distributions one cannot accurately estimate expected reward. Any attempt to use a different distribution, such as a uniform distribution, which may be arbitrarily far from the true distribution, may result in quality estimation that is arbitrarily far from reality.

We hence revert to other possible optimization criteria. Perhaps the most trivial optimization criteria under unknown probability distributions is the best case scenario, or the worst case scenario. In the best case scenario we compare plan trees based on the length of the shortest branch leading to a goal state. In the worst case scenario we compare the length of the longest branch leading to a goal state, preferring plan trees with shorter worst case branches. This may be somewhat different than the naive definition of a worst case, as a complete failure is obviously worse (less desirable) than a success after a lengthy sequence of actions. In our case, as the deadends in the plan trees are unavoidable, the naive worst case — a complete failure — is identical in all plan trees. We thus choose to ignore branches ending with deadends when considering worst case analysis.

While well defined, best and worst case optimization may not be sufficiently expressive. A best case scenario is too optimistic, assuming that all attack attempts will be successful. A worst case scenario is over pessimistic, assuming that all attack attempts, but the last one, will fail. We would like to define finer optimization criteria.

Budget Optimization One possible such criterion assumes attacks on a budget — that is, the attacker is allowed only a certain predefined number of actions (or total cost) in a branch. When the budget runs out, the attacker is not allowed any additional actions, and hence, a deadend occurs. Setting a budget prior to attacking seems like a reasonable requirement from an attacker. For example, if action costs represent the time it takes for each action, the attacker may wish to restrict attention only to attacks that require less than a certain amount of time.

Now, given two plan trees that respect a given budget, we can compare them on two possible criteria — the best case scenario and the set of solved network configurations. The worst case scenario is less interesting here as it will probably be identical to the budget.

The set of network configurations where the attacker has reached the goal under the budget is now interesting, because deadends induced by the budget may well be avoidable. That is, one can choose different attack plans, that may lead to the goal faster and hence will result in less deadends. However, simply counting the number of network configurations for which the goal has been reached is undesirable under our qualitative assumptions. For example, it may well be that plan tree τ_1 solves only for a single configuration c, while another plan tree τ_2 solves for all configurations but c. Still, it may be that the (unknown) probability of c is 0.9, making τ_1 preferable to τ_2. As we do not know these probabilities, we cannot make such comparisons.

We can hence only declare plan tree τ_1 to be better than plan tree τ_2 if the set of solved configurations ofτ_1 is a strict superset of the set of solved configurations of τ_2. As contingent planners typically maintain some type of belief over the set of possible network configurations in each search node, such computations are feasible. For example, if the belief is maintained by a logic formula, as we do, then each goal leaf g has a logic formula ϕ_g defining the belief at that leaf. We can check whether

$$\bigvee_{g \in G(\tau_1)} \phi_g \models \bigvee_{g \in G(\tau_2)} \phi_g \quad (1)$$

$$\bigvee_{g \in G(\tau_2)} \phi_g \not\models \bigvee_{g \in G(\tau_1)} \phi_g \quad (2)$$

where $G(\tau)$ is the set of goal leaves in plan tree τ.

Fault Tolerance Optimization Another possible optimization is by extending the ideas of fault-tolerance planning to pentesting. In fault-tolerance planning (Domshlak 2001), assuming that certain actions may fail with some low probability, a solution achieves the goal under the assumption that no more than k failures will occur. The underlying assumption is that the probability of more than k failures is so small, that we can ignore it. A failure in our case can be defined in one of two ways — either that we will ping a machine for a given property (say, $OS(m_i, winxp)$) and receive a negative response. Alternatively, we may declare a failure only when we attempt an exploit, and it fails to achieve control of a machine (due to some unobserved property).

With that view in mind, we can compare solution plan trees, focusing only on branches that contain exactly k failures. As having no more than k failures is an optimistic assumption, it is reasonable to check the worst case under this optimistic assumption. That is, of the branches of the plan tree that have the lowest probability that we care about, we compare the longest branches. Looking at the best case — the shortest branch when having no more than k failures, is identical to the overall best case scenario, ignoring failures all together.

A complementing approach assumes no less than k failures at each branch. This assumption is more appropriate where the probability of failure is sufficiently large, such that the probability of completing a task without any failure is very low. In such cases, we again compare only branches with exactly k branches, and as no less than k failures is a pessimistic assumption, we compare the best case scenario — the shortest branch with exactly k failures. Again, the worst case is less interesting as it is identical to the overall worst case.

5 Empirical Study

Size	k	Method	Shortest	Longest
4		h_0	8	14
4		h_1	13	28
4	0	h_0	8	9
4	0	h_1	13	18
4	1	h_0	11	12
4	1	h_1	15	21
4	2	h_0	14	14
4	2	h_1	17	24
4	3	h_0	$\times$	$\times$
4	3	h_1	19	26
4	6	h_0	$\times$	$\times$
4	6	h_1	28	28
8		h_0	8	15
8		h_1	10	27
8	0	h_0	8	8
8	0	h_1	10	18
8	1	h_0	10	10
8	1	h_1	12	20
8	2	h_0	12	13
8	2	h_1	14	23
8	3	h_0	15	15
8	3	h_1	17	24
8	6	h_0	$\times$	$\times$
8	6	h_1	24	28
16		h_0	8	12
16		h_1	13	22
16	0	h_0	8	8
16	0	h_1	13	16
16	1	h_0	10	10
16	1	h_1	15	18
16	2	h_0	12	12
16	2	h_1	17	20
16	3	h_0	$\times$	$\times$
16	3	h_1	20	22

Table 1: Comparing the performance of h_0 and h_1 over fault tolerance optimization, for varying network sizes (number of hosts). The first line for each network size is the overall best and worst case. k denotes the number of failures. In cases where h_0 did not produce k failures in any path, we marked the entry with $\times$.

We now provide some empirical proof-of-concept for our optimization criteria. Specifically, we demonstrate that the criteria we suggest above can be used to differentiate between various plan trees (graphs), helping us to select a better algorithm. We generate a number of networks of varying sizes using the generator of Hoffman and Steinmetz(Sarraute *et al.* ; Steinmetz *et al.*).

We use a simple greedy best first contingent planner that uses a heuristic to determine which state and action to expand next. In addition, we use a mechanism for detecting repeated plan tree nodes, converting the plan tree into a plan graph. We augment this algorithm with a domain specific deadend detection mechanism, checking whether there is still a path from the attack source ("the internet") to the target host.

We employ several domain specific heuristics, that leverage the network graph. We chose the next host to attack among the hosts closest to the goal host. When choosing actions, we first ping a host for its operating system, and then we ping it only for software that, combined with the observed OS, may have a vulnerability. If a possible vulnerability has been detected, we attempt an exploit, followed by sensing action to check if control was gained over the attacked host.

This simple heuristic, which we denote h_0 proves to be highly effective for this application, and we manage to produce attack graphs for networks with 80 hosts and more. However, as we are interested in generating a variety of plan trees, we also implemented a variant of the above heuristic — choosing the next host to attack only (denoted h_1). Sadly, although not surprisingly, h_1 scale much worse. We hence report results on much smaller domains, ranging from 4 to 16 hosts.

We compare these 2 heuristics over the various network sizes. Table 1 reports the best and worst case in the fault tolerance scenario, with different values of k. As can be seen, the trees vary on the lengths of the shortest and longest path to the goal. This shows that the fault tolerance optimization metric can differentiate between different trees. In this specific case, h_0 produced better trees than h_1 for all values of k. We can hence deduce that h_0, on top of being much faster, is also better than h_1 for the networks that we experimented with.

6 Conclusion and Future Work

In this paper we suggest contingent planning as an attractive option for modeling pentesting. This model allows for partial observability of various properties, such as a machine operating system and installed software, that can be sensed by ping actions. Thus, contingent planning offers a richer model than classical planning, while being able to scale up better than POMDP-based approaches.

We focus our attention on defining optimization criteria that allow the attacker to prefer one attack plan over the other. We discuss best and worst case, as well as budgeted attacks and fault tolerance approaches.

We provide some cal validation, showing that different algorithms generate different attack plans, that can be ranked using our optimization criteria.

An obvious next step on our research agenda is the development of contingent planners and heuristics that produce optimal plans given each optimization criterion, as well as approximate, scalable algorithms, that produce good, if not optimal, plans rapidly.

Acknowledgments: We thank the reviewers for their useful comments. This work was supported by ISF Grant 933/13, and by the Helmsley Charitable Trust through the Agricultural, Biological and Cognitive Robotics Center of Ben-Gurion University of the Negev.

References

Alexandre Albore, Héctor Palacios, and Hector Geffner. A translation-based approach to contingent planning. In *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, pages 1623–1628, 2009.

Burns et al. *Security Power Tools*. O'Reilly Media, 2007.

Carmel Domshlak. Fault tolerant planning: Complexity and compilation. volume 22, pages –, 2001.

Karel Durkota, Viliam Lisý, Branislav Bosanský, and Christopher Kiekintveld. Optimal network security hardening using attack graph games. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 526–532, 2015.

Jörg Hoffmann. Simulated penetration testing: From "dijkstra" to "turing test++". In *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling, ICAPS 2015, Jerusalem, Israel, June 7-11, 2015.*, pages 364–372, 2015.

Radimir Komarnitsky and Guy Shani. Computing contingent plans using online replanning. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, pages 2322–2329, 2014.

Gordon Fyodor Lyon. *Nmap network scanning: The official Nmap project guide to network discovery and security scanning*. Insecure, 2009.

Christian J. Muise, Vaishak Belle, and Sheila A. McIlraith. Computing contingent plans via fully observable non-deterministic planning. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 - 31, 2014, Québec City, Québec, Canada.*, pages 2322–2329, 2014.

Carlos Sarraute, Olivier Buffet, and Jörg Hoffmann. POMDPs make better hackers: Accounting for uncertainty in penetration testing.

Marcel Steinmetz, Jörg Hoffmann, and Olivier Buffet. Revisiting goal probability analysis in probabilistic planning.

Prioritization and Oversubscribed Scheduling for NASA's Deep Space Network

Caroline Shouraboura*, Mark D. Johnston† and Daniel Tran†

*Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh PA 15213
cshourab @ andrew.cmu.edu

†Jet Propulsion Laboratory, California Institute of Technology, 4800 Oak Grove Dr., Pasadena CA 91109
{mark.d.johnston, daniel.tran} @ jpl.nasa.gov

Abstract

NASA's Deep Space Network (DSN) is a unique facility responsible for communication and navigation support for over forty NASA and international space missions. For many years, demand on the network has been greater than its capacity, and so a collaborative negotiation process has been developed among the network's users to resolve contention and come to agreement on the schedule. This process has become strained by increasing demand, to the point that oversubscription is routinely as high as 40% over actual capacity. As a result, DSN has started investigating the possibility of moving to some kind of prioritization scheme to allow for more automated and timely resolution of network contention. Other NASA networks have used strict static mission priorities, but if this were applied in the same way to the DSN, some missions would fall out of the schedule altogether. In this paper we report on analysis and experimentation with several approaches to DSN prioritization. Our objectives include preserving as much of each each mission's requested contact time as possible, while allowing them to identify which of their specific scheduling requests are of greatest importance to them. We have obtained the most promising results with a variant of Squeaky Wheel Optimization combined with limiting each mission's input based on historical negotiated reduction levels.

1. Introduction

NASA's Deep Space Network (DSN) consists of three communications complexes, located in Goldstone, California; Madrid, Spain; and Canberra, Australia. Each complex contains one 70-meter antenna and three or four 34-meter antennas. These ground antennas are responsible for communications and navigation support for a wide range of scientific space missions, from those in highly elliptical earth orbits, to some beyond the solar system. In future years, DSN will also support human missions to the moon and beyond.

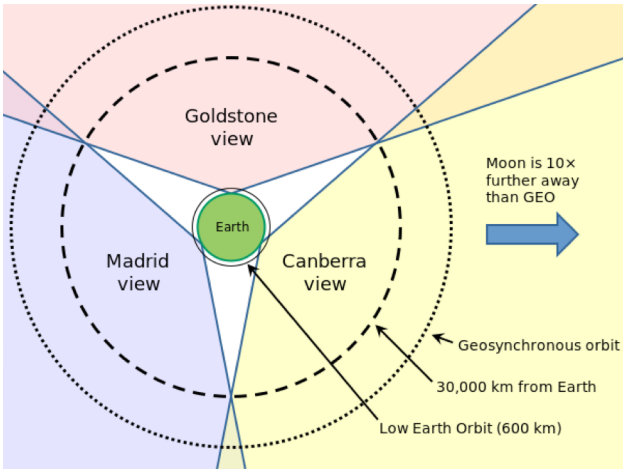


Fig. 1: Fields of the view of the DSN complexes, showing overlapped coverage for distant spacecraft.

The placement of the three DSN complexes allows at least one of them to be in view of any distant spacecraft at all times (Fig. 1); see (Imbriale, 2003).

At this time there are approximately forty missions using the Deep Space Network. There is a natural cycle of missions ending and new missions starting up in their *prime* mission phase, but the majority of DSN users are in their *extended* mission phases. The distinction between prime and extended missions plays a role in some prioritization suggestions, as will be discussed below.

The DSN has seen an increasing level of oversubscription in recent years. In 2011/2012 there was roughly 300 hours per week total difference between initial and final mission time allocations (how much tracking time was requested vs. how much could actually be scheduled.) However, in late 2015, this had grown to as much as 500 hours per week, an

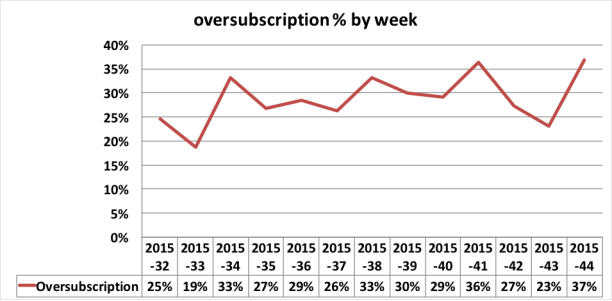


Fig. 2: Oversubscription percentage by week for a range of 13 weeks in late 2015. The average is about 30% with a standard deviation of 5%

increase of over 50% (Fig. 2). To place this in context, the weekly oversubscription amounts to about *4 additional antennas* worth of activities, over and above the 13 actual (34m and 70m) antennas in the DSN.

Along with oversubscription, the number of conflicts in the schedule has also increased. Most conflicts are due to oversubscribed resources, i.e. antennas or other assets at the DSN complexes. Resolving these conflicts takes increasingly high levels of human effort, since they are phrased as irreducible time requirements.

In the following (Section 2) we first briefly describe the DSN scheduling process, highlighting where oversubscription impacts the scheduling software and processes that generate and manage DSN schedules. We then describe some of the factors that come into play in evaluating priority schemes for the DSN (Section 3). Results of a series of experiments with different algorithms are then presented and discussed (Section 4), followed by conclusions and directions for future work (Section 5).

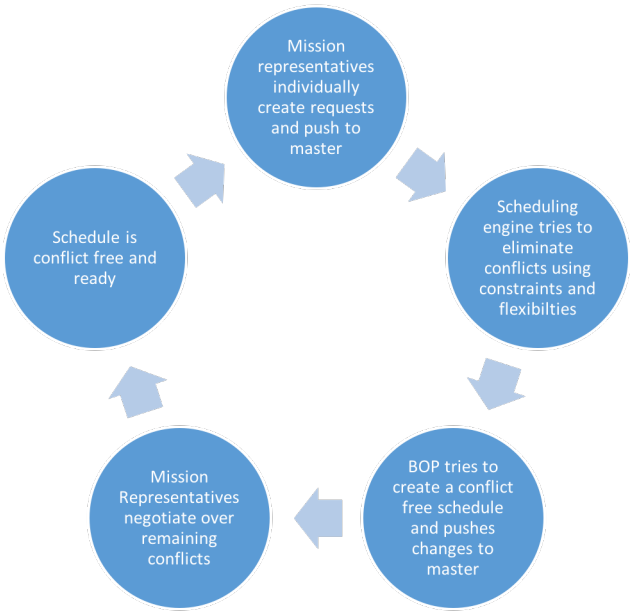


Fig. 3: Schematic diagram of the DSN mid-range scheduling process: for each week, the process starts at the top.

2. DSN Scheduling Process Overview

The DSN scheduling process (Johnston et al., 2014) operates on a rolling weekly basis (Fig. 3): as the deadline for a week approaches (roughly four months before the start of the week), mission scheduling representatives enter the requirements for that mission into the Service Scheduling Software (Johnston et al., 2012, 2012). Unlike other networks, many DSN user missions have changing requirements from week to week, reflecting mission events and phases, including a wide range of pre-planned science activities. Due to the long light travel time to many DSN spacecraft, spacecraft are sequenced with command loads that are generated many weeks ahead, and the DSN schedule is a critical input to this process.

Once all inputs for a week are in, they are integrated into a single schedule and the DSN Scheduling Engine (DSE, (Johnston et al., 2010)) is run to deconflict as much as possible, given any specified flexibilities in the input requirements from each mission. In practice, little flexibility is allowed, and the net oversubscription level means that many conflicts necessarily remain in the schedule.

Each requirement has a specified priority on a scale from 1 to 7. The default value is 7, nominal mission operations, and nearly all activities are assigned this priority level. Exceptions are made for elevated criticality events like launches, planetary landings and orbit insertions, and other high-risk mission events or unique major science opportunities, but in general these are rare. Note that priorities are on specific requirements, not on missions: there is no intrinsic priority distinction from one mission to another. Priority is used by the DSE to place higher priority activities on the schedule at their preferred times and antennas, and then to place lower priority activities where they least conflict with higher priority ones. However, given the high levels of oversubscription, many lower priority activities are placed in conflict with higher priority ones, since it is not permitted to drop them out of the schedule at this stage.

Once the scheduling engine has been run, and conflicts reduced as much as possible based on specified flexibilities, a human scheduler called “Builder of Proposal”, or BOP, starts to work on the schedule and makes further changes based on experience and background knowledge of each mission’s requirements. These changes include: deleting some activities, shortening tracks below their specified minimums, splitting tracks flagged as unsplittable and placing the (now shorter) segments into gaps in the schedule. This a time-consuming and labor-intensive process, requiring a great deal of familiarity with the entire DSN mission set and their typical requirement patterns. The BOP can generally

eliminate ~200 conflicts, but at the end there usually remain 10-20 conflicts. At the conclusion of the BOP phase, the week is released to the full set of mission scheduling representatives to negotiate the remaining conflicts and to make any adjustments to changes introduced by the BOP.

The second part of the interactive scheduling phase is when individual mission representatives collaboratively negotiate peer-to-peer, to resolve remaining conflicts and make additional changes (Carruth et al., 2010). In this process, one user will propose a set of changes, to which all affected users must concur before it becomes the new baseline. If any user disagrees with the changes, it falls on him or her to counter-propose an alternative (where just undoing a previous proposal is not allowed!). This process continues until the deadline is reached, at which point conflicts are either cleared or (rarely) waived, and the schedule is considered baselined and published. From the completion of the automated scheduling run to the baseline conflict-free schedule is typically 2-3 weeks. The overall duration of this process means that multiple weeks are being worked on in parallel.

3. Priority Considerations for DSN

As noted above, DSN currently uses a 7-level priority scheme strictly for categories of *events*: at the top are safety- and mission-critical activities, and at the bottom are normal science operations. Because nearly all activities (except essential maintenance) are considered "normal science" and thus at the lowest event priority level, the current priority scheme provides virtually no guidance for addressing oversubscription.

In terms of the current DSN scheduling process (Section 2), the greatest leverage for process improvement comes from the pre-BOP automated scheduling step: if oversubscription could be addressed *prior* to the BOP process, then both the BOP effort and the collaborative negotiation process phases could be drastically reduced. Missions would have their schedules baselined earlier, and could start work earlier to plan their onboard activities and generate their command loads. Prioritization could also play a role in later schedule changes, but these changes are of a much smaller magnitude. In the remainder of this paper, we focus entirely on the pre-BOP scheduling phase.

A variety of factors could be incorporated into a more fine-grained prioritization, including the following:

1. *Prime vs extended* missions: only about 25% of DSN mission users are still in their prime mission phase: the rest are in their extended missions (some have been flying for nearly 40 years). More than half of all requested time comes from extended missions. While prime vs extended could be used as a prioritization factor, would not help with addressing oversubscription, which would still be a problem even it there were no prime missions at all.
2. *NASA vs. non-NASA* missions: as DSN is a NASA asset, one option would be to give NASA missions priority for its use. However, high level agreements with partners provides for use by non-NASA missions on the same footing as NASA missions. As a result, this is not a factor that can help with oversubscription.
3. *Intra-mission priority tiers*: this concept calls for missions to divide their requested DSN time into priority tiers, rather than submitting all at the same event priority. This can provide explicit information about what each mission could possibly "do without" as being of lower priority. This information is implicit in the cuts that missions accept each week, accounting for the hundreds of hours of antenna time that is reduced by the BOP or negotiated away.
4. *Enforced reduced input levels*: this notion is based on the observation that missions ultimately accept reductions to deal with oversubscription, and so constraining their input levels to historically accepted output levels would be one way to enforce a request pool that would eliminate or drastically reduce oversubscription. For example, if mission X routinely states a requirement for 80 hours of tracking time, and then routinely accept 55 hours, their input could be constrained to 55 hours in the first place. This would add an additional check/enforce step to the process, but could shorten all the downstream steps. A drawback of this approach is that some mission requirements tend to vary from week to week and so a constant cut-off would be a problem for some missions.
5. *Time-dependent priority*: most DSN users require a time spread in their activities, to reflect the accumulation of scientific data and subsequent transmittal to Earth, and for regular measurements for navigation updates. Most also have a check for communication with Earth, such that if they have not been in touch for some configurable time, the spacecraft goes into "safemode". As a result, the time since last contact comes into play when considering the priority of each mission, so that no mission can be "starved" and drop out of the schedule, thus threatening spacecraft health and safety.

4. Experiments and Results

Experimental setup

To define a uniform basis for experiments, we used a 16-week period in 2012 consisting of just over 4,000 requirements for 31 missions. This particular dataset is only modestly oversubscribed, but reflects a realistic mix of typical requirement types. We used a modified version of the DSN Loading Analysis and Planning Software (LAPS) (Johnston et al., 2012) being developed for long-range planning and forecasting. This software allows for plugging in different algorithms for prioritization, thus making it easy to experiment. It can also be configured to drop requirements that can't be satisfied without conflicts.

As a baseline, we used a greedy algorithm that works as follows: for each priority tier from highest to lowest, order requirements by most constrained first, and schedule in their most preferred place (time/antenna). If there are no feasible places left, add the requirement into the unscheduled set. For our sample dataset, the overall total scheduled/requested is 88%. In this requirement sample, all missions are at the same event priority level, that of routine normal science priority.

From the perspective of any individual mission, it is not the total scheduled/requested that matters, but their own individual mission's level. In this baseline scenario, 4 of the 34 missions received 80% or less of their requested time, while one received less than half. So one of the questions we address is how to keep some missions from a proportionally greater impact, while satisfying all mission's requirements to the greatest degree possible.

To see the effect of the 'most constrained first' aspect of the baseline strategy, we removed that and scheduled all requirements at equal priority (breaking ties randomly). The net effect is as would be expected: some requirements with lots of flexibility consume places needed by more constrained requirements, and so more of the latter remain unscheduled. The overall total scheduled/requested drops to 82%, and 5 missions receive 80% or less of what they requested, with three receiving less than 50%.

We also looked at the impact of separating out prime missions from extended ones in the prioritization, assigning prime missions a higher priority. The three prime missions represented 12% of the total time requested, and when given a higher priority, they received virtually all that they requested (99.9%). However, there was a larger impact on the overall total scheduled time (reduced from 88% to 83%): of the extended missions, 5 received 80% or less of their requested time, and two receiving 50% or less. Therefore, the impact of prime vs. extended missions is not clear-cut, since including them at higher priority significantly drives down the total time scheduled, to the detriment of all.

Priority Tiers

From the observed behavior of mission users to accept less time than originally requested, it is clear that what is submitted by many users as *required* is not truly required: it represents a *desired* level of time allocation, and can be reduced as circumstances warrant to fit with everyone else in the same week. This is borne out by the BOP's strategy when working on a week: the first step is to cut virtually all missions back to "typical" levels and then to tweak and optimize the resulting schedule. In general, all users accept these cuts without complaint, and spend the negotiation period fine-tuning the resulting allocations, and attempting to horse-trade with other users to make incremental improvements. The fact that the required submission is, in fact, flexible, is not specified by users in their inputs to the scheduling process. We will return to this point later as it has a major impact on potential solutions.

Based on this observation, we considered how to modify the process if users did specify at least the *relative* priority of their own inputs. This suggested a tiered input approach, with users dividing their requirements up into tiers as illustrated in Fig. 4. We chose a simple scheme where levels 1 and 2 reflected elevated priority requirements, level 3 corresponded to normal priority, and levels 4 and 5 were desirable if possible. As an example, one mission had requirements on certain days of the week that were essential in order to upload commands to the spacecraft for the following week. For this mission, meeting those specific requirements was much more important that others in the same week, which could be more readily reduced or even occasionally dropped.

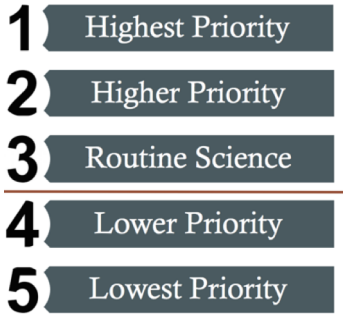


Fig. 4. A tiered priority scheme for user-specified requirement priority. Levels 1-3 are considered "must have", levels 4 and 5 are "desired if possible".

Generating an accurate dataset with this information is very difficult, so in the absence of real user inputs, we arbitrarily divided each mission's inputs into these 5 levels, with an even distribution of time across the levels. Each level was scheduled in priority order for all missions, with the results fixed when lower priority levels were considered (i.e., level 1 was scheduled for all missions, then level 2 added, and so on). The results showed that nearly all missions received 90-

100% of their level 1 and level 2 inputs, and 75% received 90-100% of the level 3 inputs. The levels that received less that half of the time requested were almost all from levels 4 and 5. However, the overall total time scheduled was still reduced to about 84%. It remains an open question whether that fact that more missions got more of their "highest priority" requests would balance out this reduced overall scheduling efficiency.

We explored some alternatives to the 5-level tiers, e.g. a 2-level tier with 80% of the time in a higher priority level, and the remaining 20% as lower priority. The results showed a slight improvement on the overall total scheduling efficiency, to 85.5%. On the other hand, the 5-level tiers provide more granular information as to how users place a relative value on the time they are receiving, and so could be the most useful in improving the automation process.

It is worth noting that while much more information about alternative requirements could be asked of users, it would place a significant additional workload on them to provide information that might not be used. For example, users could specify preferences for shrinking or dropping certain requirements, depending on which other requirements are satisfied in the schedule. Specifying these inputs could become complicated and time consuming, and would only be used if the dependency circumstances were realized. The notion of simply adding one additional relative priority field to a requirement would certainly be manageable, as well as being directly usable by the software, the BOP, and other schedulers during the negotiation phase.

Enforced Input Reduction

Another tactic to reduce the high level of oversubscription is to require users to reduce their inputs to a level that they have historically been shown to accept. We tried to simulate this by taking historical reduction results, and then arbitrarily cutting mission inputs to correspond to what each mission had found acceptable over a 6-month period around the time of our experiment 16-week time range. We only reduced the heaviest users (those requesting 40 or more hours of tracking time per week, a total of 14 missions). Reduction percentages varied over a significant range, with 7 missions receiving reductions over 20%, with the the largest reductions of about 35%. The results showed 26 of the 31 missions receiving over 90% of their requested time, with only one mission receiving less than 50%. The total input time was reduced by about 13% overall, and so the overall time scheduled was reduced accordingly.

Because our experimental reduction was arbitrary, it does not reflect the specifics of what each mission might consider essential. This would be very difficult to determine post facto to make the experiment more realistic. On the other hand, it would not be a large burden on users to ask them to fit their input requirements within an overall time cap. Furthermore, the option would remain to add back in additional requirements if it turned out that there remained opportunities to do so during the negotiation phase.

Squeaky Wheel Optimization

An optimization technique that has been used with good results on other oversubscribed scheduling problems is that of Squeaky Wheel Optimization (Joslin and Clements, 1999); see also (Barbulescu et al., 2006a, 2006b; Laura Barbulescu et al., 2004; L. Barbulescu et al., 2004; Kramer et al., 2007). In this approach, requirements are assigned an initial priority and then scheduled in priority order, then the priorities are adjusted until no further improvement in the objective function are observed. We applied this technique to the DSN scheduling problem in the following way. The objective we used was the overall total scheduled time for all missions. Note that this does not take into account that some missions might perform relatively poorly, even though the overall total scheduled time is better. This is an area for further work.

For the initial priority assignment, we tried different techniques, including: random; smallest requested time; and largest requested time. We found the best results using largest request time, likely because the larger users tend to dominate the objective function if they can be scheduled earlier and get a larger fraction of the time they request.

We adjusted the priority after each iteration by looking at which mission had the worst ratio of unscheduled to requested time, and swapping places with the next higher priority mission (Fig. 5). If the overall schedule did not improve, we restored the swap and tried instead the next worst, and so on. We terminated a run when there are no places left to swap without making the schedule worse.

For two missions, we found that unless they were excluded from the iteration process, they would invariably receive no time. Both were relatively small and so were left fixed as first and second priority in the list.

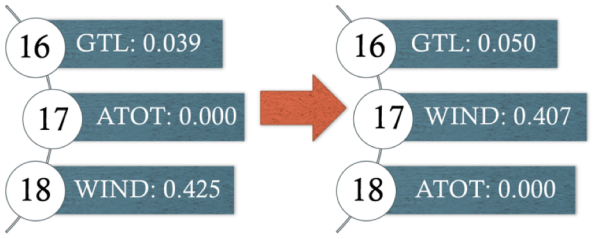


Fig. 5. Illustrative behavior of Squeaky Wheel Optimization in exchanging the priority of two missions (here ATOT and WIND) in order to achieve an improvement in the overall total time scheduled. Each mission is assessed using the ratio of unscheduled to requested time.

The results were encouraging, in that the best runs with SWO were able to schedule 91% of the total requested time, an improvement over the 88% found by greedy least constrained first.

5. Conclusions

The results of the experiments reported above are very encouraging in suggesting several promising directions to help address the DSN oversubscription problem:

1. Time reduction: use of this mechanism would require users to submit less "required" time in order to define a weekly scheduling pool that more nearly matches the antenna time available to be allocated. This requires a policy change, and leaves open the thorny question of how to set the appropriate restriction level per mission. Depending on the mission phase and the occurrence of various science and engineering events, there can be a significant variation from week to week, and so setting and policing this constraint could be burdensome. A further complication is the alignment of mission visibility periods at certain times during the year, which leads to some times being oversubscribed while others are unusable. Thus the target time to reduce to meet the objective of managing oversubscription is difficult to evaluate.

In spite of these considerations, some form of required time reduction is likely, and the most promising approach is to use the DSN long-range planning and forecasting software (Johnston et al., 2012) to set appropriate limits on how much time can be specified as "required" in the top priority tiers. This software will work from a long-range specification of requirements and can look ahead multiple years to assess oversubscription and contention due to overlapping critical events. Resolving contention far in advance could lead to "fair" and agreed input levels for the mid-range scheduling process.

2. Tiered relative priorities: this would allow users to explicitly specify how important are their different categories of requirements, knowledge which currently resides only in textual descriptive material or in the schedulers' heads. This approach could be readily combined with (1) time reduction, in that the total time in the top tiers could be restricted, while the lower priority requirements could be provided to take advantage of opportunities if they are available.

3. Squeaky Wheel Optimization with internal priorities: the use of a mission-level priority scheme does not lend itself to the DSN due to the high level of oversubscription and to the time variation in mission activity and corresponding requirements. However, the use of an *internal* and *dynamic* priority list does work well to improve the overall schedule efficiency while avoiding starvation of any mission due to being stuck in a low position on a static list. SWO could be combined with (2) tiered relative priorities to define an objective per mission to reflect the importance of meeting each mission's designated top priority activities, while attempting to fit in lower priority activities. It could also be combined with (1), time reduction to a fair level, in that the top tiers could be constrained to fit within agreed up (historical or forecast) limits on the available antenna time.

Currently, DSN is evaluating potential policy changes that would enable implementation of an approach like that described above. Further investigations will address how best to combine these approaches in a flexible but effective manner.

Acknowledgements: the authors are grateful for comments and suggestions from a wide range of participants in the DSN scheduling process, including mission personnel, scheduling team members, and the DSN scheduling office. We also acknowledge very useful suggestions from anonymous referees.

References

Barbulescu, L., Howe, A.E., Whitley, L.D., Roberts, M., 2006a. Understanding algorithm performance on an oversubscribed scheduling application. J. Artif. Intell. Res. 27, 577–615.

Barbulescu, L., Howe, A.E., Whitley, L.D., Roberts, M., 2004. Trading Places: How to Schedule More in a Multi-Resource Oversubscribed Scheduling Problem.

Barbulescu, L., Howe, A., Whitley, D., 2006b. AFSCN scheduling: How the problem and solution have evolved. Math. Comput. Model. 43, 1023–1037.

Barbulescu, L., Whitley, L.D., Howe, A.E., 2004. Leap before you look: An effective strategy in an oversubscribed scheduling problem.

Carruth, J., Johnston, M.D., Coffman, A., Wallace, M., Arroyo, B., Malhotra, S., 2010. A Collaborative Scheduling Environment for NASA's Deep Space Network. Presented at the SpaceOps 2010, Huntsville, AL.

Imbriale, W.A., 2003. Large Antennas of the Deep Space Network. Wiley.

Johnston, M.D., Tran, D., Arroyo, B., Call, J., Mercado, M., 2010. Request-Driven Schedule Automation for the Deep Space Network. Presented at the SpaceOps 2010, Huntsville, AL.

Johnston, M.D., Tran, D., Arroyo, B., Sorensen, S., Tay, P., Carruth, J., Coffman, A., Wallace, M., 2014. Automated Scheduling for NASA's Deep Space Network. AI Mag. 35, 7–25.

Johnston, M.D., Tran, D., Arroyo, B., Sorensen, S., Tay, P., Carruth, J., Coffman, A., Wallace, M., 2012. Automating Mid- and Long-Range Scheduling for NASA's Deep Space Network. Presented at the SpaceOps 2012, Stockholm, Sweden.

Joslin, D.E., Clements, D.P., 1999. Squeaky Wheel Optimization. J. AI Res. 10, 353–373.

Kramer, L.A., Barbulescu, L.V., Smith, S.F., 2007. Analyzing basic representation choices in oversubscribed scheduling problems. Presented at the 3rd Multidisciplinary International Conference on Scheduling: Theory and Application (MISTA-07), Paris, France.

Using Hierarchical Models for Requirement Analysis of Real World Problems in Automated Planning

Rosimarci Tonaco-Basbaum and Javier Martinez Silva and José Reinaldo Silva
Department of Mechatronic Engineering, University of São Paulo, Brazil
rosimarci@usp.br, javsilva@usp.br, reinaldo@usp.br

In the intelligent design field, the early phase of requirement analysis plays a fundamental role, especially when dealing with problems to which an analytic formal solutions is not applied. Automated planning appears in that category - particularly when the target are "real world" systems. Requirement analysis is exactly where the Engineering Knowledge embedded in the problem is explored to provide clues to the solution. A great effort has been made today in the area of Artificial Intelligence to define a reliable design process for automated planning that includes a Knowledge Engineering early phase. This paper intents to propose a requirements analysis formal procedure that starts by taking requirements for planning problems represented in UML and proceed to an analysis process based on Petri Nets. In fact a similar approach were insert in a tool called itSIMPLE (by one of the authors) using an old version of UML (1.4). As we will see in this proposal the process were not completely formal, even if practical. In the current work an unified Petri Net is fully complied with the ISO/IEC 15.909 standard replace the graph analysis. Using this net we can introduce a formal property analysis including invariant analysis and exploring abstraction from a hierarchical extension of classic Petri Nets. Case Studies are presented, with classic problems from the manufacturing and petroleum industries, aiming to show the differences between the early proposal and the current proposal.

Introduction

Planning characterizes a specific type of design problem where the purpose is to find an admissible sequence of actions to bring the system from a given initial state to a target final state. Current approaches in the literature aim to improve the performance of automated planners by trying to optimize the search algorithms and the general solution (Edelkamp and Jabbar 2006). In addition, most of existing work on this direction driven to synthesized and artificial problems (closed problems that have limited set of actions) as a proof of concept for the proposed algorithms. Due to the extensive development in this area some authors started to apply planning techniques on real world problems as well - like logistic problems - with a considerable higher number of variables, where domain independent approaches are computationally prohibitive (Vaquero et al. 2012). Such alternative approach could bring some light and/or good results to challenge problems and could also gave some feedback to solve a fully automated, domain-independent problem.

That said, it is clear that the automated planning area carries an uncertainty problem:

- the study made until today is historically connected to search solution methods to automatic planning problems in a domain independent approach. This, solutions could be inserted in intelligent automated devices, especially robots, or an autonomous machine system.
- on the other hand, formal techniques, especially those which are domain independent, lead to important techniques that can be applied in several demanding fields, like logistics, diagnostic systems, navigation, space robots, satellite systems, etc. However, this demand are sensible to the adequacy of formal techniques to the specific knowledge surrounding real problems. This combination can provide good solutions, while inspires new domain independent solutions.

Indeed, complex domains are hard to deal with when no abstraction is present. In these domains, a hierarchical problem decomposition based on topological structure can lead to significantly better performance. Our preliminary running exercise (ROADEF 2005) is based on a synthesized domains and shows an impressive performance when hierarchical models are introduced in the design process.

In a standard planning domain such as Logistics, topological abstraction of the real world is part of the definition. In Logistics, several packages have to be transported from their initial location to various destinations. A Logistics problem has a map of cities connected by airline routes. Transportation inside cities can be done by truck (there is one truck in each city). Cities are abstracted, being treated as black boxes. Inside a city, a truck can go from any point to any destination at no cost (Botea, Muller, and Schaeffer 2003). However, in the real world, transportation within a city is a subproblem that can involve considerable costs. In this context, removing human expertise and automatically obtaining abstracted models of the real world is an important research

problem. And this kind of problem is one of the motivations for this work.

This paper intent to propose a requirements analysis formal procedure, based on hierarchical models, that starts by taking requirements for planning problems represented in UML 2.4 and proceed to an analysis process based on classic Petri Nets. In fact, a similar approach were insert in a knowledge based tool called itSIMPLE (Integrated Tool and Knowledge Interface to the Modeling of Planning Environments) using an old version of UML (1.4). We claim that the new approach presented here turn the design process more accurate and result in clear design discipline. Formal design requirements analysis could be done in unified Petri Nets, that follows ISO/IEC 15.909 standard. Once analyzed requirements could be translated to PDDL and then submitted to an automated planner. Fourteen different planners are used, selected among those of better performance to a diversified set of planning problems.

In section 2 we focus on the use of semi-formal and formal methods in the requirement analysis and the challenge of jumping from a potentially inconsistent set of requirements to a stable and consistent new set. Section 3 will show some aspects about the designing process in automated planning, followed by a brief description of itSIMPLE old method, and introduces itSIMPLE new design process. Finally, we present two running exercises: a manufacturing problem, adapted from automotive industry and a real problem of logistic in offshore petroleum exploitation. Both illustrate the importance of including specific domain knowledge in the process and the effectiveness of the proposed process.

Requirement Analysis: Semi-formal X Formal Methods

Regardless of how the final specification is made, the success of any project depends on a correct and complete requirements definition, and requirements specification is used as a reference to test and validate what comes out from design and implementation phases. This is an essential role of the requirements engineering process, which comprises the complete representation of system behavior, considering functional and non-functional requirements. Therefore, an inadequate identification of requirements is a major cause of system failures, and avoiding errors during this phase becomes a vital process in the project development cycle.

Many researchers and practitioners have used semi formal techniques to capture system requirements. Thus, the analysis and verification phases inherit requirements excessive flexibility that results from semi-formal methods. Such methods are generally characterized by their simplicity and flexibility, and for a lack of consistency assurance, while formal methods are characterized by a formal representation which is the support for a sound simulation and analysis process.

The literature proposes some techniques and methods that can be used in analysis and specification of requirements, but none of them guarantees a complete and consistent representation of these requirements. There are three fundamental problems in the analysis of requirements, often mentioned in recent research:

- The detection and analysis of inaccurate and incomplete requirements (Liu and Yen 1996);
- The detection of inconsistencies and proposition of methods to manage it;
- Creation of a systematic process that takes the requirements informally specified, but already proved consistent, and transform it in formal specification.

In addition to these challenges we can also add the requirements volatility. Important requirements in elicitation and analysis phase, may lose importance during the design process, may disappear or merge with others. Besides that, the early detection of emerging requirements during the analysis phase is a hard task. Another important challenge is to make sure there is a sound mapping between requirements and attributes of the system improving traceability and maintenance (Vaquero 2011).

Many researchers propose to convert semi formal requirements in a formal representation (Baresi and Pezze 2001b). Among the most widespread formal methods there is Petri nets, widely used in the representation and validation of requirements based on properties that enable the verification process. In this case, a feasible design discipline involves UML to capture semi formal requirements of the system which are translated into a hierarchical Petri net to perform de analysis and verification phases. Possible inconsistencies will be detected in this translation and must be fixed.

At the present, there are several approaches that combine UML and Petri nets and their extensions. (Zhao et al. 2004), discusses some technical transformation of graphs, which can be used to convert UML diagrams in Petri nets. There are other proposals which offers methods to build Petri nets representing the functional behavior of systems considering sequence diagrams, activity diagrams, state diagrams, use case diagrams or activity diagrams (Denaro and Pezze 2004), (Guerra and Lara 2003).

Diagrams composing a UML model are interrelated, and their relationships may reflect the semantics of the diagrams. Therefore, to transform UML models in a Petri net it should be taken into account both the static structure and the dynamic structure of the diagrams, and so the relationships between them. In (Zhao et al. 2004). These relations were classified into three levels: the relationship between the same UML diagram in different contexts; the relationships between various diagrams of the same viewpoint; and the relationships between various diagrams with different viewpoints of the system. This third level describes the relationship between the diagrams with static viewpoints and with dynamic viewpoints. That will be clarified in the proposition of a design discipline for automated planning.

Design Process in Automated Planning

The interesting to solve real world problems using Automated Planning techniques is growing significantly in the last few years. In general, the major focus of the planning community is the pursuit of planners efficiency neglecting

the analysis aspects (Zimmerman and Kambhampati 2003; Upal 2005), and the need to deal with complex real world problems.

To conduct the planning of an activity it is necessary to determine all features of the system in which it is embedded. Some factors must be considered, for example the sub systems evolved, internal variables, correlations with other systems, constants and constraints. Such specification is called system modeling, and from it depends the success of the result obtained from the planning process. In this aspect several points becomes important, such as the proposed model complexity, and its accuracy from the original system.

In the design process, languages such as the traditional PDDL (McDermott 2003), or the UML (OMG 2009) are used. To help in the design and the requirement analysis phases, there are frameworks available such as itSIMPLE (Vaquero, Tonidandel, and Silva 2005) (Vaquero et al. 2007), that focus on the initial design process phases, such as specification and modeling. After design is concluded, it is necessary validate de model and to execute this task it is possible to use a formal representation like Petri nets.

To design a real life systems in the context of this work, there are two key challenges: 1) create a design discipline for modeling real life systems, using UML as the representation language; 2) translate and synthesis of the UML diagrams in a unique hierarchical Petri net (from GHENeSys, that is a class of high level Petri nets) (Miralles 2012), which will be analyzed in order to obtain information that can anticipate problems in the model and help in the design phase to generate suitable plans.

The general purpose of this paper is to propose a design process for automated planning systems, that is composed of two layers: 1) where independent domain methods are applied; and 2) using the specific knowledge and requirements analysis applying hierarchical Petri nets to increase the quality of planning applications in Artificial Intelligence. The challenge here were to discover where to insert the specific knowledge and how to include this in the design process since we are working with three classes of problems where the specific knowledge level increases from benchmark to real world problems. The classes are: benchmarks, intermediates (like ROADEFs (Perez et al. 2006)) and real world problems (like Petrobras problem presented in the International Competition of Knowledge Engineering for Planning and Scheduling (ICKEPS) 2012) (Vaquero et al. 2012).

In the first level the purpose is to design the model using UML (Class Diagram, Behavioral Statechart Diagram and Object Diagram) to model the hierarchical aspects of real world problems. From that model to suggest a formalization of this structure based on Hierarchical Petri nets. The properties will serve to analyze the similarities, repetitive cycles, invariants and other properties between the models.

In the second level the specific knowledge can be included as dynamic relationships and actions properties, that will be inserted in UML diagrams (using OCL) and translated in a Petri net (therefore having a format compatible with the previous phase). The structure receives the dependent-domain knowledge, to apply the analysis techniques of hierarchical Petri nets. These techniques are particularly sensitive when applied in real world problems, requiring a different approach from the academic applications. Real world problems must follow a very disciplined design process, grounded in Knowledge Engineering, whose initial stage is composed by elicitation and requirement analysis. Such design process is the study focus of many researchers in Automated Planning area (McCluskey et al. 2003).

Planning applications has two different classes of requirements: domain requirements and problem requirements (Vaquero, Tonidandel, and Silva 2005). In this paper we propose an evolution of this assumption, based on (Vaquero, Beck J C, and Silva 2013). Our proposal aims to divide the design process in two aspects: 1) the work domain aspects, where all the essencial characteristics are considered, (such as: name, constraints, operations, general actions and environment descriptions that are critical to the system); and 2) the planning problem aspects, where the initial state, goal state and set of objects that comprises the problem instance, as shown in figure 1. Based on this independence hypothesis the design process is performed.

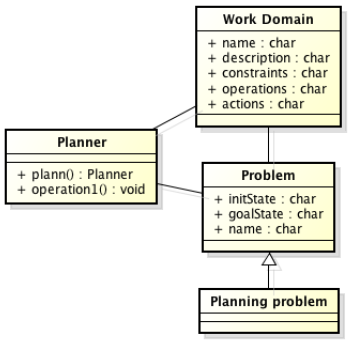


Figure 1: General schema for planning environment.

To show the efficiency and applicability of the proposal presented in this article lets recall the itSIMPLE old design process.Then, the new design process will be presented and finally a running exercise that will illustrate the new design process presented in this paper.

itSIMPLE Old Design Process

In this section it will be presented the itSIMPLE design method proposed in (Vaquero et al. 2005), in order of highlight some improvements we propose to achieve a new analysis method using Hierarchical Petri net. In (Vaquero et al. 2005) the basic process to model planning applications would start from requirements based on Use Case Diagram, Class Diagram, Statechart Diagram, Activity Diagram and Object Diagrams. Such approach became very popular in AI Planning community and won the ICKEPS [1] competition in

[1] ICKEPS (Int. Competition on Knowledge Engineering for Planning and Scheduling), is a competition organized in the scope of ICAPS (Int. Conference in Artificial Planning and Scheduling). itSIMPLE won 2009 competition and was directly involved in 2012 organization and could not compete. Since 2014 the system is being rebuilt.

2009.

According to (Vaquero et al. 2005), the Class diagram can be used as a representation of the planning domain static structure and concepts showing existing entities, their relationships, their features, methods (actions) and constraints. Classes and objects are the first and most important concepts in object-oriented modeling, and stand by entities that make sense in the application context. In UML.P (Vaquero et al. 2005), a general structure composed by an Agent class and a domain Environment class is proposed in order to organize the model. Dynamic agents change the arrangement of objects which compose the environment in order to find a path from an initial state to a goal state. Every entity that acts over the domain is a specialization of the Agent and all the others classes will be associated with the environment.

The Statechart diagram was used to define pre and post conditions. This diagram is very useful to represent entities that perform dynamic behavior. Usually, all actions (methods) defined in the class diagram are better specified in this diagram. Any class in Class diagram has its own Statechart diagram, specially those that perform actions. Each diagram does not intend to specify all changes caused by an action, instead, it shows only changes that it causes in an object of the Statechart diagram's class. Constraints in the Class diagram and all the pre and post conditions on the Statechart diagram are specified using constraint language OCL.

The initial problem statement in a planning domain is characterized by a situation where only two points are known: the initial and goal state, and a set of admissible actions. The diagram used to describe these states is called Object diagram or Snapshots (D Souza and Wills 1999). In fact, a planning problem is compose by two Object Diagrams, one describing an initial state and another describing a partial or entire goal state. Additional constraints intrinsically related to the problem can be specified by using OCL (Object Constraint Language), such as limitation on domain variable, rule and others. To illustrate the entire process, figure 2 it will present the itSIMPLE classical design process.

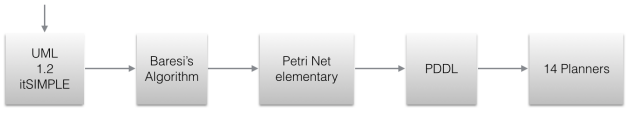


Figure 2: itSIMPLE classical design process.

DynA Design Process

Our main proposal is to substitute former itSIMPLE approach by a new one, which is more scalable - introducing a hierarchical approach - more disciplined - with a more rigorous definition of domain, where requirements are based on a minimum set of UML diagrams. Thus, this is a method directed to requirements analysis of dynamic systems ou just dynamic analysis (which will be based on Petri Nets).

Before start with the new design process we are proposing for itSIMPLE, it is important to clarify the basic concepts of the former itSIMPLE. In this work the UML diagrams chosen to be part of the design process has the same meaning they had in the original proposal (Vaquero et al. 2005). The major difference between this proposal and the old one is the introduction of a more abstract hierarchical structure to design the model, the introduction of a minimum set of UML diagrams to compose a sound representation of work domain and planning problem - the basic elements for AI planning.

After some running exercises, we discover the necessity to migrate for a more recent version of UML. The original itSIMPLE framework - even in its 4.0 release - uses UML 1.4, while our proposal introduces a hierarchical abstract structures to represent models which are not supported in UML 1.4. Considering that, in this project we upgraded the UML version to 2.4 and face all conceptual consequences, including convergence to a model driven approach.

The minimal set of UML diagrams comprises four diagrams, as will be shown. From the class of structural diagrams we take the Package diagram, the Class diagram and the Object diagram. The Package diagram shows the packages and their relationships. The Class diagram is a static structure of the system at the level of its classifiers (classes, interfaces, etc.). This diagram is able to represent some of the system classifiers, subsystems and components, the different relationships between them, their attributes, operations and constraints (in OCL). In the class of behavioral diagrams we took only the Behavioral State Machine diagram. The State diagram is used to model the discrete system behavior as a whole, by the finite state transitions. As proposed in (Vaquero et al. 2005), the problem statement is a planning domain characterized by a situation where only two states are known: the initial and goal state. To represent this it is used the Object Diagram or Snapshots (D Souza and Wills 1999).

In this section it will be presented the design process to model state transition systems (that is a class of real world systems) in the automated planning scope.

Precondition: the system must have distinct components that are inter-related.

- Define components and relationships;
- From previous step, design the Class diagram;
- If necessary, divide the Class diagram into modules;
- In the Class diagram, identify the dynamic objects of the system;
- Use OCL to define constraints;
- Design the Behavioral Statechart diagram;
- In the higher level, use composite states to design the system; and
- Design the internal states until the lower level of the system.
- Use Object diagram to define inicial and goal states.

Once a UML model is done, and requirements (and constraints) were inserted in the diagrams, the next step is to analyze the requirements and validate the model - which is now made explicitly. To perform this phase it will be used a Petri nets formalism. For this purpose it was developed a

translation algorithm, to convert the Behavioral Statechart Diagram into a Hierarchical Petri net.

Once the model was validated using Petri nets, it can be translated in some specification language that planners can interpret in order to generate a plan automatically. Since the focus here is the use of hierarchical methods it makes sense use some planner (and language) that can support this features. Originally, itSIMPLE uses PDDL (and all PDDL-driven planners) to generate a plan, and it is well known the incapacity of PDDL to process hierarchical models (McCluskey 2003). Because of this limitation we had to test our proposal outside the original itSIMPLE, using a classic UML editor (capable to use all resources UML 2.4 can offer, specially in the Behavioral Statechart diagram). The translation process (both to classic Petri nets, and to HTN) is made by a direct algorithm (which does not belong to the original itSIMPLE framework). The resulting model will be submitted to JSHOP2 planner (Ilghami 2006) which can use all knowledge engineering features included in the new design approach, including hierarchy.

This new approach can be applied to larger and more complex systems. A second running exercise (Petrobras Domain) is used to show that such approach is more scalable and could be applied to real problemas with symmetric properties but with a larger number of states (that will be shown later on). Figure 3 shows the design method proposed in this paper.

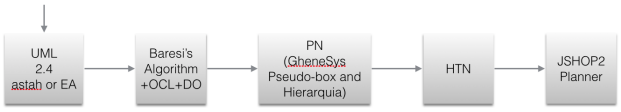


Figure 3: The new design process.

Translation algorithm: from UML to Petri nets

As expected, the practical application of AI planning techniques to real problemas lead to the need to include more abstract modeling based on hierarchy, as well as a model driven approach. This finding lead to the use of UML 2.4, especially the Behavioral Statechart diagram, that can represents the whole system in a hierarchical way. If the initial model is hierarchical, make it all sense to validate it using a formal technique also hierarchical. Initially we consider to use High level Petri nets to perform the requirement analysis and validate the model, however it became clear that the modeling process could fit better in an abstract process based on hierarchy, and thus an extension of the basic Place/Transition or High Level models. We choose Hierarchical Petri nets as a validation tool, because the hierarchical structure designed in the Behavioral Statechart diagram fits perfectly in the concept of this kind of Petri net. The remaining problem is then how to translate from UML diagrams to Hierarchical Petri Nets without any semantic loss.

Our first sound translation algorithm was based on the classic Baresi's algorithm (Baresi and Pezze 2001a). Some improvements had to be included in order to consider the Behavioral Statechart diagram and the OCL constraints present in the diagram. These constraints are essential to ensure a more faithful analysis process. That said, it will be presented the translation algorithm.

In this part of the algorithm, please consider only the Behavioral State Chart Diagram:

- The states of the diagram are modeled as places on Petri net while transitions in the diagram are modeled as transitions on the Petri nets.
- If the states of the diagram have constraints formulated in OCL, representing preconditions, they are modeled as places representing the state to which they belong; and if the states in the diagram have constraints that represents a post-condition, then it will be modeled as a place to represent the respective state.
- relations between states and transitions in the diagram are modeled as arcs between the corresponding places and transitions in the Petri net.
- The composite (or super states) states in the diagram will be modeled as macro elements in the Petri net and then refined until they reach the lower level of the system.

In the last part of the algorithm, please consider only the Object Diagram:

- The instantiated objects in the diagram will indicate the multiplicity of the process in the Petri Net.

After the net is translated the analysis and validation phases are made using the framework GhENeSys [2]. In this system, the hierarchical approach is based on homogeneous borders composed either from places or transitions with the requirement that there is only one input and one output element. Such requirement can improve the performance of property analysis using hierarchy (Miralles 2012).

Next section will show a running exercises (based on real applications) made to test the proposed design discipline. Petrobras domain is based on a real demand (the problem was reduced to fit space in this work) and refers to a demand to control the supply chain in a offshore facility to produce petroleum. Even the reduced problem would not be computationally prohibitive using the original approach. Thus, more than a comparison, the performance of the proposed discipline show an enhanced capacity to scale problems.

Running Exercise

Petrobras 2012 - Ship Operations on Petroleum Ports and Platforms

The general problem to be solved is based on the transportation and delivery of a list of requested cargo to different locations considering a number of constraints and elements such as available ports, platforms, vessel capacity, weights of cargo items, fuel consumption, available refueling stations in the ocean, different duration of operations, and costs (Vaquero et al. 2012). Given a set of cargo items, the problem is to find a feasible plan that guarantees their delivery

[2]GHENeSys (General Hierarchical Enhanced Net System) was a first attempt from our Lab to produce a Petri Net modeling environment strictly in the ISO/IEC 15.909 standard. This system has some extensions, including hierarchy and time Nets.

while respecting the constraints and requirements of the ship capacities. The objective is to minimize the total amount of fuel used, the size of waiting queues in ports, the number of ships used, the make span of the schedule and the docking cost. Next figures will show the UML model to represent this problem.

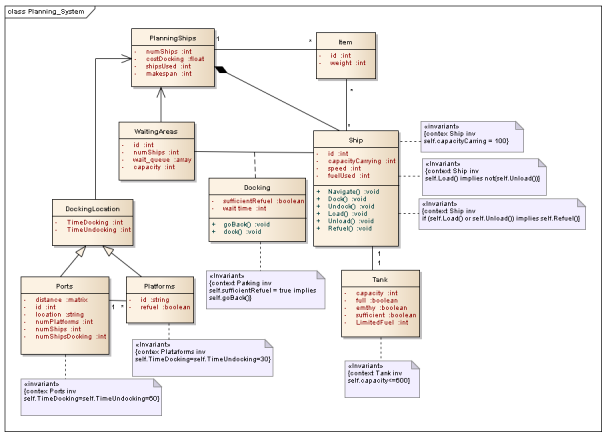


Figure 4: Class Diagram for Petrobras Domain.

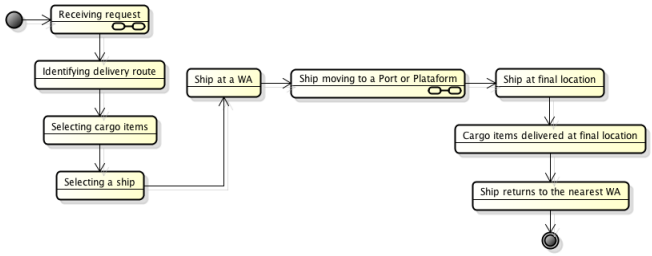


Figure 5: Behavioral Statechart Diagram for Petrobras Domain.

To complete de UML model was modeled the Object Diagrams for initial and goal states, respectively (they were omitted to fit space in this work). In the initial state the ships are docked all in Santos waiting area and the goal is to delivery cargo itens in one Santos platform and two Rio de Janeiro platforms.

According to the translation algorithm outlined in the previous section, the Statechart diagrams were transformed into a GHENeSys net (Foyo 2009). Figures 7 and 8 show the general view of Petrobras domain and the ships operations, respectively. The model was generated using the GHENeSys environment (Olivera Salmon et al. 2011).

The verification process was made using place invariants to validate properties of the system. The invariants are used both in the representation and verification of system requirements as presented in (Olivera Salmon, Del Foyo, and R. 2014).

To verify the accuracy of the model we first compute the invariants, thereafter we verify that the sets of places of

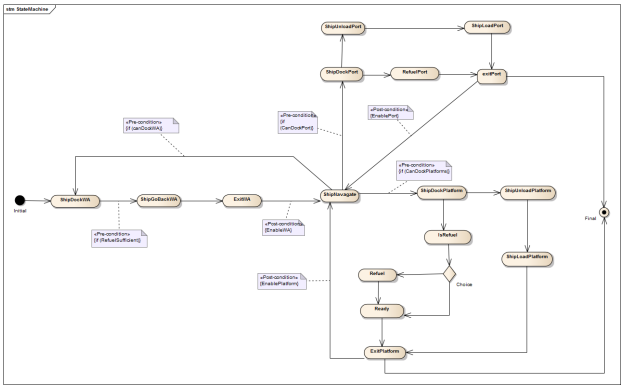


Figure 6: Behavioral Statechart Diagram for the ships in Petrobras Domain.

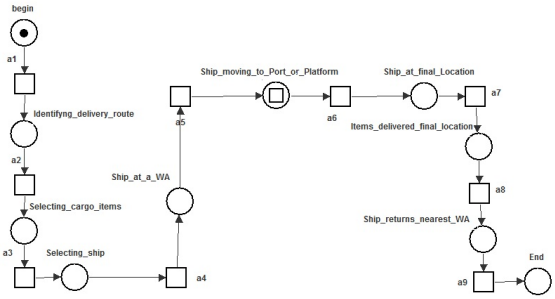


Figure 7: Hierarchical Petri net for Petrobras Domain.

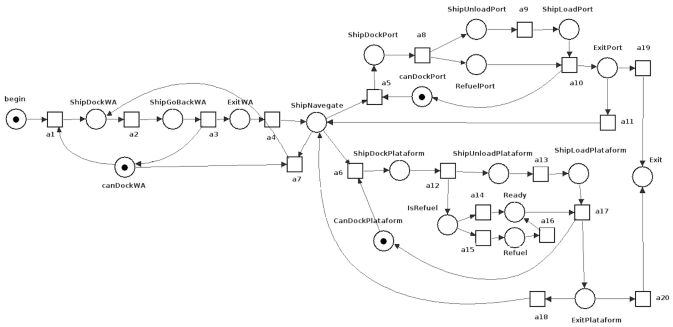


Figure 8: Hierarchical Petri net for ships operations in Petrobras Domain.

each inequality belongs to some vector in the solution set of Petri net place invariants. The invariants were calculated by GHENeSys system, which provides a set of invariants that represent the basic solution i.e. the generator of all possible invariants (Olivera Salmon et al. 2011). Thus, any invariant of the net can be obtained as a linear combination of the vectors of the generator sets.

Using the invariants set calculated and generated by GHENeSys, we obtained the invariants shown in the previous figures. These set of invariants demonstrate that the

model are adequate to represent the requirements defined in (Vaquero et al. 2012). Thus, it is possible to verify that the system has the desired requirements. The invariants places that we want to verify.

According to the design process proposed in previous sections the next step is to translate the UML/Petri Net model to HTN language - the language JSHOP2 accept to run the model and generate the plan. We tested the Petrobras domain with different planning problems, in this paper it will be presented one of this where problem instance were defined with 3 ships and 6 platforms (2 in Santos port and 4 in Rio de Janeiro port). To facilitate the understanding of the generated plan it was translated into 3 Statechart Diagrams (one for each ship). Figures 10, 11 and 12 will show the plan for ship 1, ship 2 and ship 3, respectively.

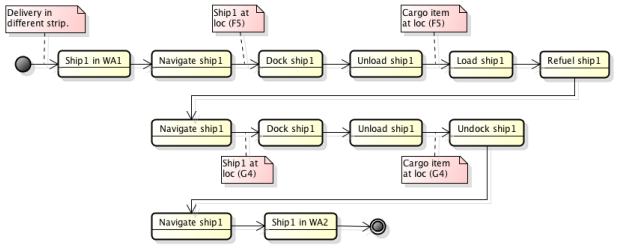


Figure 9: Statechart diagram for ship 1 action plan.

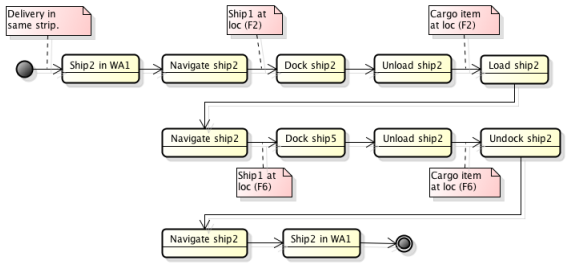


Figure 10: Statechart diagram for ship 2 action plan.

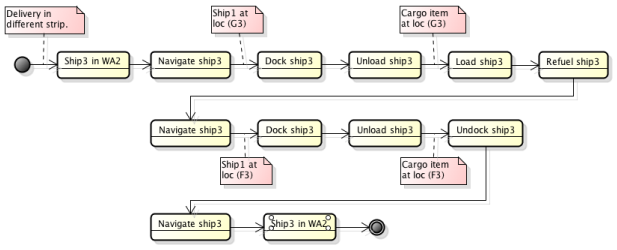


Figure 11: Statechart diagram for ship 3 action plan.

The states waas previously defined in the domain (and were represented in the diagrams) and the text box represents the auxiliary functions defined in the HTN code to support the main HTN function. As we can see in figures 12 and 16 (that represents the Behavioral Statechart diagram and Hierarchical Petri net for the ships, respectively), the ship must be at first in some waiting area. That is what happens with every ship. Another requirement is: the ship have always to perform an unload action before start the load process. This is part of the requirements defined in Petrobras domain documentation that we can verify using the design process present in this paper.

Conclusion

The first conclusion we can derive from this work is about the efficiency of the combined use of UML and Petri nets to capture and analyze requirements in challenge real problems related to planning, which keeps UML in a good position as a standard language. Such combination could be used successfully in the design of real problems that demand the use of AI Planning techniques. However such approach turn to be prohibitive to large problems, which make it attractive to use abstraction and hierarchy both in UML and Petri nets. That means more than a representation enhancement but a significant change in the design discipline.

During the development of this work, we realized that the Petri net generated by the original version of itSIMPLE missed some details. itSIMPLE consider just the old version of State Machine Diagram and this is not enough to derive a detailed Petri net. Another weakness of itSIMPLE is how the planning application is modeled. There is no design process to guide the user and this can lead the to some mistakes in the modeling, confusing the work domain with the planning problem. Our proposal is to separate them and this approach offers different viewpoints that complement the information needed to generate the Petri net.

After some running exercises, we observe that large real world problems can benefit from hierarchical structures. The most complex domains are hard to deal with when no abstraction is present. In these domains, a hierarchical problem decomposition based on topological structure can lead to a significantly better performance. Our preliminary running exercises using these domains as a testbed has already shown an impressive potential to use hierarchical models in the design process.

This work showed the need to restructure the framework itSIMPLE starting with the update of the UML version. This discover was made after several running exercises that were performed aiming refine the first proposal presented in (Tonaco-Basbaum, Vaquero, and Silva 2013). Another relevant discovery was the perfect adequacy of hierarchical models to represent real world problems. With these three main findings we justify the need of a new itSIMPLE version encompassing methods presented in this work. The main changes was made outside itSIMPLE to test the proposal, both running exercises presented in this paper were previously solved using classical version of itSIMPLE but only in the last (Petrobras domain), we can ”compare” the results, even our problem scope being greater than the scope presented in (Vaquero et al. 2012).

In this paper, we presented a different proposal for Petri nets in automated planning, that uses the Petri net in Knowledge Engineering to improve the design process. The objec-

tive was to create a better and disciplined way to model planning applications using UML and Petri nets in the context of itSIMPLE framework, trying fix and improve the design and validation processes.

References

Baresi, L., and Pezze, M. 2001a. Improving uml with petri nets, electronic notes in theoretical computer science 44.

Baresi, L., and Pezze, M. 2001b. On formalizing UML with high-level petri nets. *Concurrent object-oriented programming and petri nets: advances in petri nets*.

Botea, A.; Muller, M.; and Schaeffer, J. 2003. Extending pddl for hierarchical planning and topological abstraction. *ICAPS 2003*.

D Souza, D. F., and Wills, A. 1999. *Objects, components, and frameworks with UML: the catalysis approach*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Denaro, G., and Pezze, M. 2004. Petri nets and software engineering. *Lectures on Concurrency and Petri Nets. Springer Berlin Heidelberg,*.

Edelkamp, S., and Jabbar, S. 2006. Action Planning for Directed Model Checking of Petri Nets. *Electronic Notes in Theoretical Computer Science* 149(2).

Foyo, P. M. G. 2009. Verificação formal de sistemas discretos distribuídos. *Tese (Doutorado) - Escola Politécnica da Universidade de São Paulo.*

Guerra, E., and Lara, J. 2003. A framework for the verification of uml models. examples using petri nets.

Ilghami, O. 2006. Documentation for jshop2. Technical report cs-tr-4694., Department of Computer Science. University of Maryland.

Liu, X., and Yen, J. 1996. An analytic framework for specifying and analyzing imprecise requirements. international conference of software engineering.

McCluskey, T. L.; Aler, R.; Borrajo, D.; Haslum, P.; Jarvis, P.; Refanidis, I.; and SCHOLZ. 2003. Knowledge Engineering for Planning Roadmap.

McCluskey, T. L. 2003. Pddl: A language with a purpose? In *ICAPS03, 13th International Conference on Automated Planning and Scheduling*.

McDermott, D. 2003. PDDL2.1 - The Art of the Possible? Commentary on Fox and Long. *Journal of Artificial Intelligence Research (JAIR)* 20.

Miralles, J. S. P. 2012. *GHENeSys , uma Rede Unificada e de Alto Nível*. Ph.D. Dissertation, São Paulo.

Nau, D.; Au, T.; Ilghami, O.; Kuter, U.; Murdock, J.; Wu, D.; and Yaman, F. 2003. Shop2: An htn planning system. *AI Access Foundation.*

Nguyen, A. 2005. Challenge ROADEF 2005 - Car Sequencing Problem.

Olivera Salmon, A.; Miralles, J. A.; Del Foyo, P.; and R., S. J. 2011. Towards a unified view of modelingand design with ghenesys. *Proceedings of COBEM2011*.

Olivera Salmon, A. Z.; Del Foyo, P.; and R., S. J. 2014. Verification of automated systems using invariants. *Anais do XX Congresso Brasileiro de Anais do XX Congresso Brasileiro de Anais do XX Congresso Brasileiro de Automação.*

OMG. 2009. OMG Unified Modeling Language TM (OMG UML), Superstructure.

OMG. 2011. *OMG Unified Modeling Language (TM) (OMG UML), Superstructure. Version 2.4.1*.

Perez, O.; Reines, F.; Olivares, J.; Vidal, L.; and Hervas, T. 2006. Planning process from a user perspective. In *Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS 2006) Workshop on Plan Analysis and Management. Cumbria, UK.*

Tonaco-Basbaum, R.; Vaquero, T.; and Silva, J. R. 2013. Requirement analysis method for real world systems in automated planning. *Knowledge Engineering for Planning and Scheduling (KEPS). The 23rd International Conference on Automated Planning and Scheduling. Rome, Italy*.

Upal, M. 2005. Learning to Improve Plan Quality. *Computational Intelligence* 21(4).

Vaquero, T.; Beck J C, McCluskey, T. L.; and Silva, J. R. 2013. Knowledge engineering for planning and scheduling: Tools and methods. *JAIR*.

Vaquero, T.; Tonidandel, F.; Barron, L.; and Silva, J. R. 2005. On the use of uml.p for modeling a real application as a planning problem. *American Association for Artificial Intelligence*.

Vaquero, T. S.; Romero, V.; Tonidandel, F.; and Silva, J. R. 2007. itSIMPLE2.0: An integrated Tool for Designing Planning Environments. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS 2007). Providence, Rhode Island, USA*.

Vaquero, T.; Costa, G.; Tonidandel, F.; Igreja, H.; Silva, J. R.; and C, B. J. 2012. Planning and Scheduling Ship Operations on Petroleum Ports and Platforms. *Proceedings of the Scheduling and Planning Applications Workshop*.

Vaquero, T.; Tonidandel, F.; and Silva, J. R. 2005. The itSIMPLE tool for Modelling Planning Domains. In *Proceedings of the First International Competition on Knowledge Engineering for AI Planning, Monterey, Califormia, USA*.

Vaquero, T. 2011. *Pós-design para Problemas de Planejamento Automático : uma abordagem combinando diagnóstico, realidade virtual e reutilização de rationales*. Tese de doutorado, Universidade de São Paulo.

Zhao, Y.; Fan, Y.; Bai, X.; Wang, Y.; Cai, H.; and Ding, W. 2004. Towards formal verification of uml dia- grams based on graph transformation, proceedings of the ieee international conference on e-commerce technology for dynamic e-business, ieee computer socity.

Zimmerman, T., and Kambhampati, S. 2003. Learning-assisted automated planning: looking back, taking stock, going forward. *AI Magazine* 24(2).