Research Workshop of the
Israel Science Foundation

Proceedings of the 9th International

Scheduling and Planning Applications Workshop (SPARK-15)

Edited By:

Steve Chien, Mark Giuliano, Riccardo Rasconi

Jerusalem, Israel — 8 June 2015

Organizing Committee

Steve Chien
NASA Jet Propulsion Laboratory, USA

Mark Giuliano
Space Telescope Science Institute, USA

Riccardo Rasconi
ISTC-CNR, Italy

Program Committee

Mark Boddy, Adventium Labs, USA
Luis Castillo, University of Granada, Spain
Steve Chien, NASA Jet Propulsion Laboratory, USA
Riccardo de Benedictis, ISTC-CNR, Italy
Minh Do, NASA Ames, USA
Simone Fratini, ESA-ESOC, Germany
Mark Giuliano, Space Telescope Science Institute, USA
Christophe Guettier, SAGEM, France
Patrik Haslum, NICTA, Australia
Russell Knight, NASA Jet Propulsion Laboratory, USA
Angelo Oddi, ISTC-CNR, Italy
Bryan O'Gorman, NASA Ames, USA
Nicola Policella, ESA-ESOC, Germany
Cedric Pralet, ONERA, France
Riccardo Rasconi, ISTC-CNR, Italy
Tiago Vaquero, University of Toronto, Canada
Ramiro Varela, University of Oviedo, Spain
Terry Zimmerman, University of Washington, USA

Foreword

Application domains that entail planning and scheduling (P&S) problems present a set of compelling challenges to the AI planning and scheduling community, 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'15 is the ninth 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'15 workshop and publication chairs for their support.

The SPARK’15 Organizers

Table of Contents

Planning and Scheduling Actions in a Computer-Aided Music Composition System

Dimitri Bouche and **Jean Bresson**
STMS: IRCAM-CNRS-UPMC
1, place Igor Stravinsky, Paris F-75004
{bouche,bresson}@ircam.fr

Abstract

This paper presents a scheduling model for computer music systems. We give an overview of planning and scheduling issues in computer-aided music creation and rendering, and propose strategies for executing actions and computations in music composition or performance contexts.

Introduction

It is well known that the notion of scheduling can imply different levels and complexity in planning tasks and sharing resources (Lawler et al. 1993). Minimizing the resources and optimizing the timing of a process requires a strategy to determine the best ordering of tasks, and every task or computing instruction may itself require a careful planning of operations. In this paper we highlight some specificities of the planning and scheduling processes involved in a computer music application.

Music is a prolific field for computer systems and domain-specific programming environments. Many of them have been developed to support composition and other interactive tasks related to music writing and performance (Dannenberg, Desain, and Honing 1997). Therefore, a variety of applications and computing paradigms exist within computer music environments, implying different perspectives and concerns regarding the notion of scheduling.

We consider a particular subset of computer music systems dedicated to *computer-aided composition* (Assayag 1998). These systems focus on the production and transformation of musical structures, which can be read as scores or rendered by audio players or synthesizers. In computer-aided composition systems the *planning* (generation and ordering of musical actions) and the *execution* (or "rendering") are usually two separate processes which operate sequentially. In this context real-time constraints only concern the execution phase. In the planning phase, musical data can be computed following simple best-effort strategies.

Other types of musical systems are more oriented towards interaction, and process events and audio streams in real-time during music performances (Puckette 1991). In these systems the musical rendering is the output of periodic computations driven by interruptions or callbacks from audio drivers or external systems, which results are produced in bounded and minimal time intervals. Usually in this case, preliminary planning is very basic and complex temporal scenarios can hardly be developed.

Between these two archetypal cases, a number of current projects and software are challenged by the joint management of real-time interaction and the planning of musical structures organised on the longer term (Echeveste et al. 2013; Agostini and Ghisi 2013; Bresson and Giavitto 2014).

In this paper we describe the characteristics and design of a scheduling engine for computer music systems conforming with both compositional applications (i.e. static and independent planning and execution processes) and dynamic/interactive situations (where planning operations occur continually and concurrently with the execution). We introduce a two-fold representation connecting the low-level sequence of actions and the higher-level musical structures involved in score editing and rendering. We successively describe the score planning and scheduling models, and show how they can be made dynamic, allowing planning operations to be part of the execution process.

Score Representations and Planning

The score is a central notion in music composition, considered both as a musical object and as a working environment for composers (see Figure 1). During the process of *rendering*, it is reduced to a sequence of timed actions (notes and other instructions). This process is performed mentally and naturally by musicians interpreting a score, but it has to be carefully designed in an computer rendering system. A planning algorithm (or *planner*) must translate the score into this sequence of actions, by mapping the musical data (pitch, dates etc.) to rendering primitives (functions producing sound from the data).

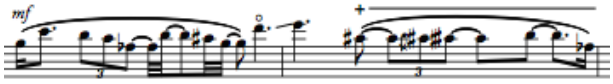


Figure 1: Example of a traditional score.

As contemporary music scores usually include varied kinds of musical data and actions (e.g. sounds, gesture notations, automations for controllers etc. – see Figure 2), the planning strategy must be designed with open and generic representations of both data and actions.

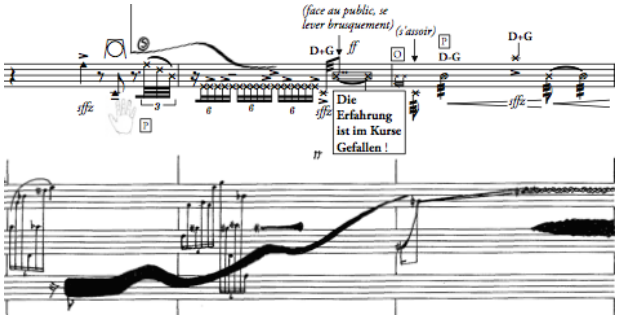


Figure 2: Heterogeneous musical data and controls in a score – extract from *Nachleben* by J. Blondeau (2014).

Planning Model

Let P (the plan) be a list of actions containing rendering operations (e.g. instructions to the audio system, transmission of MIDI[1] messages, or any kind of user-defined actions). Each element in P is an action $a : < t^a, id^a, f^a >$ where:

- t^a is a time-stamp,
- id^a is a unique identifier,
- f^a is a function to execute.

P is a low-level representation optimized for scheduling the rendering process. It must be updated at every modification of the score occurring in the system (for instance from the score editing front-end) and must remain sorted by increasing time-stamps. Three main operations are allowed:

- $schedule(P, a) \equiv$ inserts a at the adequate position in P,
- $unschedule(P, a) \equiv$ removes a from P,
- $reschedule(P, a, t') \equiv$ changes the position of a in P.

Hierarchical Representation

Composers or compositional processes running in a computer-aided composition environment manipulate musical objects with a high degree of structure and hierarchy. A note for instance, which can be considered the minimal specification unit of a musical score, requires at least two distinct actions to be rendered via a MIDI synthesizer: a *key-on*, and a *key-off* action. The *key-on* action must be scheduled at the actual time of the note, and the *key-off* at the time + duration of the note. Between these two actions, *continuous controllers* can also be transmitted to specify the variation of some parameters such as the volume, pitch bending, or other effects implemented in the synthesizer.

One musical object is therefore interpreted as a set of actions. Nevertheless, these actions need to be gathered together in some way in order to ease musical manipulations (a time modification of the note may require the whole set of corresponding actions to be rescheduled). Following the same principle, higher-level musical objects aggregate other objects (*e.g.* a chord gathers several simultaneous notes, a sequence gathers a number of chords under a common time referential) and a hierarchy of musical objects emerges. Hierarchical structures are therefore natural representations for time structures in music (Barbar, Desainte-Catherine, and Miniussi 1993).[2]

The planning model we propose is based on this hierarchical conception, where every musical object has a container and/or a set of children objects. It allows to maintain a correspondence between arbitrarily complex structures manipulated at the musical level and the linear sequence of timed-actions in P.

We consider a musical structure S: $< t^S, id^S, C^S >$ where:

- t^S is a time-stamp relative to the container of S,
- id^S is a unique hierarchical identifier,
- C^S is a list of children objects.

The hierarchical identifiers are constructed by appending a local unique identifier i to the container's id:

$$S_X \in C^{S_Y} \rightarrow id^{S_X} = id^{S_Y}.i$$

Figure 3 shows a graphical representation of a structure S with the following structure:

S = $< 0, 0, [S_1, S_2] >$
S_1 = $< t_1, 1, [S_{1.1}, S_{1.2}] >$
S_2 = $< t_2, 2, \emptyset >$
$S_{1.1}$ = $< t_{1.1}, 1.1, [S_{1.1.1}, S_{1.1.2}, S_{1.1.3}] >$
$S_{1.1.1}$ = $< 0, 1.1.1, \emptyset >$
...

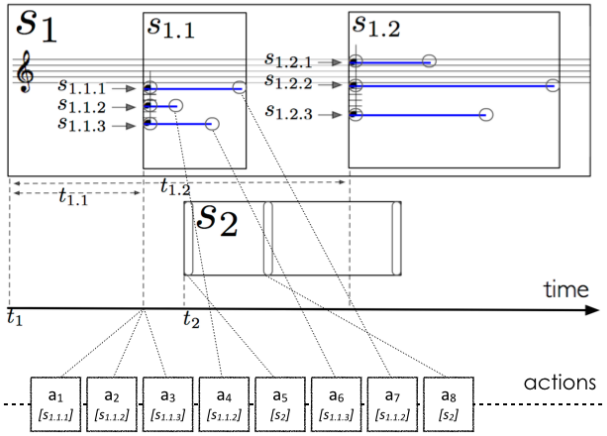


Figure 3: Example of a hierarchical musical structure and conversion to a plan (timed list of actions). S_1 is a sequence structured as a 3-levels hierarchy (sequence/chords/notes). S_2 is a sequence of 3 actions (e.g. parameter changes in a synthesizer's effect controller).

[1] MIDI (Musical Instrument Digital Interface) is a standard protocol and file format for transferring scores and instructions between musical software and digital instruments. MIDI *messages* can be seen as instructions sent to external synthesizers (play/stop note, set volume or effect parameters, etc.)

[2] Similar models have been proposed as well as in other domains, see for instance (Balaban and Murray 1998).

We call A^S is a list of actions a_i directly related to a structure S (not including the ones corresponding to its children). A^S must be specified by the system designer, depending on the type and value of S.

The planner retrieves the set of actions $\overline{A^S}$ corresponding to a hierarchical musical structure S through a recursive traversal of the children tree C^S:

$$P = \overline{A^S} = append(A^S, \bigcup_{S_i \in C^S} \overline{A^{S_i}}) \mid \overline{A^\emptyset} = \emptyset$$

In this process for each $a_j =< t^{a_j}, id^{a_j}, f^{a_j} > \in A^{S_i}$:

- The **functions** f^{a_j} depend on the type and value of S_i and are independent from the planning and scheduling model. Their determination is the main task of the system programmer using this model.
- The **time-stamps** t^{a_j} of the actions must be expressed as absolute time values in P. They can be partially derived from the hierarchy of the top-level structure S; for instance, the absolute time of $S_{1.1}$ in Figure 3 is $t^{S_1} + t^{S_{1.1}} = t_1 + t_{1.1}$.[3]
  The specific properties of S_i may also be taken into account to determine t^{a_j} for $a_j \in A^{S_i}$; for instance to a "note" structure N of duration d^N correspond two actions respectively at times t^N and $(t^N + d^N)$.
- The **action identifiers** id^{a_j} are automatically derived from id^{S_i}. For instance, three actions directly related to a structure with identifier $id^{S_i} = a.b$ will be assigned identifiers $id^{a_1} = a.b.1$, $id^{a_2} = a.b.2$ and $id^{a_3} = a.b.3$. This correspondence allows to maintain and retrieve information about the musical structure and hierarchy of S from the planning and scheduling processes.

The basic scheduling operations mentioned previously can then be applied to any musical structure S:

- $schedule(P, S) \equiv schedule(P, a)\ \forall\ a \in \overline{A^S}$
- $unschedule(P, S) \equiv unschedule(P, a)\ \forall\ a \in \overline{A^S}$
- $reschedule(P, S, t') \equiv reschedule(P, a, t'^a)\ \forall\ a \in \overline{A^S}$ with $t'^a = t^a + (t' - t^S)$

Score Rendering and Scheduling

A scheduler must execute the plan P derived from the score (or more exactly, from the musical structure represented in the score), executing all actions $a_i \in P$ on due time. This execution of P can be implemented using standard scheduling and optimization strategies.

Basic Execution model

We note $a_j = P[j]$ the action at position j in P ($j \in \mathbb{N}^+$) and we define a virtual "cursor" position j^P so that $a_{j^P} = P[j^P]$ is the next action in P that the scheduler will execute. At all time t, as P is sorted by increasing t^{a_i}, we will verify that:

$$j^P = min(i \in \mathbb{N}^+ \mid t^{a_i} \geq t).$$

The scheduler loop below checks periodically $t^{a_{j^P}}$ against the current clock time and executes actions from the last time interval at each iteration:

Algorithm 1 RENDER(P)

```
loop
    while t^{a_{j^P}} ≤ CLOCK_TIME( ) do
        CALL(f^a)
        j^P ← j^P + 1
    end while
    SLEEP(T)
end loop
```

Note that past actions (i.e. actions $a_i \mid t^{a_i} < t$ that are already executed at time t) are not removed from P, so that backward modifications and jumps of the cursor remain possible at any time.

With a period T in the order of a millisecond, this simple algorithm will support and render most of the standard musical scores. However, it may be challenged with scores including complex or high-rate sampled data, or if the actions involve computations with execution times that can not be neglected as compared to T. Some strategies for optimizing its execution are discussed further on.

Dealing with Long-term Executions

The system described so far is mostly suitable for dealing with instantaneous actions. In musical systems however (and in particular in the dynamic context we will consider in the next section), we must take into account actions that build or modify musical structures, which might involve arbitrarily complex computations. The execution time can then become a critical point for the correct rendering of the score.

The estimation of this execution time (or of the worst-case execution time – WCET) and its consideration in scheduling systems has been broadly discussed in the literature (Wilhelm et al. 2008). In our case we will consider that either the approximate execution time of an action is known and considered null, or this action falls into the category of "long-term" execution actions. In order to have the renderer loop performing as fast as possible, we will delegate the execution of long-term actions to a separate background process.

The execution strategy adopted for an action a is deduced from f^a: the action is instantly executable if $f^a \in A_{ZT}$, where A_{ZT} is a finite set of functions considered instantaneous in our system ($_{ZT}$ standing for "zero-time"). The macro CALL in Algorithm 1 is therefore defined as follows:

Algorithm 2 CALL(a)

```
if f^a ∈ A_ZT then
    EXECUTE(f^a)
else
    PROCESS(a)
end if
```

The call PROCESS(a) in Algorithm 2 sends the execution of the action a outside the scope of the scheduler: the ac-

[3] In the same example, notice for instance that the note's relative time-tags $t^{S_{1.i.j}} = 0$ since these objects are synchronized with their respective containers (the chords $S_{1.1}$ and $S_{1.2}$).

tion is wrapped into a *task* structure and stored in a FIFO queue. The FIFO queue is managed by a *thread-pool* (Kriemann 2004), which dispatches the tasks to *worker* threads: if a worker is available, the task is executed immediately, otherwise it remains in the queue until a worker thread becomes available. Most of the time, the length of the queue remains very small: accumulation happens only when many actions are scheduled in a same, very short time interval.

Tasks are therefore executed as non preemptible in separate threads (Henzinger, Horowitz, and Kirsch 2003) and do not impact the timing of the rendering process.

In a musical software, A_{ZT} contains for instance MIDI and external control message senders. In the next sections we will consider examples of more complex actions that are not in A_{ZT}.

Dynamic Planning and Scheduling

In its traditional form, a score is a static structure resulting from a compositional process. It is said static for it does not undergo any modification while being performed or rendered. In computer music systems however it is possible to imagine that an action triggered during the score rendering modifies its own structure (the initial plan). We will speak of an interactive, or *dynamic* score (Desainte-Catherine and Allombert 2005).

Dynamic Scores

In a dynamic score, actions or external events can redefine the plan during its own execution. In this case the scheduling and planning are concurrent processes. The planning is said "continual" (desJardins et al. 1999).

In our model, this dynamic characteristics amounts to allowing the functions f^{a_i} attached to the actions $a_i \in P$ to perform changes on the structure S, and thereby to request updates of P. In other words, actions can invoke the basic scheduling operations *schedule*, *unschedule* and *reschedule*, which respectively schedule new actions, remove and modify previously planned actions. The overall architecture of the system is sketched in Figure 4.

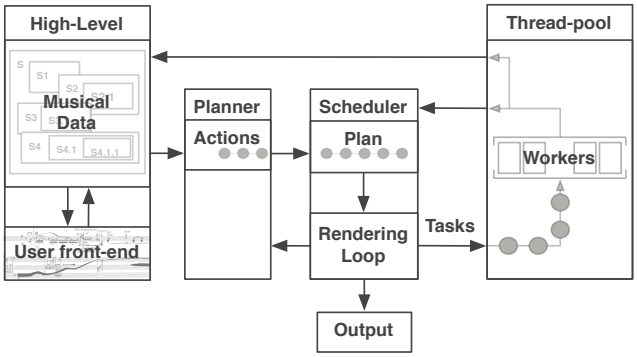


Figure 4: Dynamic architecture. Interactions between high-level structures, planning and scheduling.

The concurrent planning and scheduling operations in the dynamic model require the use of a lock mechanism to secure the concurrent read/write operation on P, as well as efficient sort strategies to be called at adequate moments when the plan is modified.

Extension of the Model

The dynamic score model allows user actions to modify the score S, leading the modification or scheduling of other actions. The result of an action execution if $f^a \notin A_{ZT}$ may therefore affect a musical structure in the middle of its rendering process, and new planning operations may be required immediately when this execution finishes. For this purpose the tasks sent to the *thread pool* are assigned an optional callback returning data to the high-level structure upon completion (see the *Thread-pool* to *High-Level* arrow on Figure 4).

A number of other situations are to be taken into account, such as actions being unscheduled while their associated task is running in the thread pool. This case can be handled if the scheduler stores a pointer to the task in the action structure at transferring it to the thread pool. Scheduling operations on the action can then easily change the state or abort the associated task.

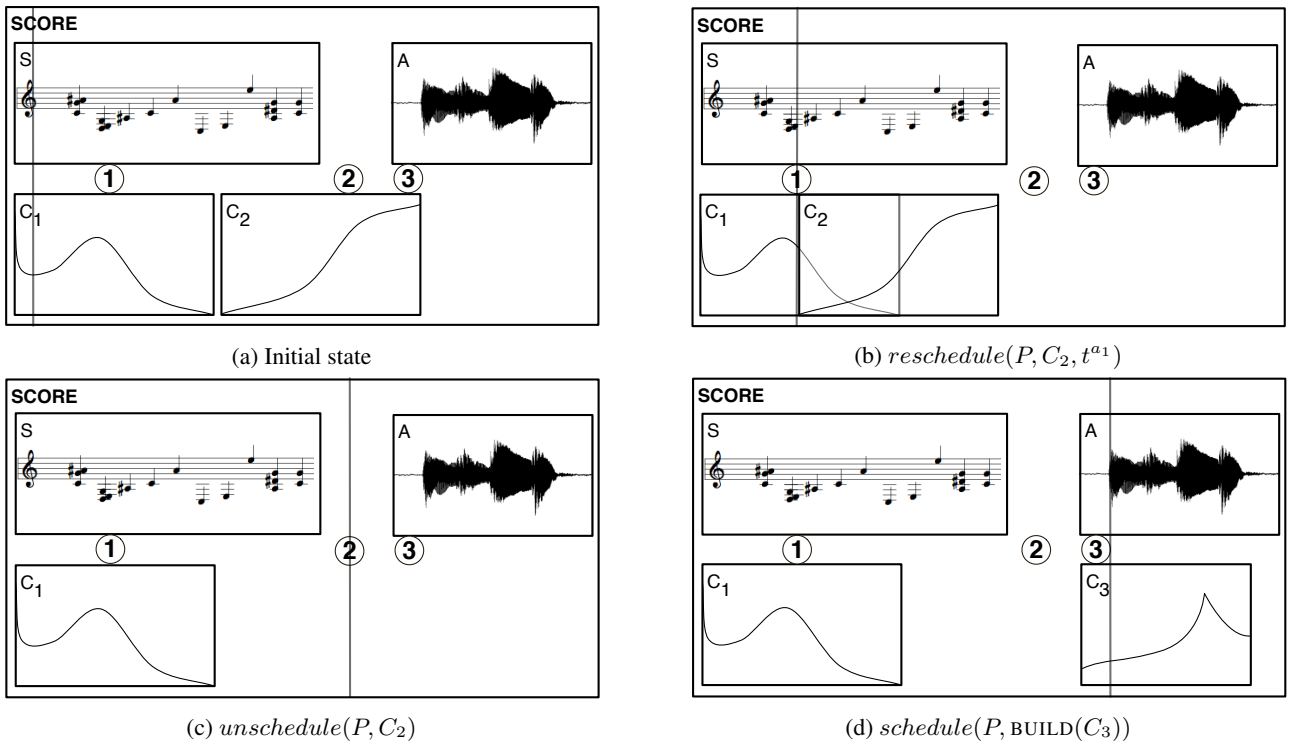
The dynamic model also makes it possible that musical structures be only partially known while the rendering process starts running, which requires considering the availability of the data before to perform actions. It is therefore useful in the score execution to separate functions and data. For this purpose we extend our definition of an action as: $< t^a, id^a, f^a, D^a >$ where D^a is a piece of data used by the function f^a attached to the action (in the general case D^a is a description of the musical structure S). We must then consider the case where the data D^{a_1} required by f^{a_1} and set by f^{a_2} is not available (e.g. if the computation of f^{a_2} does not finish on time). The availability of D^a can be checked by the scheduler prior to the creation of a task for an action a, and behaviours can be determined to react accordingly (e.g. the action a may be skipped, or sent to a thread that will sleep until D^a becomes available). The implementation of such behaviours, though not described here, is done by extending the definition of the action tuple.

Of course this architecture does not guarantee that computations will finish and make any data available on time. However, the score rendering process can run safely delegating actions to the thread pool and reacting to task termination (or non-termination) with predefined behaviours.

Example

In this section we propose a simple score example making use of the dynamic scheduling operations described in the previous sections. The score on Figure 5a contains the following objects:

- A hierarchical sequence of chords and notes (S) rendered as a sequence of MIDI messages,
- An audio file (A) rendered through a standard audio player,
- Two continuous controllers (C_1, C_2) sending values to external audio systems at a high rate (in the order of 100Hz),



(a) Initial state

(b) $reschedule(P, C_2, t^{a_1})$

(c) $unschedule(P, C_2)$

(d) $schedule(P, \text{BUILD}(C_3))$

Figure 5: Score example.

- Special events labelled ①, ② and ③ which respectively:
  - reschedule C_2 to the event's position,
  - unschedule (remove) C_2,
  - build and schedule a new controller (C_3).

We can see this score as a dynamic system controlling sound synthesizers and audio effects. We can imagine for instance that C_1 acts on a parameter of the synthesizer receiving the MIDI notes, and that C_2 and C_3 control audio effects applied to the general audio output. In order to make this dynamic score an interactive one we can also imagine that the events ①, ② and ③ appear dynamically during the execution of the score as the consequences of external events (e.g. performer inputs, sensors, etc.)

The functions f^{a_1}, f^{a_2} and f^{a_3} corresponding to the events ①, ② and ③ can be defined as:

- $f^{a_1} : reschedule(P, C_2, t^{a_1})$
  $\equiv reschedule(P, a_i, t^{a_1} + t^{a_i})$ for each $a_i \in \overline{A^{C_2}}$
- $f^{a_2} : unschedule(P, C_2)$
  $\equiv unschedule(P, a_i)$ for each $a_i \in \overline{A^{C_2}}$
- $f^{a_3} : schedule(P, \text{BUILD}(C_3))$
  $\equiv \text{BUILD}(C_3)$ then $schedule(P, a_i)$ for each $a_i \in \overline{A^{C_3}}$

The successive execution states after each event are displayed on Figure 5b, 5c and 5d.

The execution of a score like the one in this example remains continuous despite the dynamic plan modifications. Still, we can notice artefacts in the rendered output. In situations like ③ where an object is computed and immediately scheduled, we can observe a latency between the time a_3 is executed and the time C_3 is effectively scheduled. This latency seems hard to manage due to the unpredictability of the OS-controlled preemptive scheduling environment in which the system runs. It is important to precise however that the previous remark is due to an object being scheduled on the fly at the exact action time, and would not hold (or would not be detectable) if a reasonable delay is secured between the action time and the newly scheduled objects' date. Defining operations feasible on time is part of the responsibility of the composer (or of the musical system designer); nevertheless, the estimation and consideration of such delays and constraints in the action planning and execution could be an interesting direction for future works.

Conclusion and Perspectives

The scheduling engine we described implements dynamic features, including the execution of actions with non-deterministic behaviours or execution times, in a musical score renderer system (that is, the kernel of a score-based musical software). The hierarchical structure we propose permits manipulations at the musical level to be propagated at the low-level of the scheduler, and the scheduler actions to modify the top-level musical representations. At the difference of models such as the Hierarchical Task Network planning (Georgievski and Aiello 2015), the hierarchy here is considered at the level of the user (musical) representations and related planning operations, but remains out of the scope of task executions.

The straightforward approach described in this paper unveils planning and scheduling problematics in computer science applied to music. We are currently comparing it to a number of different approaches, for instance using tree-structured action lists and shorter-term planning.

The system is implemented in the OpenMusic environment (Bresson, Agon, and Assayag 2011). This environment has a wide user base in the contemporary/computer music community, which shall soon provide real-sized situations and use cases to assess its efficiency and reliability.

At a higher, musical level, our future work will concern the interfaces and tools proposed to the musicians that will allow them to take full advantage of the system, for instance for choosing or defining dynamic (re)scheduling actions, or specifying the behaviours of the scheduler regarding the availability of data.

Acknowledgments

This work is funded by the French National Research Agency project with reference ANR-13-JS02-0004. We thank Jérémie Garcia and Jean-Louis Giavitto for their proofreading and numerous suggestions.

References

Agostini, A., and Ghisi, D. 2013. Real-time computer-aided composition with bach. *Contemporary Music Review* 32(1):41–48.

Assayag, G. 1998. Computer Assisted Composition Today. In *1st symposium on music and computers*.

Balaban, M., and Murray, N. 1998. Interleaving Time and Structures. *Computer and Artificial Intelligence* 17(4).

Barbar, K.; Desainte-Catherine, M.; and Miniussi, A. 1993. The Semantics of Musical Hierarchies. *Computer Music Journal* 17(4).

Bresson, J.; Agon, C.; and Assayag, G. 2011. OpenMusic. Visual Programming Environment for Music Composition, Analysis and Research. In *ACM MultiMedia 2011 (OpenSource Software Competition)*.

Bresson, J., and Giavitto, J.-L. 2014. A reactive extension of the openmusic visual programming language. *Journal of Visual Languages and Computing* 4(25):363–375.

Dannenberg, R. B.; Desain, P.; and Honing, H. 1997. Programming Language Design for Music. In Roads, C.; Pope, S. T.; Piccialli, A.; and DePoli, G., eds., *Musical Signal Processing*. Swets and Zeitlinger.

Desainte-Catherine, M., and Allombert, A. 2005. Interactive scores: A model for specifying temporal relations between interactive and static events. *Journal of New Music Research* 34(4):361–374.

desJardins, M. E.; Durfee, E. H.; Charles L. Ortiz, J.; and Wolverton, M. J. 1999. A Survey of Research in Distributed, Continual Planning. *AI Magazine* 20(4).

Echeveste, J.; Cont, A.; Giavitto, J.-L.; and Jacquemard, F. 2013. Operational semantics of a domain specific language for real time musician-computer interaction. *Discrete Event Dynamic Systems* 4(23):343–383.

Georgievski, I., and Aiello, M. 2015. Htn planning: Overview, comparison, and beyond. In Elsevier., ed., *Artificial Intelligence*, volume 222, 124–156.

Henzinger, T. A.; Horowitz, B.; and Kirsch, C. M. 2003. Giotto: A time-triggered language for embedded programming. In *Proceedings of the IEEE*, volume 91, 84–99.

Kriemann, R. 2004. *Implementation and Usage of a Thread Pool based on POSIX Threads*. Max-Planck-Institue for Mathematics in the Sciences, Inselstr. 22-26, D-04103 Leipzig, Germany.

Lawler, E. L.; Lenstra, J. K.; Kan, A. H. R.; and Shmoys, D. B. 1993. Sequencing and scheduling: Algorithms and complexity. *Handbooks in OR & MS* 4(445-521).

Puckette, M. 1991. Combining Event and Signal Processing in the Max Graphical Programming Environment. *Computer Music Journal* 15(3).

Wilhelm, R.; Engblom, J.; Ermedahl, A.; Holsti, N.; Thesing, S.; Whalley, D.; Bernat, G.; Ferdinand, C.; Heckmann, R.; Mitra, T.; Mueller, F.; Puaut, I.; Puschner, P.; Staschulat, J.; and Stenström, P. 2008. The Worst-case Execution-time Problem - Overview of Methods and Survey of Tools. *ACM Transactions on Embedded Computing Systems* 7(3).

Compiling Away Uncertainty in Strong Temporal Planning with Uncontrollable Durations*

Andrea Micheli
FBK and University of Trento
Trento, 38123, Italy
amicheli@fbk.eu

Minh Do and **David E. Smith**
NASA Ames Research Center
Moffet Field, CA, 94035, USA
{minh.do, david.smith}@nasa.gov

Abstract

Real world temporal planning often involves dealing with uncertainty about the duration of actions. In this paper, we describe a sound-and-complete compilation technique for strong planning that reduces any planning instance with uncertainty in the duration of actions to a plain temporal planning problem without uncertainty.

We evaluate our technique by comparing it with a recently-presented technique for PDDL domains with temporal uncertainty. The experimental results demonstrate the practical applicability of our approach and show a complementary behavior with respect to the previous techniques. We also demonstrate the high expressiveness of the translation by applying it to a significant fragment of the ANML language.

1 Introduction

For most real world planning problems there is uncertainty about the duration of actions. For example, robots and rovers have transit times that are highly uncertain due to terrain, obstacle avoidance, and traction. There is also uncertainty in the duration of manipulation and communication tasks. For cars and trucks, transit times are uncertain due to traffic, road conditions and traffic signals. Any actions to be executed by humans are also likely to have uncertain durations. While there are domains in which the variability on the action duration is small enough that it can be ignored, there are many others where it can be significant, such as transit times during rush hour.

When there are no time constraints, no required concurrency, and plan duration is unimportant, this uncertainty can often be ignored to find a feasible plan. However, if there are exogenous events that affect action conditions, or time-constrained goals, the uncertainty on action durations must be considered.

In general, temporal conditional planning is very hard, particularly for actions with duration uncertainty (Younes and Simmons 2004; Mausam and Weld 2008; Beaudry, Kabanza, and Michaud 2010). In practice, most practical planners take one of two much simpler approaches:

1. Plan using expected action durations, and rely on runtime replanning and plan flexibility to deal with actions that take more or less time than expected.
2. Plan using worst case action durations.

The first of these approaches is risky – there is no guarantee that the plan will succeed or that runtime replanning can achieve the goals. The second approach, while generally more conservative, can also fail if there are time constraints or goals with lower bounds (e.g. an action should not be completed, or a goal should not be completed before some particular time). For space applications where any failure during plan execution is potentially very costly, having a plan that is guaranteed to execute successfully is often critical.

Recently, Cimatti, Micheli, and Roveri (2015) addressed these issues by extending PDDL2.1 to explicitly model duration range for actions, and devised a planner that soundly reasons to produce robust plans. In that work, the authors introduced the "Strong Planning Problem with Temporal Uncertainty" (SPPTU) as the problem of finding a sequence of action instances and fixed starting times, such that for every possible duration of each action in the plan, the plan is valid and leads to the goal. In this work, we address the same problem. However, to make it more relevant to real-world applications, we consider a much richer language for representing temporal planning domains. Specifically, we use and support: (i) a variable-value language; (ii) durative conditions over arbitrary sub-intervals of actions; (iii) effects at arbitrary time points during an action, (iv) exogenous events; (v) disjunctive conditions; and (vi) temporal constraints on goals. We address the SPPTU by automatically translating a planning instance with uncertainty on action durations into a plain temporal planning problem with controllable action durations. We exploit all the features in the planning language to cast the temporal uncertainty in action durations into discrete uncertainty over the problem variables. This compilation enables existing off-the-shelf techniques and tools for temporal planning to find strong plans for SPPTU.

We also present two sets of experimental evaluations of the compilation technique showing that it can be practically applied on expressive domains:

- On existing PDDL2.1-extended benchmarks: comparing against the techniques proposed in (Cimatti, Micheli, and Roveri 2015)
- On a set of problems modeled in ANML (Smith, Frank, and Cushing 2008), which enables modeling realistic tem-

*A shorter version of this paper appears in IJCAI 2015.

poral planning domains more naturally.

2 Related Work

Temporal uncertainty is a well-understood concept in scheduling and has been widely studied (Morris 2006; e Assis Santana and Williams 2012; Muise, Beck, and McIlraith 2013; Cimatti, Micheli, and Roveri 2014). The problem we address can be seen as a generalization of Strong Controllability for Temporal Problems (Vidal and Fargier 1999; Peintner, Venable, and Yorke-Smith 2007) to planning rather than scheduling. Dealing with planning is much harder because the actions (and thus the time points associated with them) in a plan are not known a-priori and must be searched for. Moreover causal relationships between actions are much more complex.

In temporal planning, duration uncertainty is a known challenge (Bresina et al. 2002), but few temporal planners address it explicitly. Some temporal planners (Frank and Jónsson 2003; Cesta et al. 2009) cope with this issue by generating *flexible* temporal plans: instead of fixing the execution time of each action, they return a (compactly represented) set of plans that must be scheduled at run-time by the plan executor. This approach cannot guarantee plan executability and goal achievement at runtime, because there is no formal modeling of the boundaries and contingencies in which the system is going to operate. In addition, this requires that the executor be able to schedule activities at runtime. In fact, flexibility and controllability are complementary: controllability provides guarantees with respect to the modeled uncertainty, while flexibility allows the plan to be adjusted during execution. In principle, we can use any temporal planner that can generate flexible plans (e.g., VHPOP) in combination with our compilation to generate a flexible strong plan.

IxTeT (Ghallab and Laruelle 1994) was the first attempt to apply the results in temporal reasoning under uncertainty to planning, but the planner demanded the scheduling of a Simple Temporal Network with Uncertainty (STNU) (Vidal and Fargier 1999) by the plan executor. Here, we aim at generating plans that are guaranteed to work regardless of the temporal uncertainty. Nonetheless, IxTeT provides dynamic controllability: it generates a strategy for scheduling the actions depending on contingent observations. Although dynamic plans indeed can work in more situations than strong plans, they are also complex to generate, understand, and execute. When safety is paramount (e.g., space applications), dynamic plans might not be permitted because they require run-time computation that is hard to certify and to execute on-board in real-time. Strong plans are simple to execute and to check, and are suitable for safety critical systems where guarantees are needed for the uncertainty and sub-optimality is an acceptable price to pay.

In contrast to Beaudry, Kabanza, and Michaud (2010) we are concerned with qualitative uncertainty, meaning that we are not dealing with probability distributions, but only with durations that are bounded in convex intervals. In addition, we aim to guarantee goal achievement, while Beaudry, Kabanza, and Michaud maximize the probabilistic expectation.

There is a clear parallel between the problem we are solving and conformant planning (Ghallab, Nau, and Traverso 2004). In this sense, our work is similar to (Palacios and Geffner 2009) in which the authors transform conformant planning into deterministic planning, although the translation is very different.

The closest work to ours is that of Cimatti, Micheli, and Roveri (2013, 2015). Cimatti, Micheli, and Roveri (2013) present a logical characterization of the SPPTU for timelines with temporal uncertainty, as well as a first-order encoding of the problem having bounded horizon. Cimatti, Micheli, and Roveri (2015) cast the same idea in PDDL by extending state-space temporal planning. In this paper, we generalize both these frameworks, as we do not impose any bounded horizon for the problem and we consider a more expressive language allowing disjunctive preconditions, effects at arbitrary time points during actions and durative conditions on arbitrary sub-intervals. In Section 5 we provide a comparison with the techniques proposed in (Cimatti, Micheli, and Roveri 2015).

3 Modeling Duration Uncertainty

In (Cimatti, Micheli, and Roveri 2015), the authors propose an extension of PDDL 2.1 to model actions with uncontrollable duration. In this paper we use a richer language that includes timed-initial-literals (PDDL 2.2), durative goals (PDDL 3.0), and multi-valued variables (PDDL 3.1). In addition, we extend the language to allow conditions expressed over sub-intervals of actions, and effects at arbitrary time points during an action. These features turn out to be particularly useful for encoding many problems of interest, and for encoding our translation.[1] We first provide some brief background on PDDL 2.x and then describe our extensions.

In PDDL 2.2, a planning problem P is represented by a tuple $P \doteq \langle V, I, T, G, A \rangle$ where:

- V is a set of propositions.
- I is the initial state: a complete set of assignments of values `T` or `F` to all propositions in V.
- T is a set of timed-initial-literals, which are tuples $\langle [t] f := v \rangle$ with $f \in V$ and $t \in \mathbb{R}^+$ is the wall-clock time at which f will be assigned the Boolean value v.
- $G \subseteq V$ is a goal state: a set of propositions that need to be true when the plan finishes executing.
- A is a set of durative actions a, each of the form $a \doteq \langle d_a, C_a, E_a \rangle$ where:
  - $d_a \in \mathbb{R}^+$ is the action duration. Let st_a and et_a be the start and end times of action a then $d_a \doteq et_a - st_a$.
  - C_a is the set of conditions; each $p \in C_a$ is of the form $\langle (st_p, et_p)\ f = v \rangle$ where st_p and et_p indicate the start and end time points of the condition p and are restricted to be equal to st_a or et_a. When $st_p = et_p = st_a$ or $st_p = et_p = et_a$ then p is an instantaneous *at-start* or

[1] To simplify the presentation, we exclude some features of PDDL that are orthogonal to our approach of handling temporal uncertainty, such as *numeric variables* and *domain axioms*. Our techniques will work whether or not those features are included.

at-end condition holding at the st_p time point. When $st_p = s_a$ and $et_p = e_a$ then p is an *overall* durative condition holding in the open interval (st_p, et_p). $f \in V$ is a proposition with value $v = \texttt{T}$ or $v = \texttt{F}$ over the specified time period.

– E_a is a set of instantaneous effects, each $e \in E_a$ is of the form $\langle [t_e]\ f := v \rangle$ where $t_e \doteq st_a$ or $t_e \doteq et_a$ is the time at which the *at-start* or *at-end* effect e occurs.

We allow disjunctive action conditions p of the form $\langle (st_p, et_p)\ \bigvee_{i=1}^{n} f_i = v_i \rangle$ in which p is satisfied if at least one disjunct is satisfied for every time point in (st_p, et_p).

A plan π of P is a set of tuples $\langle t_a, a \rangle$, in which actions $a \in A$ are associated to wall-clock start times t_a. π is valid if it is executable in I and achieves all goals in G.

We extend the above features of PDDL 2.2 to include the following features from PDDL 3.0 and 3.1:

- *Multi-valued variables:* introduced in PDDL 3.1, allow variables f in V to have domains $Dom(f)$ with arbitrary values, instead of just T and F.
- *Durative goals:* which can be modeled as constraints in PDDL 3.0, allow each goal $g \in G$ to be associated with an interval $[st_g, et_g]$ specifying when the goal must be true. In this setting, we allow the time constant et_π that indicates that the goal must be reached at the end of the plan.

Beyond PDDL. Additionally, the key features in our framework that go beyond PDDL are: (1) actions can have uncontrollable durations, and (2) action conditions and effects are not restricted to action start or end time points. Specifically:

1. Action duration d_a is replaced by an interval $[d_a^{lb}, d_a^{ub}]$ specifying lower- and upper-bound values on action duration: $d_a^{lb} \leq d_a \leq d_a^{ub}$. We further divide the set of actions A into two subsets:
   - *Controllable* actions A_c, where the duration can be chosen by the planner within the bounds $[d_a^{lb}, d_a^{ub}]$.
   - *Uncontrollable* actions A_u, where the duration is not under the planner's control.
2. Instead of constraining the times st_p and et_p of each condition p or the time t_e of effect e to be either st_a or et_a, we allow each of them to take an arbitrary value: $st_a + \delta$ or $et_a - \delta$, with $\delta \in \mathbb{R}^+$ (the temporal constraint $st_p \leq et_p$ should still be satisfied). We require δ to be positive and less than or equal to the action minimal duration to prevent effects before the start or after the end of the action. Analogously to PDDL, If $st_p = et_p$ the condition is instantaneous and is required to hold at the single point st_p, otherwise, the condition is required to hold in the open interval (st_p, et_p).

A (strong) plan π^u for a planning problem with uncertainty P^u is valid iff each instance of π^u, obtained by fixing the duration d_a for each uncontrollable action $a \in \pi^u$ to any value within $[d_a^{lb}, d_a^{ub}]$, is a valid plan.

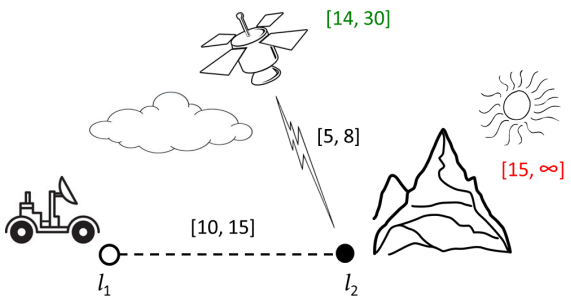


Figure 1: A graphical representation of the running example.

Example. A rover, initially at location l_1, needs to transmit some science data from location l_2 to an orbiter that is only visible in the time window $[14, 30]$. The rover can move from l_1 to l_2 using an action $move$ (abbreviated μ) that has uncontrollable duration between 10 and 15 time units. The data transmission action $transmit$ (abbreviated τ) takes between 5 and 8 time units to complete. The goal of the rover is to transmit the data to the orbiter. Because of the harsh daytime temperatures at location l_2 the rover cannot arrive at l_2 until the sun goes behind the mountains at time 15. Figure 1 illustrates this scenario, which we encode as:

$$V \doteq \{pos : \{l_1, l_2\}, visible : \{\texttt{T}, \texttt{F}\}, hot : \{\texttt{T}, \texttt{F}\}, sent : \{\texttt{T}, \texttt{F}\}\}$$
$$I \doteq \{pos = l_1, visible = \texttt{F}, sent = \texttt{F}, hot = \texttt{T}\}$$
$$T \doteq \{\langle [14]\ visible := \texttt{T} \rangle, \langle [30]\ visible := \texttt{F} \rangle, \langle [15]\ hot := \texttt{F} \rangle\}$$
$$G \doteq \{\langle [et_\pi, et_\pi]\ sent = \texttt{T} \rangle\}$$
$$A_c \doteq \varnothing$$
$$A_u \doteq \{\langle [10, 15], C_\mu, E_\mu \rangle, \langle [5, 8], C_\tau, E_\tau \rangle\}$$
$$C_\mu \doteq \{\langle (st_\mu, st_\mu)\ pos = l_1 \rangle, \langle (et_\mu, et_\mu)\ hot = \texttt{F} \rangle\}$$
$$C_\tau \doteq \{\langle (st_\tau, et_\tau)\ pos = l_2 \rangle, \langle (st_\tau, et_\tau)\ visible = \texttt{T} \rangle\}$$
$$E_\mu \doteq \{\langle [et_\mu]\ pos := l_2 \rangle\}$$
$$E_\tau \doteq \{\langle [et_\tau]\ sent := \texttt{T} \rangle\}$$

Figure 2 graphically shows a valid plan:

$$\pi^u \doteq \{\langle 6, move(l_1 \rightarrow l_2) \rangle, \langle 22, transmit \rangle\}$$

Note that all the actions in π^u have uncontrollable duration.

Discussion. In general, finding a strong plan for a problem with duration uncertainty is different from simply considering the maximum or the minimum duration for each action. Consider our rover example and its strong plan shown in Figure 2. The μ (i.e., $move$) action must terminate before the transmit action can start and, at the same time, μ cannot terminate before time 15 due to the temperature constraint. If we only consider the lower-bound on the duration of μ (i.e., planning with $d_\mu^{lb} = 10$) then one valid plan is: $\pi_{lb} \doteq \{\langle 11, \mu \rangle, \langle 22, \tau \rangle\}$. However, because of the uncertainty in the actual execution duration of μ, it may actually take 14 time units to arrive at l_2. Thus, the rover would start transmitting at time 22 before it actually arrives at l_2 at time $11 + 14 = 25$. Thus, plan π_{lb} is invalid. Similarly, if we consider only the maximal duration (i.e., planning with $d_\mu^{ub} = 15$), then one possible plan would

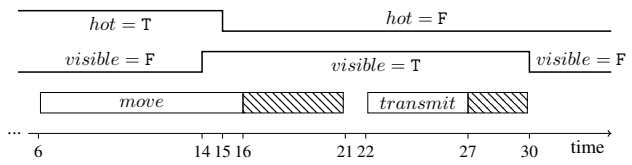


Figure 2: Graphical execution of π^u. Striped regions represent the uncertainty in the action duration.

be: $\pi_{ub} \doteq \{\langle 1, \mu\rangle, \langle 22, \tau\rangle\}$. However, during the actual execution of μ, it may again take only 11 time units (and not the planned maximum 15 time units) to arrive at l_2. This would violate the constraint that it should arrive at l_2 after $t = 15$ to avoid the sun, so π_{ub} is also not a valid plan[2].

Disjunctive Conditions: In contrast to ordinary temporal planning, it is not possible to compile away disjunctive conditions using the action duplication technique (Gazen and Knoblock 1997), because the set of satisfied disjuncts in the presence of uncertainty can depend on the contingent execution. For example, consider an action a starting at time t where two Boolean variables p_1 and p_2 are F. a has uncontrollable duration in $[l, u]$, an at-start effect $e_1 \doteq \langle [st_a] p_1 := \mathtt{T}\rangle$ and two at-end effects $e_2 \doteq \langle [et_a] p_1 := \mathtt{F}\rangle$ and $e_3 \doteq \langle [et_a]\, p_2 := \mathtt{T}\rangle$. An at-start condition $p_1 \vee p_2$ of another action b is satisfied anywhere between the start of the action a and the next deletion of p_2. Thus, b can start anytime within $d \doteq (t + l, t + u]$. However, if we compile away this disjunctive condition by replacing b with two actions b_1 and b_2: one with an at-start condition p_1 and the other with an at-start condition p_2, then b_1 is not executable within d because there is no time point in d in which we can guarantee that $p_1 = \mathtt{T}$ (because a may take the minimum duration l and thus the at-end effect e_2 will occur at $t + l$ to set $p_1 = \mathtt{F}$). Similarly, we cannot start b_2 within d because we also cannot guarantee that $p_2 = \mathtt{T}$ at anytime point within d (because a may take the maximum duration u and thus e_3 that set $p_2 = \mathtt{T}$ will not happen until $t + u$). Thus, compiling away disjunctive conditions leads to incompleteness when there are uncontrollable durative actions. For this reason it is important to explicitly model disjunctive conditions in our language.

4 Compilation Technique

In this section, we present our compilation technique, which can be used to reduce any planning instance P having duration uncertainty into a temporal planning instance P' in which all actions have controllable durations. The translation guarantees that P is solvable if and only if P' is solvable. Moreover, given any plan for P' we can derive a plan for P. This approach comes at the cost of duplicating some of the variables in the domain, but allows for the use of off-the-shelf temporal planners.

[2]In some cases it is possible to soundly consider only the maximal duration for an action but this special-case optimization is not sound in general.

The overall intuition behind the translation is the following. Consider the $transmit$ (i.e., τ) action in our example, and suppose it is scheduled to start at time k. Let v be the value of $sent$ at time $k + 5$; since $transmit$ has an at-end effect $\langle [et_\tau]\ sent := \mathtt{T}\rangle$, we know that the value of the variable $sent$ during the interval $(k + 5, k + 8]$ will be either v or T depending on the duration of the action. After time $k + 8$ we are sure that the effect took place, and we are sure of the value of $sent$ until another effect is applied.[3] Since we are not allowed to observe anything at run-time in strong planning, we need to consider this uncertainty in the value of $sent$ and produce a plan that works regardless. Since $sent$ could appear as a condition of another action (or as a goal condition, as in our example) we must rewrite such conditions to be true only if both T and v are values that satisfy the condition.

To achieve this, we create an additional variable $sent_\sigma$ (the *shadow variable of* $sent$). This secondary variable stores the alternative value of $sent$ during uncertainty periods. When there is no uncertainty in the value of $sent$, both $sent$ and $sent_\sigma$ will have the same value. In this way, all the conditions involving $sent$ can be rewritten in terms of $sent$ and $sent_\sigma$ to ensure they are satisfied by both the values.

In general, our translation works by rewriting a planning instance $P \doteq \langle V, I, T, G, A\rangle$ into a new planning instance $P' \doteq \langle V', I', T', G', A'\rangle$ that does not contain actions with uncontrollable duration.

Uncertain Variables. The first step is to identify the set of variables $L \subseteq V$ that appear as effects of uncontrollable actions and are executed at a time depending on the end of the action.

$$L \doteq \{f \mid a \in A_u, \langle [t]\ f := v\rangle \in E_a, t \doteq et_a - \delta\}$$

Intuitively, this is the set of variables that can possibly have uncertain value during plan execution. A variable that is modified only at times linked to the start of actions or by timed initial literals, cannot be uncertain as neither the starting time of actions nor the timed initial literals can be uncertain in our model. In our running example, the set L becomes $\{sent, pos\}$.

We now define the set V' as the original variables V plus a shadow variable for each variable appearing in L.

$$V' \doteq V \cup \{f_\sigma \mid f \in L\}$$

We use the pair of variables f and f_σ to represent uncertainty: if $f = f_\sigma$ we know that there is no uncertainty in the value of f, while if $f \neq f_\sigma$ we know that the actual value of f in the original problem is either f or f_σ.

Disjunctive Conditions. At the end of Section 3, we outlined the reason why existing approaches of compiling away disjunctive conditions will not work with uncontrollable action durations. In order to rewrite a disjunctive condition $p \doteq \langle (st_p, et_p)\ \bigvee_{i=1}^{n} f_i = v_i\rangle$ we need to ensure that the

[3]Note that there cannot be another concurrent action in the plan having an effect on $sent$ during the interval $[k + 5, k + 8]$ because this would allow for the possibility of two concurrent effects on the same variable.

result is satisfied if and only if both the values of f and f_σ for each $f \in L$ are satisfying values for p. For this reason, we define an auxiliary function $\chi(\psi)$ that takes a single disjunctive condition in P and returns a set of disjunctive conditions in P'.

$$\chi(\psi) \doteq \begin{cases} \{\langle f = v \rangle\} & \text{if } \psi \doteq \langle f = v \rangle, f \notin L \\ \{\langle f = v \rangle, \langle f_\sigma = v \rangle\} & \text{if } \psi \doteq \langle f = v \rangle, f \in L \\ \{r \vee s \mid r \in \chi(\psi_1), s \in \chi(\psi_2)\} & \text{if } \psi \doteq \psi_1 \vee \psi_2 \end{cases}$$

For example, the condition of the τ action, $pos = l_2$, is translated as the two conditions $pos = l_2$ and $pos_\sigma = l_2$. Analogously, assuming that both f and g are in L, a given condition $(f = \mathtt{T}) \vee (g = \mathtt{F})$ in P is translated by function χ as the set of conditions $\{(f = \mathtt{T}) \vee (g = \mathtt{F}), (f_\sigma = \mathtt{T}) \vee (g = \mathtt{F}), (f = \mathtt{T}) \vee (g_\sigma = \mathtt{F}), (f_\sigma = \mathtt{T}) \vee (g_\sigma = \mathtt{F})\}$ in P'.

Uncertain Temporal Intervals. We also need to identify the temporal interval in which the value of a given variable can be uncertain. Given an action a with uncertain duration d_a in $[l, u]$, let $\lambda(t)$ and $\nu(t)$ be the earliest and latest possible times at which an at-end effect at $t \doteq et_a - \delta$ may happen. Thus: $\lambda(t) \doteq st_a + l - \delta$ and $\nu(t) \doteq st_a + u - \delta$. Both functions are equal to t if $t \doteq st_a + \delta$. For example, consider the effect $e_1 \doteq \langle [et_\tau] sent := \mathtt{T} \rangle$ of action τ. We know that the duration of transmit is uncertain in $[5, 8]$, therefore the effect can be applied between $\lambda(et_\tau) \doteq st_\tau + 5$ and $\nu(et_\tau) \doteq st_\tau + 8$ and the $sent$ variable has an uncertain value within that interval.

Uncontrollable Actions. For each uncontrollable action $a \doteq \langle [l, u], C_a, E_a \rangle$ in A_u in the original model we create a new action $a' \doteq \langle [u, u], C_{a'}, E_{a'} \rangle$ in A'_c. Specifically, we first fix the maximal duration u as the only allowed duration for a' and then insert appropriate effects and conditions *during* the action to capture the uncertainty.

The effects $E_{a'}$ are partitioned in two sets $E^l_{a'}$ and $E^u_{a'}$ to capture possible values within the uncertain action execution duration. The conditions $C_{a'}$ are also composed of two elements: the rewritten conditions $C^R_{a'}$ and the conditions added to protect the new effects $C^E_{a'}$ (thus $C' \doteq C^R_{a'} \cup C^E_{a'}$).

Rewritten conditions $C^R_{a'}$: are obtained by rewriting existing action conditions by means of the χ function. The intervals specifying the duration of the conditions are preserved; since the action duration is set to its maximum, the intervals of the conditions are "stretched" to match their maximal duration.

$$C^R_{a'} \doteq \{\langle (\lambda(t_1), \nu(t_2))\ \alpha \rangle \mid \alpha \in \chi(\psi), \langle (t_1, t_2)\ \psi \rangle \in C_a\}$$

For example, the set C^R_τ for the τ action is: $\{\langle (st_\tau, st_\tau + 8)\ pos = l_2 \rangle, \langle (st_\tau, st_\tau + 8)\ pos_\sigma = l_2 \rangle, \langle (st_\tau, st_\tau + 8)\ visible = \mathtt{T} \rangle\}$. This requires variables $visible$, pos and pos_σ to be true throughout the execution of τ.

Compiling action effects: The effects of the original action are duplicated: both the affected variable f and its shadow f_σ are modified, but at different times. We first identify the earliest and latest possible times at which an effect can happen due to the duration uncertainty (see earlier discussion on $\lambda(t)$ and $\nu(t)$). We then apply the effect on f_σ at the earliest time point $\lambda(t)$, and at the latest time point $\nu(t)$ we re-align

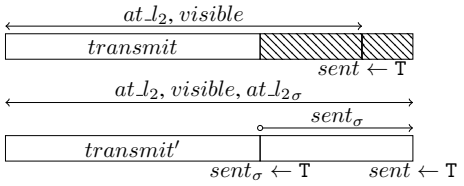


Figure 3: Graphical view of the original $transmit$ action instance (top) and its compilation (bottom).

f and f_σ by also applying the effect on f:

$$E^l_{a'} \doteq \{\langle [\lambda(t)]\ f_\sigma := v \rangle \mid \langle [t]\ f := v \rangle \in E_a\}$$

$$E^u_{a'} \doteq \{\langle [\nu(t)]\ f := v \rangle \mid \langle [t]\ f := v \rangle \in E_a\}$$

For example, the τ action has $E^l_\tau \doteq \{\langle [st_\tau + 5]\ sent_\sigma := \mathtt{T} \rangle\}$ and $E^u_\tau \doteq \{\langle [st_\tau + 8]\ sent := \mathtt{T} \rangle\}$.

Additional conditions $C^E_{a'}$: let $t \doteq et_a - \delta$ be the time of an at-end effect that affects the value of f. In order to prevent other actions from changing the value of f during the interval $(\lambda(t), \nu(t)]$ where the value of f is uncertain, we add a condition in $C^E_{a'}$ to maintain the value of f_σ throughout the uncertain duration $(\lambda(t), \nu(t)]$.

$$C^E_{a'} \doteq \{\langle (\lambda(t), \nu(t))\ f_\sigma = v \rangle, \mid \langle [t]\ f := v \rangle \in E_a\} \cup \{\langle (\nu(t), \nu(t))\ f_\sigma = v \rangle \mid \langle [t]\ f := v \rangle \in E_a\}$$

We are in fact using a left-open interval $(\lambda(t), \nu(t)]$ by specifying the same condition on the open interval $(\lambda(t), \nu(t))$ and the single point $[\nu(t)]$. Since the effect on f_σ (belonging to $E^l_{a'}$) is applied at time $\lambda(t)$, the condition is satisfied immediately after the effect and we want to avoid concurrent modifications of either f or f_σ until the uncertainty interval ends at $\nu(t)$. For example, the τ action has $C^E_{\tau'} \doteq \{\langle (st_\tau + 5, st_\tau + 8)\ sent_\sigma = \mathtt{T} \rangle\}$. Compilation of the τ action is depicted in Figure 3.

Controllable actions: are much simpler. For each $a \doteq \langle [l, u], C_a, E_a \rangle \in A_c$ we introduce a replacement action $a' \doteq \langle [l, u], C_{a'}, E_{a'} \rangle \in A'_c$, in which: (1) each condition in C is rewritten to check the values of both the variables and their shadows, and (2) each effect is applied to a variable and its shadow, if any.

$$C_{a'} \doteq \{\langle (\lambda(t_1), \nu(t_2))\ \alpha \rangle \mid \alpha \in \chi(\psi), \langle (t_1, t_2)\ \psi \rangle \in C_a\}$$

$$E_{a'} \doteq E_a \cup \{\langle [t]\ f_\sigma := v \rangle \mid f \in L, \langle [t]\ f := v \rangle \in E_a\}$$

Initial state I: is handled by initializing variables and their corresponding shadow variables in the same way as in the original problem.

$$I' \doteq I \cup \{f_\sigma = v \mid f \in L, f = v \in I\}$$

For example, the initial state of our running problem is the original initial state plus $\{sent_\sigma = \mathtt{F}, pos_\sigma = l_1\}$.

Timed Initial Literals: T' are set similarly to the effects.

$$T' \doteq T \cup \{\langle [t]\ f_\sigma := v \rangle \mid f \in L, \langle [t]\ f := v \rangle \in T\}$$

In our example, we do not have timed initial literals operating on uncertain variables, thus $T \doteq T'$.

Goal conditions: G is augmented to consider both the original and shadow variables, without modifying the application times, since they are fixed and cannot be uncertain.

$$G' \doteq G \cup \{\langle [t_1, t_2]\ f_\sigma = v\rangle \mid f \in L, \langle [t_1, t_2]\ f = v\rangle \in G\}$$

In our example, the set G' becomes $\{\langle [et_\pi, et_\pi]\ sent = \mathtt{T}\rangle, \langle [et_\pi, et_\pi]\ sent_\sigma = \mathtt{T}\rangle\}$.

Example. The compilation for our example problem is:

$$V' \doteq V \cup \{pos_\sigma : \{l_1, l_2\}, sent_\sigma : \{\mathtt{T}, \mathtt{F}\}\}$$
$$I' \doteq I \cup \{pos_\sigma = l_1, sent_\sigma = \mathtt{F}\}$$
$$T' \doteq \{\langle [14]\ visible := \mathtt{T}\rangle, \langle [30]\ visible := \mathtt{F}\rangle, \langle [15]\ hot := \mathtt{F}\rangle\}$$
$$G' \doteq \{\langle [et_\pi, et_\pi]\ sent = \mathtt{T}\rangle, \langle [et_\pi, et_\pi]\ sent_\sigma = \mathtt{T}\rangle\}$$
$$A'_c \doteq \{\langle [15, 15], C_{\mu'}, E_{\mu'}\rangle, \langle [8, 8], C_{\tau'}, E_{\tau'}\rangle\}$$
$$C_{\mu'} \doteq \{\langle (st_\mu, st_\mu)\ pos = l_1\rangle, \langle (st_\mu, st_\mu)\ pos_\sigma = l_1\rangle,$$
$$\langle (et_\mu, et_\mu)\ hot = \mathtt{F}\rangle, \langle (st_\mu + 10, st_\mu + 10)\ pos_\sigma = l_2\rangle,$$
$$\langle (st_\mu + 10, st_\mu + 15)\ pos_\sigma = l_2\rangle\}$$
$$C_{\tau'} \doteq \{\langle (st_\tau, et_\tau)\ pos = l_2\rangle, \langle (st_\tau, et_\tau)\ pos_\sigma = l_2\rangle,$$
$$\langle (st_\tau, et_\tau)\ visible = \mathtt{T}\rangle, \langle (st_\tau + 5, st_\tau + 8)\ sent_\sigma = \mathtt{T}\rangle,$$
$$\langle (st_\tau + 5, st_\tau + 5)\ sent_\sigma = \mathtt{T}\rangle\}$$
$$E_{\mu'} \doteq \{\langle [st_\mu + 10]\ pos_\sigma := l_2\rangle, \langle [st_\mu + 15]\ pos := l_2\rangle\}$$
$$E_{\tau'} \doteq \{\langle [st_\tau + 5]\ sent_\sigma := \mathtt{T}\rangle, \langle [st_\tau + 8]\ sent := \mathtt{T}\rangle\}$$

Discussion. This compilation is sound and complete. Thus, the original problem is solvable if and only if the resulting problem is solvable Any plan for the rewritten temporal planning problem is automatically a strong plan for the original problem (with the obvious mapping from the rewritten to the original actions).

Theorem 1. *Let $P \doteq \langle V, I, T, G, A\rangle$ be a planning instance and $R \doteq \langle V', I', T', G', A'\rangle$ be its translation. P has a strong plan π if and only if R has a temporal plan σ.*

Proof. (Sketch) Let π be a strong plan for P. Let σ be the plan starting a' at time t for each a starting at time t in π. σ is a valid temporal plan for R because:

- It achieves the goal G' of R: all original goals in G are achieved by π and by σ in the same way, and the goals on the shadow variables must be achieved because π is a strong plan. Given that π achieves the goals regardless of the concrete durations of the actions, it achieves them outside of the uncertainty intervals, where the variables and the shadow variables are aligned.
- Each action a' is executable in R, because each $a \in \pi$ is executable in P regardless of the action durations. Thus the possible uncertainty introduced by the durations is irrelevant for the executability of a (all the conditions are satisfied). In the translated instance R, all the conditions are also satisfied because the conditions are imposed via the χ function that only checks that both the variable and its shadow fulfill the original condition.
- No conflicting effects are possible: the conditions added in $C^E_{a'}$ prevents any modification of the interested shadow variables during the uncertainty intervals.

Similarly, let σ be a plan for R. Let π be the plan for P starting a at time t for each a' starting at time t in σ. π is a valid strong temporal plan for P because:

- It achieves the goal G, because σ achieves the goal G' that is a super-set of G and each translated action has all effects of the original actions.
- Each action a is executable in P regardless of the action duration, because each $a' \in \sigma$ is executable in R and the conditions in the translated actions are a super-set of the ones in the original action, due to the χ function.
- No conflicting effects are possible regardless of the uncertain duration, because each effect at time t can be uncertain only between $\lambda(t)$ and $\nu(t)$ and we guarantee no other effect is possible in that interval by means of $C^E_{a'}$.

□

The compilation produces a problem that has: (i) at most twice the number of variables of the original problem, (ii) at most twice the initial and timed assignments and (iii) exactly the same number of actions. The only point in which the compilation might produce exponentially large formulae is in the application of the χ function, which is exponential in the number of disjuncts constraining variables appearing in L. Since this only happens for disjunctive conditions, and the number of disjuncts is typically small, this is normally not a serious issue.

5 Implementation and Experiments

We conducted two sets of experiments. In the first, we compare our approach against the techniques proposed in (Cimatti, Micheli, and Roveri 2015). This is the only domain-independent planner that we are aware of that can find strong plans for PDDL 2.1 problems with uncontrollable durations. For this experiment, we use an extension to PDDL 2.1 that includes actions with uncontrollable durations (but none of the other extensions that we described in Section 3 such as preconditions and effects at arbitrary times, multi-valued variables, timed-initial-literals, or disjunctive preconditions).

In the second set of experiments, we show the applicability of our technique on a very expressive fragment of the ANML language (Smith, Frank, and Cushing 2008) extended with uncertainty in action durations. Except for action duration uncertainty, ANML natively supports all the other features described in Section 3.

PDDL with duration uncertainty. Cimatti, Micheli, and Roveri (2015) extended the COLIN planner (Coles et al. 2012) to solve SPPTUs by replacing the STN scheduler with a solver for strong controllability of STNUs. This yields a sound but incomplete SPPTU planner. The authors

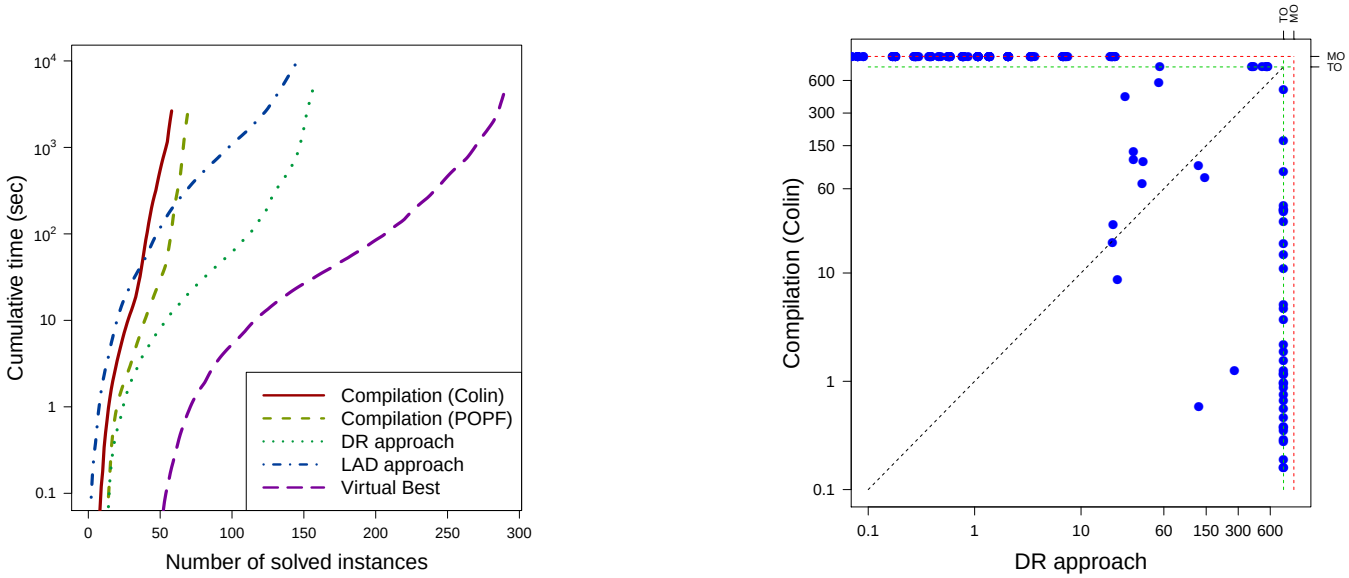


Figure 4: Experimental results for the PDDL with duration uncertainty comparison. The left picture shows the cumulative time plot of the solving time for the PDDL benchmarks. The right picture reports the scatter plot of the running time in seconds for the compilation approach (solved using the COLIN planner) against the DR approach.

then proposed two techniques to overcome the incompleteness: (1) "Last Achiever Deordering" (LAD) is a sound-but-incomplete technique that tries to limit the incompleteness by building the partial orderings in STNU using the last achiever for each condition that needs support; (2) "Disjunctive Reordering" (DR) is a sound-and-complete technique obtained by considering, at each step, all the possible valid action reorderings using a disjunctive form of STNU.

We compare against this approach by first compiling away temporal uncertainties and then using both the COLIN and POPF planners to solve the compiled instances[4]. We compared our *sound and complete* technique against both the *complete* DR and the *incomplete* LAD. We used a timeout of 600 seconds, a memory limit of 8 GB and the full benchmark set of 563 problems described in (Cimatti, Micheli, and Roveri 2015).

The left plot of Figure 4 reports the cumulative time of the three techniques and the "Virtual Best" solver, obtained by picking the best solving technique for each instance. The right scatter-plot compares our technique (instantiated with COLIN) with the DR approach. The left plot shows that the compilation technique cannot solve as many instances as DR or LAD. However, we note that the "Virtual Best" solver solves many more problems than both the DR and LAD. This shows that the techniques are complementary: problem instances that cannot be solved by LAD or DR are solved quickly by our compilation, and vice-versa. This situation is also visible in the scatter plot: there is a clear subdivision of the problem instances solved by these two different planners.

Our investigation indicates that the main factor that hinders the performance of our approach is the "clip-action" construction (Fox, Long, and Halsey 2004) that is needed to reduce our compilation to PDDL 2.1. In particular, our compilation generates actions with conditions and effects that occur at intermediate times. Compiling this to PDDL 2.1 requires three PDDL 2.1 actions for each action in A_u: a container action, and two actions inside the container action that are clipped together. This deepens the search and lengthens the plans for COLIN and POPF.

ANML with duration uncertainty. As described in Sections 3 and 4, our framework handles many useful features beyond PDDL 2.1. Some of these can be represented in higher levels of PDDL (e.g., multi-valued variables), some cannot (e.g., arbitrary timed action conditions and effects). While comparing against current state-of-the-art in PDDL2.1 shows the feasibility of our approach, it restricts us to a subset of features that can be handled by our compilation. Moreover, as discussed above, the limitation of PDDL 2.1 adversely impacts the performance of our approach.

To show the full expressive potential of our approach, we used the Action Notation Modeling Language (ANML) (Smith, Frank, and Cushing 2008), which can natively model all those constraints. ANML is a variable-value language that allows high-level modeling, complex conditions and effects, HTN decomposition and timeline-like temporal constraints. Our only addition to ANML is the capability to model uncertain action durations: `duration :in [l, u]` where l and u are constant values specifying the lower and upper bounds on the duration of a. We name our

[4]Our approach allows the use of any PDDL2.1 planner that can handle required concurrency. Unfortunately, many temporal planners such as LPG and TemporalFastDownward do not support this, and therefore cannot find solutions to the problems generated by our compilation.

Planning Instance	Controllable	Uncontrollable
match 1	0.517	0.626
match 2	0.522	0.637
match 3	0.593	1.115
rovers 1	1.196	1.293
rovers 2	1.497	1.810
rovers 3	1.190	2.009
handover 1	0.800	1.081
handover 2	2.302	4.043
handover 3	2.863	32.484

Table 1: Results of the ANML comparison. The table reports the solving time in seconds for the two analyzed configurations. The *Controllable* column reports the runtime for the instances in which the durations are considered as controllable by the planner. The *Uncontrollable* column reports the runtime for solving SPPTU using our compilation in combination with the FAPE planner.

ANML extension: ANuML.

We implemented our compilation approach in an automatic translator that accepts an ANuML planning instance and produces plain ANML. We then use the FAPE (Dvorak et al. 2014) planner to produce a plan for the compiled ANML problem instance. To the best of our knowledge no other approach is able to solve the problems we are dealing with in ANML.

We considered two domains adapted from the FAPE distribution namely "rover" and "handover". The former models a remote rover for scientific operations, similar to our running example, while the latter models a situation in which two robots must synchronize to exchange items. Additionally, we model a "match" domain derived from the "matchcellar" domain used in IPC 2011. For each domain, we tested with three different configurations: different initial states, goals, and variable domains.

Table 1 compares the time needed for FAPE to produce a plan ignoring the temporal uncertainty (i.e. considering the environment to be completely cooperative) with the time needed to solve the compiled problem. Although the performance of the encoding depends on the planning instance, the results show that the slowdown is acceptable for the tested instances. An exception is "handover 3", in which the translation shows a significant slowdown. We remark that this is not a comparison between two equivalent techniques, as the two columns correspond to results in solving very different problems: plain temporal planning vs. strong planning with temporal uncertainty. Instead, this is an indication of the slowdown introduced by the translation compared to the same problem without uncertainty. Even though the results are preliminary, we can infer that our approach is more than a theoretical exercise and can be practically applied for temporal planning domains modeled natively in ANML.

6 Future Work

While the preliminary results are promising, we are considering several possible extensions.

Model simplification: it is sometimes possible to simplify a strong planning problem with temporal uncertainty by considering the maximal or minimal duration of an action having uncertain duration. As we discussed in Section 3, this "worst-case" approach is in general unsound; nonetheless, it is possible to recognize some special cases in which it is sound and complete. This simplification can be done upfront and could be beneficial for both our compilation and the approaches in (Cimatti, Micheli, and Roveri 2015). Precisely understanding when and how this simplification can be applied is an open problem, but a preliminary analysis suggests that an action a with uncertain duration $[l, u]$, having conditions involving variables in V_C and effects involving variables in V_E, can be considered as being controllable in $[u, u]$ (without changing its conditions or effects) if all the following conditions hold:

- A is *mutually exclusive* with any other action that has an effect on a variable in $V_E \cup V_C$.
- No Timed Initial Literal modifies a variable in $V_E \cup V_C$.
- Every action with a condition involving V_C is *mutually exclusive* with A.

These strict requirements are sufficient to guarantee that the action can be considered to last for its maximal duration as it is impossible to impose a "lower-bound" constraint in any valid execution of the actions.

Increase expressiveness: Even though the formalization we presented is quite expressive and general, the ANML language has many features that are not covered. A prominent example is the support of conditional effects, which cannot be expressed in our language but are possible in both ANML and PDDL. We note that, analogously to disjunctive preconditions, the common compilation of conditional effects is unsound in the presence of temporal uncertainty, because it transforms a possibly uncontrollable effect into a controllable decision for the planner. Nonetheless, our translation (with some modifications) is still applicable in the presence of conditional effects, but it sacrifices completeness. The intuitive reason is that we represent uncertainty as a pair of variables (original and shadow) with the assumption that the value of the variable in the original execution is either of the two values. With conditional effects, it is possible to design models in which the variable can actually be uncertain between more than two values.

Improve performance: Finally, we would like to study ways to overcome the disappointing performance of the compilation into PDDL by hybridizing the "native" DR and LAD techniques with our approach to exploit their complementarity. Another possibility is to modify a temporal planner so that it understands the clip-action construct and avoids useless search when dealing with the instances produced by our translation.

Acknowledgements: We would like to thank Jeremy Frank, Alessandro Cimatti and Paul Morris for suggestions, fruitful discussion, and feedback on an early version of this paper. This work was supported by the NASA Automation for Operations (A4O) project.

References

Beaudry, E.; Kabanza, F.; and Michaud, F. 2010. Planning for concurrent action executions under action duration uncertainty using dynamically generated bayesian networks. In *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling (ICAPS)*.

Bresina, J. L.; Dearden, R.; Meuleau, N.; Ramakrishnan, S.; Smith, D. E.; and Washington, R. 2002. Planning under continuous time and resource uncertainty: A challenge for AI. In *UAI '02, Proceedings of the 18th Conference in Uncertainty in Artificial Intelligence, University of Alberta, Edmonton, Alberta, Canada, August 1-4, 2002*, 77–84.

Cesta, A.; Cortellessa, G.; Fratini, S.; Oddi, A.; and Rasconi, R. 2009. The APSI Framework: a Planning and Scheduling Software Development Environment. In *Working Notes of the ICAPS-09 Application Showcase Program*.

Cimatti, A.; Micheli, A.; and Roveri, M. 2013. Timelines with temporal uncertainty. In *AAAI*, 195–201.

Cimatti, A.; Micheli, A.; and Roveri, M. 2014. Solving strong controllability of temporal problems with uncertainty using SMT. *Constraints*.

Cimatti, A.; Micheli, A.; and Roveri, M. 2015. Strong temporal planning with uncontrollable durations: a state-space approach. In *AAAI*.

Coles, A. J.; Coles, A.; Fox, M.; and Long, D. 2012. Colin: Planning with continuous linear numeric change. *J. Artif. Intell. Res. (JAIR)* 44:1–96.

Dvorak, F.; Bit-Monnot, A.; Ingrand, F.; and Ghallab, M. 2014. A flexible anml actor and planner in robotics. In *ICAPS-14 Planning and Robotics Workshop*.

e Assis Santana, P. H. R. Q., and Williams, B. C. 2012. A bucket elimination approach for determining strong controllability of temporal plans with uncontrollable choices. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada.*

Fox, M.; Long, D.; and Halsey, K. 2004. An investigation into the expressive power of PDDL2.1. In *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004, Valencia, Spain, August 22-27, 2004*, 328–342.

Frank, J., and Jónsson, A. 2003. Constraint-based Attribute and Interval Planning. *Constraints* 8(4):339–364.

Gazen, B. C., and Knoblock, C. A. 1997. Combining the expressiveness of UCPOP with the efficiency of Graphplan. In Steel, S., and Alami, R., eds., *Recent Advances in AI Planning: 4th European Conference on Planning, ECP'97*. New York: Springer-Verlag.

Ghallab, M., and Laruelle, H. 1994. Representation and control in ixtet, a temporal planner. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems (AIPS), University of Chicago, Chicago, Illinois, USA, June 13-15, 1994*, 61–67.

Ghallab, M.; Nau, D. S.; and Traverso, P. 2004. *Automated planning - theory and practice*. Elsevier.

Mausam, and Weld, D. S. 2008. Planning with durative actions in stochastic domains. *J. Artif. Intell. Res. (JAIR)* 31:33–82.

Morris, P. 2006. A structural characterization of temporal dynamic controllability. In *Principles and Practice of Constraint Programming - CP 2006, 12th International Conference, CP 2006, Nantes, France, September 25-29, 2006, Proceedings*, 375–389.

Muise, C. J.; Beck, J. C.; and McIlraith, S. A. 2013. Flexible execution of partial order plans with temporal constraints. In *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*.

Palacios, H., and Geffner, H. 2009. Compiling uncertainty away in conformant planning problems with bounded width. *J. Artif. Intell. Res. (JAIR)* 35:623–675.

Peintner, B.; Venable, K. B.; and Yorke-Smith, N. 2007. Strong controllability of disjunctive temporal problems with uncertainty. In *CP*, 856–863.

Smith, D. E.; Frank, J.; and Cushing, W. 2008. The ANML language. In *ICAPS 2008 - Poster session*.

Vidal, T., and Fargier, H. 1999. Handling contingency in temporal constraint networks: from consistency to controllabilities. *J. Exp. Theor. Artif. Intell.* 11(1):23–45.

Younes, H. L. S., and Simmons, R. G. 2004. Solving generalized Semi-Markov Decision Processes using continuous phase-type distributions. In *AAAI*, 742748.

On Applicability of Automated Planning for Incident Management

Lukáš Chrpa
PARK Research group
School of Computing and Engineering
University of Huddersfield

Kristinn R. Thórisson
Icelandic Institute of Intelligent Machines, Reykjavik &
School of Computer Science, Reykjavik University

Abstract

Incident management aims to save human lives, mitigate the effect of accidents, prevent damages, to mention a few of their benefits. Efficient coordination of rescue team members, allocation of available resources, and appropriate responses to the realtime unfolding of events is critical for managing incidents successfully. Coordination involves a series of decisions and event monitoring, usually made by human coordinators, for instance task definition, task assignment, risk assessment, etc. Each elementary decision can be described by a named action (e.g. boarding an ambulance, assigning a task). Taken as a whole, the team coordinating an incident response can be seen as a decision-making system. In this paper, we discuss how invaluable assistance can be brought to such a system using automated planning. In consultation with experts we have derived a set of requirements from which we provide a formal specification of the domain. Following the specification, we have developed a prototype domain model and evaluated it empirically. Here we present the results of this evaluation, along with several challenges (e.g uncertainty) that we have identified.

Introduction

Automated planning is a research discipline that addresses the problem of generating a totally- or partially-ordered sequence of actions that transform the environment from some initial state to a desired goal state. Automated planning has been successfully applied for decades in several areas, including space exploration (Ai-Chang et al. 2004) or machine tool calibration (Parkinson, Longstaff, and Fletcher 2014), to mention a couple.

Incident management consists of time- and resource-critical operations that aim to save human lives, mitigate the effect of traffic or industrial accidents, prevent excessive damages, to mention a few of their benefits. In other words, incident management involves groups of people who need to achieve close coordination to carry out one or more time-critical tasks. Carrying out such tasks may be very stressful even for well trained professionals. Efficient coordination of rescue team members, as well as efficient allocation of available resources, are key determinants of success. Incident management is typically planned, coordinated, and monitored from rescue centres by human coordinators. Rescue operation planning involves a series of decisions such as which team member will do what task, whether a team should reach an incident location by car or by helicopter, and so on. Making these decision "manually" can be inefficient and error prone, even for well experienced professionals. Automated rescue operation planning can both be used to assist people during an actual live incident and to provide simulation for training exercises. The technological requirements for either case are more or less the same; the system must be able to represent objects with various static and temporal properties, i.e., anything from an unchanging building or a vehicle whose only change is from position x to y (and associated energy consumption) to an accident victim whose vitals are continuously changing via highly complex processes and a tornado causing vast changes to the environment and conditions in a short amount of time. Planning thus involves making decisions about events over which rescue teams have control, while being constrained by those out of their control.

Existing approaches for rescue operation planning (and execution) consist of coordination of (heterogeneous) autonomous vehicles to perform given tasks (e.g. surveillance of some area) (Doherty and Heintz 2011; George, Sujit, and Sousa 2011). Plans are often sequences of waypoints that are passed to the autonomous vehicles, and emphasis is given on "low-level" planning, i.e., planning of elementary vehicles' manoeuvres. For incident management planning, "high-level" planning (e.g. releasing a trapped victim) is sufficient. The most related "high-level" planning work is about road traffic incident management (Shah et al. 2013), which also inspired our work. Incident management is more general, although abstract concepts are similar to those in traffic incident management.

In this paper, we present our ongoing work involving automated planning in incident management. Typically, incidents are managed by coordinators whose decisions are based on experience and are made in a semi-reactive way. Automated planning can provide a complete overview on the incident management task, i.e., from the initial situation when incident(s) are reported to the goal situation when incidents are successfully dealt with. This is especially useful when the coordinator has to deal with situations that are unusual or when training of some specific skills

of rescue teams is designed. Automated planning can be straightforwardly used in incident management since each elementary decision the incident coordinator must take (e.g. a *paramedic1* boards the *ambulance2*) can be formalised as an action. Plans generated by planning engines provide the coordinator (partially ordered) sequences of elementary decisions that must be taken during the rescue operations. However, these plans must be provided quickly (a few seconds latency at most) and in a good quality (nearly optimal), otherwise it might have a negative impact on success of these operations. We have derived a set of requirements, in consultation with experts, from which we provide a formal specification of the domain. Following the specification we have developed a prototype domain model using which we are able to generate plans quickly and in reasonable quality as we empirically verified. Of course, incident management control requires a complex system involving planning and execution episodes overseen by human coordinators. We have designed an architecture of such a system and have identified several challenges, mostly due to uncertainty, that we also discuss in this paper.

Related Work

Search and Rescue (SAR), which can be understood as a subset of incident management, is the search for and provision of aid to people in need (often after a disaster). SAR is a well established research topic. The research in this area is promoted by the Robocup-Rescue project[1] that was motivated by earthquake in Kobe in 1995. This led to the RoboCup Rescue Robot and Simulation competitions that have been held since 2000 (Akin et al. 2013). Most of the SAR approaches consist of (semi)-autonomous robotic systems that are especially useful in "high-risk" areas such as Fukushima after the 2011 incident (Sato, Muraoka, and Hozumi 2014). Besides systems coordinating multiple heterogeneous vehicles (Doherty and Heintz 2011), a system supporting human-robot teams in disaster management scenarios has been recently developed and deployed (Kruijff et al. 2014). SAR often takes place in military operations. Comirem (Smith, Hildum, and Crimm 2005), a system incorporating mixed-initiative planning and scheduling in order to allocate resources more efficiently and with strong emphasis to the user interface, has been applied to Special Operational Forces planning.

Traffic accident management, which deals with situations after traffic accidents in order to provide aid to involved victims as well as to restore the situation into normal, can be also understood as a subset of incident management. Here, the need is to coordinate a team of human agents (paramedics, policemen etc.) rather than robots, so there might not be a need for "low level" control. Applying AI planning in Traffic accident management has been studied from the perspective of stochastic integer programming (Ozbay et al. 2013) as well as of "classical" domain-independent planning (Shah et al. 2013).

[1] http://www.robocuprescue.org/

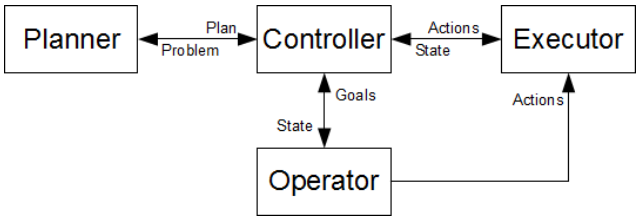


Figure 1: An architecture of an incident management system

Incident Management and Control

A system for incident management and control requires a planning/execution platform that is overseen by a human coordinator. Similar existing platforms such as T-REX (Rajan and Py 2012), which is run on-board of autonomous vehicles, or NEPTUS (Dias et al. 2006), which is a command and control system for autonomous vehicles, provide a framework for planning and executing operations for these vehicles. In our case, we do not need a "low-level" planning and control for coordinating autonomous vehicles, since human agents (rescue team members) are able to autonomously execute "high-level" actions (e.g. drive from one place to another). Our system architecture, depicted in Figure 1 is inspired by the architecture of T-REX.

The *Planner* component consists of one or more planning engines and one or more domain models. It receives the planning problem descriptions from the Controller component and provides solution plans back to it.

The *Executor* component consists of rescue team members who execute actions provided by the Controller component or the human coordinator (the Operator component). During the operation (executing the actions) rescue team members inform the Controller about their progress (e.g. whether the action has been successfully completed) as well as about the state of the environment (e.g. whether an incident victim is seriously injured).

The *Operator* component refers to a human expert who monitors the operations, setting up goals and, eventually, gives orders to the rescue team members (the Executor component). The latest gives the human coordinator to overrule the system in case of some unexpected event, emergency or error in the system.

The *Controller* component is the 'brain' of the system. It receives information about the operation progress from the Executor component and stores information about the current state of the environment. It accepts mission descriptions (goals) from the Operator component from which it generates planning problems that are sent to the Planner component. Plans retrieved from the Planner component are dispatched to the Operator and Executor components. The Controller component also monitors whether execution of plans matches what has been planned. In case of discrepancies, or intervention from the Operator component, re-planning is triggered.

Domain Specification

Given the requirements from experts, we provide a formal conceptualisation of the incident management domain. The way how the conceptualisation is made was inspired by work of Shah et al. (2013). The ontology has three main categories: object types, relation and function types, and action types. The incident management domain is specified on an abstraction and expressiveness level that is relatively independent of specific scenarios and planning engines. We will consider time in our specification, so hereinafter let T be a set of time-stamps that refer to a given scenario (a planning episode). Hereinafter, the constant $\perp$ will refer to an 'undefined value'.

The main abstract *object types* in the incident management domain are as follows:

- *Assets* $X = X_s \cup X_m$ are divided into two categories, *static assets* X_s (e.g. buildings) and *mobile assets* X_m (e.g. vehicles). Assets can accommodate *artefacts* or *agents* (see below).
- *Agents* A are intelligent entities that can interact with assets or artefacts (e.g rescue team members).
- *Artefacts* Y are various objects that cannot act on their own (e.g. incident victims, first aid kits). Artefact can be loaded to assets or carried by agents.
- *Locations* L are 'points of interest' (e.g towns, incident locations).

Notice that agents were not considered in work of Shah et al., since it was assumed that agents are "connected" to assets (e.g. a paramedic is always in an ambulance). Here, we distinguish between agents and assets, since an asset can accommodate agents of various capabilities (e.g. a helicopter can take paramedics, policemen or rescuers).

The main abstract *relation and function types* in the incident management domain are as follows:

We define a relation $conn \subseteq (X_m \cup A) \times L \times L$ for determining whether a mobile asset or agent can move between locations. We define a function $at : (X \cup A \cup Y) \times T \rightarrow L \cup \{\perp\}$ referring to a position of an object in a given time-stamp. Clearly for a static asset the function at is constant (does not change its value with time). We define a function $in : Y \times T \rightarrow (X \cup A \cup \{\perp\})$ referring to situations when an artefact is loaded to an asset or carried by an agent (or $\perp$ if not). Similarly, we define a function $inside : A \times T \rightarrow X \cup \{\perp\}$ referring to whether an agent is inside some asset or not ($\perp$). Each asset or agent may have a limited *capacity* which, informally said, stands for a maximum number (or weight) of loaded or carried artefacts (or agents) in the same time. Formally, a capacity is defined as a function $cap : X \cup A \rightarrow \mathbb{N}$, *volume* of agents or artefacts is represented by a function $vol : A \cup Y \rightarrow \mathbb{N}$, and, finally, *fullness* of agents or assets in a given time-stamp is defined as a function $full : (X \cup A) \times T \rightarrow \mathbb{N}$.

We also define *properties* as functions characterising a state of an object in a given time-stamp. Properties can reach values that are specific for a given class of objects. For example, a 'status' of an incident victim might have one of the following values: GREEN (no injury), YELLOW (minor injuries), RED (severe injuries), BLACK (dead). Agents have *knowledge* about the environment that can be obtained by observation or communication with other agents. Let a finite set K be an *universal knowledge base*. We define a function $k : A \times T \rightarrow 2^K$ referring to a knowledge of agents.

The environment specified by object, relation and function types can be modified by actions, specified via preconditions (what must be met in order to apply the action) and effects (what is changed in the environment after applying the action). We define the following *action types* which modify the environment of the incident management domain. We assume that the action is applied in a time-stamp t and lasts for Δt time.

move(e, l_1, l_2) moves a mobile asset or agent $e \in X_m \cup A$ from a location l_1 to a location l_2 ($l_1, l_2 \in L$). As a precondition it must hold that $(e, l_1, l_2) \in conn$ and $at(e, t) = l_1$. The effect of applying the action is that $at(e, t + \Delta t) = l_2$ and $\forall t' \in (t, t + \Delta t) : at(e, t') = \perp$.

load(z, x, l) loads an artefact or agent $z \in Y \cup A$ to an asset $x \in X$ in a location $l \in L$. As a precondition it must hold that $at(z, t) = l, \forall t' \in [t, t+\Delta t] : at(x, t') = l$ and $\forall t' \in [t, t + \Delta t) : full(x, t') \leq cap(x) - vol(z)$. The effect of applying the action is that $in(z, t + \Delta t) = x$ (resp. $inside(z, t + \Delta t) = x$), $\forall t' \in (t, t + \Delta t) : in(z, t') = \perp$ (resp. $inside(z, t') = \perp$), $\forall t' \in (t, t+\Delta t] : at(z, t') = \perp$ and $full(x, t + \Delta t) = full(x, t) + vol(z)$.

unload(z, x, l) unloads an artefact or agent $z \in Y \cup A$ from an asset $x \in X$ in a location $l \in L$. As a precondition it must hold that $\forall t' \in [t, t + \Delta t] : at(x, t') = l$ and $in(z, t) = x$ (resp. $inside(z, t) = x$). The effect of applying the action is that $at(z, t + \Delta t) = l$, $\forall t' \in (t, t+\Delta t] : in(z, t') = \perp$ (resp. $inside(z, t') = \perp$) and $full(x, t + \Delta t) = full(x, t) - vol(z)$.

fetch(y, a, l) allows an agent $a \in A$ to fetch an artefact $y \in Y$ in a location $l \in L$. As a precondition it must hold that $at(y, t) = l$, $\forall t' \in [t, t + \Delta t] : at(a, t') = l$ and $\forall t' \in [t, t + \Delta t) : full(a, t') \leq cap(a) - vol(y)$. The effect of applying the action is that $in(y, t + \Delta t) = a$, $\forall t' \in (t, t + \Delta t) : in(y, t') = \perp$ and $\forall t' \in (t, t + \Delta t] : at(y, t') = \perp$.

drop(y, a, l) drops an artefact $y \in Y$ carried by an agent $a \in A$ in a location $l \in L$. As a precondition it must hold that $\forall t' \in [t, t + \Delta t] : at(a, t') = l$ and $in(y, t) = a$. The effect of applying the action is that $at(y, t+\Delta t) = l$, $\forall t' \in (t, t + \Delta t] : in(y, t') = \perp$ and $full(a, t + \Delta t) = full(a, t) - vol(y)$.

communicate(a_1, a_2, m) sends a message $m \in K$ from an agent a_1 to an agent a_2 ($a_1, a_2 \in A$). As a precondition it must hold that $m \in k(a_1, t)$ and communication must be possible between a_1 and a_2 in a time interval $[t, t + \Delta t]$. The effect of applying this action is that $k(a_2, t + \Delta t) = k(a_2, t) \cup \{m\}$.

observe(a, m) allows an agent $a \in A$ to get an elementary knowledge $m \in K$ by observing the environment. The effect of applying this action is that $k(a, t + \Delta t) = k(a, t) \cup \{m\}$.

```
(:durative-action firstaid
 :parameters (?p - paramedic
              ?f - firstaidkit
              ?v - victim
              ?l - location)
 :duration (= ?duration 20)
 :condition (and
    (over all (at ?p ?l))
    (over all (at ?v ?l))
    (over all (in ?f ?p))
    (at start (available ?p))
    (at start (injured ?v)))
 :effect (and
    (at start (not (available ?p)))
    (at end (available ?p))
    (at end (not (injured ?v)))
    (at end (aided ?v)))
)
```

Figure 2: The First-aid action in PDDL.

interact$(a, l, \dots)$ changes a property (or properties) of an object (or objects). At least one agent ($a \in A$) must be involved and be present in a location l ($l \in L$), i.e., $\forall t' \in [t, t + \Delta t] : at(a, t') = l$. Also, a cannot be simultaneously involved in another interact action.

From the description of the action types we can derive the following invariants: $\forall y \in Y, \forall t \in T : at(y, t) = \perp \Leftrightarrow in(y, t) \neq \perp$, $\forall a \in A, \forall t \in T : inside(a, t) \neq \perp \Rightarrow at(a, t) = \perp$ (the opposite implication does not hold because the agent might be 'on the way'), $\forall z \in A \cup X, \forall t \in T : full(z, t) \leq cap(z)$, and $\forall z \in A \cup X, \forall t \in T : full(z, t) = \sum_{\{x|in(z,t)=x \vee inside(z,t)=x\}} vol(x)$.

Clearly, the above specification does not include all possible constraints – further constraints relate to specific assets, agents or artefacts. For example, incident victims (as a subclass of artifacts) can be loaded to ambulances and cannot be loaded to fire trucks (both are subclasses of mobile assets). The interact action type is specified in a very general way here, generalising a heterogeneous set of actions (e.g. giving first aid, extinguishing fire etc.)

A fundamental challenge in incident management is dealing with the situation immediately after an incident (or incidents) has been reported. The goal of a incident management team or coordinator is to restore the situation to the normal order (e.g. providing first aid to incident victims and transporting them to hospitals).

Prototype Domain Model

Following the domain specification given above we have developed a prototype domain model. The domain is developed in PDDL 2.1 (Fox and Long 2003), following the same requirements as in the Temporal Track on the International Planning Competitions, as these requirements are widely supported by temporal planning engines.

In our domain model, we consider the following types of *static assets*: Hospital, Fire station, Rescue station, Police station and Building; *mobile assets*: Ambulance, Police car, Fire truck, Rescue car and Helicopter; *agents*: Paramedic, Fireman, Policeman, Rescuer; and, finally, *artefacts*: Victim, First-aid-kit, Extinguisher, Rescue equipment.

```
...
0.09: (move ambulance0 reykjavik hella) [50]
1.10: (board policeman0 helicopter0 reykjavik) [1]
1.11: (fetch paramedic0 firstaidkit2 reykjavik) [1]
1.12: (fetch paramedic1 firstaidkit0 reykjavik) [1]
1.13: (fetch rescuer0 rope0 reykjavik) [1]
1.14: (fetch rescuer1 rope1 hella) [1]
2.15: (move helicopter0 reykjavik landmanalaguar) [15]
2.16: (firstaid paramedic0 firstaidkit2 victim4 reykjavik) [20]
2.17: (firstaid paramedic1 firstaidkit0 victim1 reykjavik) [20]
2.18: (rescue rescuer0 rope0 victim0 reykjavik) [20]
2.19: (rescue rescuer1 rope1 victim2 hella) [20]
17.20: (debark policeman0 helicopter0 landmanalaguar) [1]
18.21: (move helicopter0 landmanalaguar hekla) [5]
18.22: (secure policeman0 landmanalaguar) [10]
...
```

Figure 3: A fragment of a sample plan.

We have also defined the following properties that come on top of the relations and function defined in the previous section. Assets can be *onfire* or *extinguished*. Locations can be *unsecured* or *secured*. Finally, victims can be *injured* or *aided*, and *trapped* or *released*.

Actions are derived from the action types introduced in the previous section. In case of action types *move, unload, fetch* and *drop*, actions in our domain model are encoded straightforwardly according to descriptions of the action types. For the action type *load* we have implemented two variants of actions. One is specific for victims, i.e., victim can be loaded to a mobile asset only if the victim is *released* and *aided*; the other is general and encoded straightforwardly from the *load* action type. We have implemented four actions that extends the *interact* action type:

secure(p, l) allows a policeman p to secure a location l, i.e., the property of l changes from *unsecured* to *secured*.

extinguish(f, e, x, l) allows a fireman f to extinguish an asset x by an extinguisher e in a location l. It must hold that $\forall t' \in [t, t + \Delta t] : at(x, t') = l \wedge in(e, t') = f$. The effect is that the property of x changes from *onfire* to *extinguished*.

firstaid(p, fa, v, l) allows a paramedic or rescuer p to give a first aid to a victim v using a first-aid-kit fa in a location l. It must hold that $\forall t' \in [t, t + \Delta t] : at(v, t') = l \wedge in(fa, t') = p$. The effect is that the property of v changes from *injured* to *aided*. The PDDL encoding of this action is depicted in Figure 2.

release(r, re, v, l) allows a rescuer r to release a victim v using a rescue equipment re in a location l. It must hold that $\forall t' \in [t, t + \Delta t] : at(v, t') = l \wedge in(re, t') = r$. The effect is that the property of v changes from *trapped* to *released*.

Notice that action types *sense* and *communicate* are not implemented. This is due to the requirements for a deterministic and fully observable environment. Clearly, sensing and communication are needed for environments that are partially observable. Moreover, it creates contingency (non-deterministic action effects). Such issues are discussed later.

	LPG-td		Yahsp3-MT	
	T	Q	T	Q
Small	0.62±0.42	116±24	0.16±0.18	143±27
Medium	3.22±1.82	178±63	0.11±0.10	249±63
Large	5.45±1.79	419±83	0.39±0.39	384±86

Table 1: Results showing runtime of the planners in seconds (T) and quality (make-span) of the firstly generated plans (Q) with respect to different size of problems.

	Small	Medium	Large
Time	1.35±2.10	4.25±3.38	3.60±2.54
Quality	109±20	197±41	313±47

Table 2: Results showing runtime of Yahsp3-MT in seconds and quality (make-span) of the highest quality generated plans with respect to different size of problems.

Our prototype domain model deals with the following incidents: an asset in fire, a location to be secured from public, and injured and/or trapped victims. The goal is to manage these incidents such that: the asset in fire is extinguished, the location is secured, and the victim is rescued and eventually transported to the hospital (if his/her injury is serious). A fragment of a sample plan is depicted in Figure 3.

Experimental Evaluation

For the experimental evaluation of our approach we used a scenario consisting of: 5 locations, 1 hospital, 1 ambulance, 1 helicopter, 2 of each of the remaining types of assets, 2 of each agents, 3 first-aid-kits, 2 extinguishers and 2 units of rescue equipment. Each agent is 'loaded' to "its" asset (e.g. a police man is in the Police station) and mobile assets are in the same locations as "their" static assets (e.g. an ambulance is in the same location as the hospital). Then, we randomly select some locations where we set their property to 'unsecured', generate n victims and distribute them randomly along the locations with randomly assigned properties to *injured* and/or *trapped*. Finally, we generate k buildings randomly distributed along the locations with properties set to *onfire*. The goal is to change all unsecured locations to secured, buildings being on-fire to extinguished, victims being trapped to released, and finally, victim changed from being injured to aided and some of them to be delivered to the hospital.

We defined 3 classes of problems: "Small" ($n = 5$, $k = 2$), "Medium" ($n = 10$, $k = 3$) and "Large" ($n = 20$, $k = 5$). We generated 5 problems per each class and for solving them we used four state-of-the-art planning engines: Yahsp3-MT (Vidal 2014), LPG (Gerevini, Saetti, and Serina 2003), Popf2 (Coles et al. 2010) and Optic (Benton, Coles, and Coles 2012). We have observed that Popf2 as well as Optic do not scale well in our domain model, so the time needed to find solutions considerably increases with problem size. On the other hand, LPG as well as Yahsp scales well and time needed to extract the first solution is within a few seconds even for the large instances as shown in Table 1. LPG and Yahsp thus comply with one of the required criteria – obtain solutions in at most a few seconds. However, quality of first solutions is often not very good. On the other hand, both LPG and Yahsp can incrementally improve solutions, so it is possible to obtain solutions of better quality. We took a closer look on Yahsp, where we have observed that solutions can improve considerably, as depicted in Table 2, even while keeping quite strict cutoff of 10 seconds.

Our domain model does not require concurrency, so it is possible to solve problems as classical ones and then schedule actions from the plans in order to minimise make-span. This seems to be the main reason that extended classical planners (LPG and Yahsp) performed much better than "purely" temporal planners (Popf2 and Optic).

Given the performance of Yahsp, we can obtain solutions in reasonable quality (however, sub-optimal) in a very little time even for relatively large problems. Generated plans to large extent comply with expectations of the domain experts. This gave us a promising outlook for applying AI planning in incident management. Clearly, it applies for "standard" situations where it is not necessary to involve a large number of entities and/or consider complex cooperative actions (e.g. releasing the victim while giving him/her first aid). For disaster management, where thousands of entities are involved, we might use a different domain model, which is more abstract. For considering cooperative actions, a modified domain model is also required.

Challenges

We have identified several challenges related to application of AI planning in incident management. These challenges can be divided into three categories, namely Task Complexity, Uncertainty and Goal prioritisation. Most of these challenges can possibly be overcome without necessarily changing the domain model we have presented, or introducing more expressiveness (e.g. conformant planning). Testing this belief is planned in our future work.

Task Complexity

In real world scenarios, it is common to have a huge amount of objects being possibly in a plenty of different relations. Hence, we might have an excessive number of possible actions do deal with. AI planning is generally intractable, so it is impossible to handle very large models. However, we do not have to represent everything in very detail, we can either abstract or relax. For example, the firstaid action is a good example of abstraction. Although there are various ways how to give the first aid to incident victims, which mostly depends on what sort of injuries they have, professional paramedics know how to give the first aid (i.e. execute the firstaid action) without need to provide details. Also, we do not have to, for example, consider road traffic, so we can relax it. In normal conditions, it can marginally affect the driving time of rescue vehicles. If traffic is very heavy, we might consider, in the worst case, the road being blocked, and if there is no alternative that we might assume that relevant locations are not connected for the rescue vehicles (we apply abstraction).

Our prototype domain model, therefore, does not have to deal with an excessive number of objects. Problems are

thus relatively easy to solve. Possible issues are related to accuracy of the model and understanding the correct representation of the objects. In our case, most of actions are restricted to a single agent. However, in some case having more agents to perform an action might be more appropriate. For instance, if a victim has a serious injury, then more paramedics might give the victim the first aid. Also, an actual meaning of "rescue equipment" might vary regarding the situation in which it is used. Rescuers might use a different equipment for rescuing a victim in the mountains than in the forest. While the latter issue can be tackled by some sort of meta-reasoning, the former needs an enhanced domain model.

Unsolvable Problems

Solvability of the problem cannot be guaranteed. The problem might become unsolvable if, for example, some location is unreachable, or some kind of artefact is not present (e.g. fire extinguisher). In these cases the system can easily find the reason and notify the mission coordinator, so s/he can remove problematic goals.

Similarly, we cannot guarantee that the solution will be retrieved in a given time limit. The reason might be that the problem is too large (too many goals), or there are some specific constraints that makes the problem too hard for a planner. Although we believe that this is an unlikely scenario, if it occurs, the coordinator has to take over the control. In particular, the coordinator might decide to control the mission manually, or remove some (less important) goals in order to make the problem easier.

Uncertainty

In real world scenarios, unexpected situations may arise just before, or during, the operation being carried out. While there is always a human coordinator who can overrule the system, the system must be able to appropriately react on such situations.

Bad weather is often an issue for incident management. Although weather might be rather unpredictable in long term (days), it can be well predictable for short-term (hours). Since incident management is usually of short-term, weather be considered in the problem description by providing some restrictions (e.g. a helicopter cannot fly to locations affected by T-storms).

It might not be known how long some actions will take. For example, giving first aid to a victim with minor injuries will take much less time than first aid to a victim with severe injuries. However, severity of victim's injuries might not be very well known, since incidents are often reported with many inaccuracies. Therefore, several plans considering different action durations (e.g. optimistic, realistic and pessimistic estimation) might be generated. If these plans vary widely the human coordinator may decide which plan will be executed.

Often, complete information about the environment may not be available and thus we need sensing actions (e.g. "observe location X"). Sensing actions lead to non-deterministic contingencies. To make it deterministic we might consider the most likely outcome of the sensing action. If the actual outcome is different we might re-plan. Alternatively, we might provide plans for every outcome of the sensing actions (unless there are too many possible outcomes).

Goal Prioritisation

Another problem might be prioritising of some goals. It might be the case, for example, that life of one of the victim is in danger, so the victim has to be rescued as quickly as possible. If there are more victims in different locations the planner might not consider priorities while planning when it optimises for 'make-span'. Of course, there is a possibility to put priorities into the problem definition and force the planner to optimise for them. However, such a feature is not widely supported by planners. A possible solution to the prioritisation issue is to isolate goals with a very high priority and generate plans achieving only these goals. The remaining goals can be achieved in a separate plan, where, of course, objects (e.g. agents) that are involved in the former plan will not be used.

Conclusion

In this paper, we have derived a set of requirements from which we have conceptualised a formal specification of the incident management domain. Following the specification we have developed a prototype domain model and evaluated it by using state-of-the-art planning engines. We have shown that it is possible to solve "common-size" problems in a reasonable time (up to a few seconds) and in decent quality (short 'make-span'), so the preliminary results indicated a promising direction of our work. We have identified several challenges, most of them related to uncertainty issues, and showed how to deal with most of them without the necessity to extend our model as it stands to support more expressive features. On the other hand, in situation with higher level of uncertainty it might be useful to exploit probabilistic planning (state-of-the-art probabilistic planning engines accept domain and problem specification in RDDL (Sanner 2011)). Developing an RDDL domain model will be our future work.

This work being intended as part of a larger system, we plan to integrate our domain model (and some planning engines) into a planning and execution simulation framework. Then we plan to test it on some real cases in order to determine how plans retrieved by planning engines using our domain model differ from plans provided by mission coordinators. We believe that it will provide us with further insights that will help us to develop and deploy the system.

Acknowledgements

This work is funded by University Research Fund of University of Huddersfield under contract number URF2014_17.

References

Ai-Chang, M.; Bresina, J. L.; Charest, L.; Chase, A.; Hsu, J. C.; Jónsson, A. K.; Kanefsky, B.; Morris, P. H.; Rajan, K.; Yglesias, J.; Chafin, B. G.; Dias, W. C.; and Maldague, P. F. 2004. MAPGEN: mixed-initiative planning and scheduling

for the mars exploration rover mission. *IEEE Intelligent Systems* 19(1):8–12.

Akin, H. L.; Ito, N.; Jacoff, A.; Kleiner, A.; Pellenz, J.; and Visser, A. 2013. Robocup rescue robot and simulation leagues. *AI Magazine* 34(1):78–86.

Benton, J.; Coles, A. J.; and Coles, A. 2012. Temporal planning with preferences and time-dependent continuous costs. In *Proceedings of the Twenty-second International Conference on Automated Planning and Scheduling (ICAPS-12)*.

Coles, A. J.; Coles, A. I.; Fox, M.; and Long, D. 2010. Forward-chaining partial-order planning. In *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling (ICAPS-10)*.

Dias, P. S.; Gomes, R. M. F.; Pinto, J.; Gonçalves, G. M.; de Sousa, J. B.; and Pereira, F. L. 2006. Mission planning and specification in the neptus framework. In *Proceedings of the 2006 IEEE International Conference on Robotics and Automation, ICRA 2006, May 15-19, 2006, Orlando, Florida, USA*, 3220–3225.

Doherty, P., and Heintz, F. 2011. A delegation-based cooperative robotic framework. In *2011 IEEE International Conference on Robotics and Biomimetics, ROBIO 2011, Karon Beach, Thailand, December 7-11, 2011*, 2955–2962.

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

George, J.; Sujit, P. B.; and Sousa, J. B. 2011. Search strategies for multiple UAV search and destroy missions. *Journal of Intelligent and Robotic Systems* 61(1-4):355–367.

Gerevini, A.; Saetti, A.; and Serina, I. 2003. Planning through stochastic local search and temporal action graphs in lpg. *Journal of Artificial Intelligence Research (JAIR)* 20:239–290.

Kruijff, G. M.; Kruijff-Korbayová, I.; Keshavdas, S.; Larochelle, B.; Janícek, M.; Colas, F.; Liu, M.; Pomerleau, F.; Siegwart, R.; Neerincx, M. A.; Looije, R.; Smets, N. J. J. M.; Mioch, T.; van Diggelen, J. V.; Pirri, F.; Gianni, M.; Ferri, F.; Menna, M.; Worst, R.; Linder, T.; Tretyakov, V.; Surmann, H.; Svoboda, T.; Reinstein, M.; Zimmermann, K.; Petrícek, T.; and Hlavác, V. 2014. Designing, developing, and deploying systems to support human-robot teams in disaster response. *Advanced Robotics* 28(23):1547–1570.

Ozbay, K.; Iyigun, C.; Baykal-Gursoy, M.; and Xiao, W. 2013. Probabilistic programming models for traffic incident management operations planning. *Annals OR* 203(1):389–406.

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

Rajan, K., and Py, F. 2012. T-rex: partitioned inference for auv mission control. *Further advances in unmanned marine vehicles. The Institution of Engineering and Technology (IET)*.

Sanner, S. 2011. Relational dynamic influence diagram language (RDDL): Language description. Technical report, NICTA and the Australian National University.

Sato, M.; Muraoka, K.; and Hozumi, K. 2014. Flight control design and demonstration of unmanned airplane for radiation monitoring system. In *19th World Congress of The International Federation of Automatic Control (IFAC)*.

Shah, M. M.; Chrpa, L.; Kitchin, D.; McCluskey, T. L.; and Vallati, M. 2013. Exploring knowledge engineering strategies in designing and modelling a road traffic accident management domain. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence (IJCAI)*, 2373–2379. AAAI Press.

Smith, S. F.; Hildum, D. W.; and Crimm, D. R. 2005. Comirem: An intelligent form for resource management. *IEEE Intelligent Systems* 20(2):16–24.

Vidal, V. 2014. Yahsp3 and yahsp3-mt in the 8th international planning competition. In *Proceedings of the 8th International Planning Competition (IPC-2014)*.

Extending an Online (Re)Planning Platform for Crop Mapping with Autonomous UAVs through a Robotic Execution Framework

Alexandre Albore[†‡] and **Nathalie Peyrard**[†] and **Régis Sabbadin**[†] and **Florent Teichteil-Königsbuch**[‡]
† INRA UR 875 (MIA), chemin de Borderouge, 31326 Castanet-Tolosan, France - *name.surname*@toulouse.inra.fr
‡ ONERA (DCSD) - 2 avenue É. Belin, 31000 Toulouse, France - *name.surname*@onera.fr

Abstract

Maps of plant pests are widely used to support farmer decisions to manage production at the scale of crop fields. Such maps are generally obtained manually, by human annotators or with human-controlled Unmanned Aerial Vehicles (UAVs), but this process is slow and costly. We propose an AI planning approach to fly fully autonomous UAVs equipped with on-line sequential decision-making capabilities for pests sampling and mapping in crop fields. We use a Markov Random Field framework to represent knowledge about the uncertain map and its quality, in order to compute an optimised pest-sampling policy. Since this planning problem is PSPACE hard, thus too complex to be solved exactly, we thus interleave planning and execution, generating plans from a subset of sampling sites selected.

This approach has already been proved to be successful (Albore et al. 2015), favourably comparing with existing methods, but encounters some computation limits due to this division of tasks, considering that the planning execution framework is not adapted to the anytime-like behavior needed by real-world applications. We discuss the next steps in developing our approach, namely integrating the planning process and calculus of probabilities distribution in a framework able to deal with task management and execution under time constraints. Such extension, and integration within the AMPLE robotic execution framework, is promising as it associates the success of the replanning approach to the flexibility of an anytime executing architecture.

Introduction

An important tool in precision agriculture for supporting the management of production in crop fields is a map of pest abundance spatial distribution, i.e. a map of the spatial distribution of the observed species. Such maps are costly to obtain since they require intensive surveys of pests in the field, until now mainly performed by human annotators; under these conditions, the whole field cannot generally be inspected and a complete abundance map is estimated from samples in sub areas of the field. Remote sensing tools are emerging as a promising alternative due to their flexibility to gather information on large areas. So far, the primary remote platforms for collecting images in agriculture have been piloted aircraft and satellites, but they are giving way to Unmanned Aerial Vehicles (UAVs) that provide a better spatial and temporal resolution for the image analysis at a lower cost, and offer an ideal point of view for the acquisition of ground-based data. Moreover, UAVs can operate below cloud covers – which is impossible for higher altitude aircraft and satellites – and can be deployed quickly and repeatedly, which permits an almost continuous monitoring of an area. Even with UAVs, a sampling strategy has to be determined because of the limited flight time, and the extension of the area to map, generally too big for an exhaustive survey.

Autonomous systems are well-suited to exploit dynamic information from images and to in-flight optimisation of navigation between areas to sample. The same operations cannot be easily performed by piloted UAVs, as computations and heuristic evaluations to choose the best trajectory and altitude for optimal pest sampling are generally non trivial. So, the next challenge is to use completely automated UAVs with on-board computation capabilities. These UAVs should be deployed on demand, without requiring heavy logistics as side effects of autonomy capabilities; this includes expensive ground computational units, training, or optimisation of exploration strategies before the flight. Therefore, strategy optimisation must be conducted on-board, during the flight, at a low computational cost: farmers actually expect automated techniques to be as easy as possible, with UAVs ready to be used.

Adaptive sampling techniques have been developed in the context of manual sampling done by humans, with the purpose of mapping invasive species or weeds in large areas (Peyrard et al. 2013; Bonneau et al. 2014). These approaches rely on a Markov Random Field model (MRF) (Geman and Geman 1984) of the abundance map to estimate, and on methods for sequential decision-making under uncertainty. Existing techniques that deal with the task of crop mapping, and that as well make use of MRFs, fail in two main aspects, motivating our approach based on automated planning. In (Peyrard et al. 2013), observations result from sampled sites chosen *greedily*, without considering the future sampling steps, which result in a fast adaptive method where the next-to-be-applied action depends on the history of previous observations, but that does not optimise resources such as remaining flying time, which in turn have an impact on the quality of the resulting map, and has no lookahead capaci-

ties whatsoever.

The Reinforcement Learning approach proposed by (Bonneau et al. 2014) extends the previous one by including in the adaptive sampling design the full sampling horizon and the sampling budget; the problem is cast as a Markov Decision Processes (MDPs) with state variables corresponding to the list of observed variables so far, and consequently requires very long off-line computation time to preliminarily compute the parameters of the sampling policy. This does not match our needs for UAVs to be deployed on-demand, efficiently performing pest sampling and mapping with light logistics.

The planning approach that we adopted and describe here has gathered good results compared to these preexisting techniques (Albore et al. 2015), showing an efficient trade-off between quality of selected sampling sites, and planned navigation within the flight-time limits imposed by the battery life. In order to achieve this challenging time-constrained objective, we dissociated the problem of selecting the observation sites from the one of planning their visiting order. The result is a non-greedy method for sampling in MRFs, in opposition to (Peyrard et al. 2013), that nevertheless does not require as much computation time as (Bonneau et al. 2014).

Note that the planning model we adopt is still not a variant of the well-known travelling salesman problem, even if there are some points in common; here, the information obtained when sampling a plot has the side effect of reducing the expected rewards from neighbouring plots. This impact is to be considered when producing a visiting order, and a planner is more suitable to heuristically consider such causal links: namely, observing a given site prevents from exploring nearby locations because it may not substantially improve the knowledge of the map.

Even if the resulting framework based on interleaving planning and execution has been successful in the task of mapping pests with an autonomous UAV, there are certain limitations due to the decoupling approach adopted. In fact, the elaboration of the expected quality gains from the MRF, which is required to select informative enough sampling sites, is an expensive operation that often interrupts the flight of the UAV, besides slowing down the task. We plan to overcome this hindrance in two ways. First, by reimplementing the Loopy Belief-Propagation (LBP) algorithm (Murphy, Weiss, and Jordan 1999) to approximate inference in the MRF, restricting the marginal updates only to those variables actually affected by any perturbation in the MRF. Second, by adopting an execution model that allows to continually improve an incomplete partial policy over time, like in anytime probabilistic methods, but enforcing to query the policy before completion of the optimisation at any time and stop the current planning instance in order to replan from a new execution state, or plan from possible future outcomes of the plan while executing the current action. This would mean to elaborate the expected gains from sampling sites, and therefore select the most interesting sites, while continuing to move and gather new observations, and to dispose of several planning strategies to switch between, depending on the UAV's status. The approach relies on AMPLE (Chanel, Lesire, and Teichteil-Königsbuch 2014), a configurable anytime meta-planner that drives our framework, dealing with all pending and time-bounded planning requests sent by the execution layer from many reachable possible future execution states, in anticipation of the probabilistic evolution of the system.

In the following, we first recall the modelling of the problem of optimal pest sampling for mapping in MRF as described in (Bonneau et al. 2014) and see why the existing solutions are not suitable for on-line mapping with UAVs. We then present our original approach based on interleaving planning and execution, and show its performances, empirically illustrated on a problem of weeds sampling in crop fields, comparing favourable in terms of quality and resource consumption to the greedy approach, which is the only existing on-line solution to this problem. We finally describe the AMPLE robotic execution framework and our integration of the automated crop mapping architecture within it.

Modelling UAV-based sampling as a sequential decision-making problem under uncertainty

A map is divided into a regular grid of N plots of ground (or sites) of small area. Observing a plot provides the weed abundance there (discretised in K classes) and we assume no measurement error. Getting observations from all the sites of the field is impossible, because of the UAV's limited battery capacity, so only a sample of the total plots can be observed, and from this sample we want to provide a full map of the field by estimating the value at unobserved sites. To do so, we design an adaptive sequence of sampled plots (an adaptive plan) that maximises the quality of the estimated map, reconstructed from the gathered observations, under UAV's physical constraints. An adaptive plan implies that the sequence of plots is not defined beforehand, and the next site where to sample may depend on the history of previous observations (positions and values) and it is determined dynamically. We follow the work of (Peyrard et al. 2013) where MRF are used to model weed abundance maps distributions, and the definitions of estimation and quality are based on the remaining uncertainty in the estimated complete map. This representation of spatial phenomena through MRF is general and can be used for a variety of data collected from sensors.

MRF modelling of abundance maps

The MRF model for abundance map is defined as follows. To each site $i \in V = \{1, \dots, N\}$ of the field is attached a discrete random variable X_i with domain $D = \{1 \dots K\}$, where K is the number of abundance classes. The joint distribution of the whole map $X = (X_1, \dots, X_N)$ is assumed to be expressed as a pairwise MRF: $\forall x \in D^N$,

$$\mathbb{P}(X = x) = \frac{1}{Z} \prod_{i=1}^{N} f_i(x_i) \prod_{(i,j) \in E} f_{i,j}(x_i, x_j) \quad (1)$$

The set E is the set of all pairs of order 1 neighbours in the grid of sites and Z is a normalising constant. The f_i and $f_{i,j}$

are non negative functions called respectively order 1 and order 2 potential functions. Roughly speaking, the order 1 potential functions weight the relative proportions of the K abundance classes while the order 2 potential functions encode spatial correlation between abundance values at different sites. The choice of an appropriate MRF model amounts to the choice of these potential functions. We will provide an example of such a choice in the case of weeds maps.

Adaptive sampling in MRF

Optimal adaptive sampling in MRF with an objective of map reconstruction has been modelled in (Bonneau, Peyrard, and Sabbadin 2012) as the problem of finding a policy tree that optimises a given quality criterion under sampling budget constraint depending on the distance, the wind direction and force, etc. The *quality* $U(A, x_A)$ of a trajectory A is given by the sum of max marginals over the sequence of observations gathered x_A:

$$U\big(A, x_A\big) = \sum_{i \in V} \Big[\max_{x_i \in D} \Big\{ \mathbb{P}(x_i \mid x_A) \Big\} \Big]. \tag{2}$$

$U(A, x_A)$ can be interpreted as the expectation of the number of well-classified sites, when allocating their values to the modes of the marginals $\mathbb{P}(x_i \mid x_A)$.

The quality of a sampling policy δ is, in those cases, defined as an expectation over all possible trajectories τ_δ (de facto, a policy tree) that can be followed by executing δ:

$$V(\delta) = \sum_{\tau_\delta} \mathbb{P}(x_A) \cdot U(A, x_A). \tag{3}$$

And the optimal sought quality is given by:

$$\delta^* = \arg \max_{c(\delta) \leq B} V(\delta). \tag{4}$$

Here, $c(\delta)$ is defined as the maximum of the costs of all trajectories in τ_δ, and B is a fixed *budget* for sampling. In the UAV case, B is the battery total energy and a policy is admissible only if none of the trajectories it can generate use more energy than available.

Apart from the cost constraint, our problem is similar to a Partially Observable Markov Decision Process with terminal rewards $U\big(A, x_A\big)$, which generally prevents to solve it online on-board the UAV for limited-CPU and time reasons.

The sampling policy optimisation problem defined above is too hard to solve exactly for two independent reasons. First, to determine whether there exists an adaptive sampling policy whose utility is above a given threshold is PSPACE-complete (Bonneau, Peyrard, and Sabbadin 2012), and representing such a policy tree takes exponential space in the problem description. Second, computing the Maximum Posterior Marginals $\mathrm{argmax}_{x_i \in D} \mathbb{P}(x_i \mid x_A)$ for a given sampling observation is #P-complete and NP-hard to approximate (Roth 1996).

As exact inference algorithms like junction tree and Monte Carlo methods, that calculate exact marginal probability distributions, are too computationally expensive for on-board computations, we turn toward approximate methods. Several approaches have been proposed to approximately solve this problem, but none seems to fit the needs of our application-oriented framework. The *greedy* approach mentioned before (Peyrard et al. 2013) fails in considering neither future sampling steps nor the available sampling budget. A *reinforcement learning (RL)* type approach (Barto, Sutton, and Watkins 1989; Bonneau, Peyrard, and Sabbadin 2012), together with a dedicated value-function approximation method, would require excessive "off-line" computation resources to solve the RL problem.

A replanning-based approach for efficient pest sampling and field mapping

We proposed an original approach to solve the optimisation problem stated above, using a replanning framework to generate a pest sampling strategy – in fact a partial MDP policy – synthesised by means of several calls to a classical (deterministic) planner (Albore et al. 2015).

The planning problem of reconstructing a pest abundance spatial distribution map with an UAV is built and executed in a closed-loop fashion. First, we generate a set of n plots to sample that maximise the *expected quality gain*, a quantity defined below and derived from the MRF model. Then we compute a full *plan* to find a trajectory that minimises the navigation cost while visiting all the n sampling sites. This plan consists in a sequence of locations *and* expected observations. The plan is then executed by the UAV, and the observations collected. We monitor the execution so to stop it and recompute a new plan whenever the accumulated difference between the actual observations and the expected ones exceeds a given threshold. In this case, we also update the expected quality gain for all the sites. This replanning approach is close in spirit to *fault-tolerant* planning (Domshlak 2013) and *planning under assumptions* (Albore and Bertoli 2006), with the difference that in this problem, no execution dead ends can occur.

After a given set of observations, (A, x_A), we define, for each variable X_i of the MRF, the *expected quality gain* as an optimistic approximation of the increase of the updated utility $U(A \cup \{i\}, x_A, X_i) - U(A, x_A)$. For a variable X_i, we define the expected quality gain $\bar{q}(X_i, x_A)$ as:

$$\begin{aligned} \bar{q}(X_i, x_A) = & \max_k \Big(\sum_{dist(i,j) \leq r} \max_{x_j} \mathbb{P}(x_j | X_i = k, x_A) \Big) + \\ & - \sum_{dist(i,j) \leq r} \max_{x_j} \mathbb{P}(x_j | x_A). \end{aligned} \tag{5}$$

Sensitivity perimeter Elaborating the marginals $\mathbb{P}(x_j | X_i = k, x_A)$ for any possible observation is generally a costly operation that slowed the approach presented in (Albore et al. 2015). We improved the marginals elaboration modifying the Loopy Belief-Propagation (LBP) algorithm (Murphy, Weiss, and Jordan 1999) to approximate inference in the MRF by considering that the convergence in a Markov Field is governed by the second largest eigenvalue of its transition matrix (the transition matrix is diagonal and always has the first eigenvalue equal to 1), and its "decay rate" is given by the eigenvalue itself. We can compute from the eigenvalues, the subset of variables for which the information changes after each observation above a certain

threshold (1% of the former value). When applying the LBP algorithm, we thus update only the varibles in that *sensitivity perimeter* of the observed variable, regardless the dimension of the field, obtaining a good quality estimation while reducing the calculation time.

A Classical Planning Model for the navigation task

The navigation problem in an unknown field can be modelled as a deterministic planning model with action costs. Such model can be characterised as a tuple $\mathcal{S} = \langle S, s_0, S_G, A, f, c\rangle$ where S is a finite set of states, $s_0 \subseteq S$ is the initial state, S_G is the set of goal states, A is a set of actions with $A(s)$ denoting the actions in A that are applicable in the state s. An action a applicable in a state s changes the state to $s' = f(a, s)$, with $f : A \times S \to S$ the transition function. The cost function c for actions is $c : A \to \mathbb{R}_0^+$.

An action sequence $\pi = a_0, \ldots, a_n$ is applicable in a state s_0 if $a_i \in A(s_i)$, $0 \leq i \leq n$, and there exists a sequence of states $s_0, \ldots, s_{n+1}$, such that $s_{i+1} = f(a_i, s_i)$; in such a case we say that π achieves s_{n+1} when executed in s_0. π is a plan for P if it achieves a state g in G, when executed in the initial state. The cost of a plan π is $c(\pi) = \sum_{a\in\pi} c(a)$.

For the UAV's navigation problem, the planning space consists in states encoding an UAV's pose, and the status of the sites (observed/not observed). The initial state is given by the UAV's initial pose, while the goal is to have performed an observation in all the sites given in a set that maximises the expected quality gain. We consider two kinds of actions: *goto* actions move the agent between neighbour sites, and each *observation* action flags a plot and its neighbours in the field as observed. This causal relation between observed sites and their neighbours is dictated by the result of updating a site i in the MRF with an observation: the max marginal values of the neighbour sites vary more for sites closer to i. This implies that the quality gain expected from observing the value of a site is small if it is close to a plot that has already been observed.

Moving between two adjacent sites has unit cost, which corresponds to measuring the Manhattan distance for distant sites, while sampling has a slightly higher cost, as we consider that stabilising the UAV to take a picture consumes more resource than flying between two adjacent plots.

Interleaving planning and execution The map reconstruction task is separated in two clear parts: (1) selection of observation sites; (2) search of a visiting order that optimises the flying time constraints of the UAV. This decoupling permits to observe and update the knowledge model within a robotic platform in real time while moving to the next site.

To guarantee that distance constraints (reflected in the flying time) are respected, the states that violate the given constraints are pruned from the search space, in a very similar way to what (Ivankovic et al. 2014) do using global numerical state constraints (but without considering them yet in the heuristic evaluation), or what the planner MBP (Bertoli et al. 2001) does with problem invariants coded as the verification of invariant properties in the NuSMV model checker (Cimatti et al. 2000). We use a numerical variable that is updated along with the state and whose value is monitored at planning-time.

On top of this constraint, replanning occurs when the quality requirements are not met at execution-time. We illustrate this behaviour in the following pseudo-code: Routine

Algorithm 1: Main (re)planning loop.

```
1 Function ReplanningLoop(s_0):
      Input: initial state s_0
2     s ⟵ s_0;
3     goals ⟵ ∅;
4     repeat
5         goals ⟵ bestPlots();
6         π ⟵ plan(s, goals);
7         s ⟵ execute(π, s);
8     until π = ∅;
```

Replanning-loop(init) takes the initial state of the planning problem as input and obtains n sampling locations by selecting the sites with the biggest expected quality gain in the MRF at step 5. This list is set to be the goals of the planning problem: at step 6 the planner synthesises a plan consisting in *goto* and *observation* actions. Step 7 updates the current state with the outcome of executing the plan π. The loop repeats until all the sites are visited, or no plan is can be synthesised under the given constraints (step 8).

Routine *Execute(π, s)* applies the actions a in the plan π, and updates the current state, and, if a is an *observation*, updates the MRF as well. This step also updates the accumulated difference in expected quality gain $\bar{q}$, which must remain smaller than a fixed value ϵ, otherwise a replanning episode is triggered.

Empirical evaluation of the platform

We implemented the previously described replanning algorithm applied to the weeds mapping problem within the Robot Operating System (ROS) framework (Quigley et al. 2009), a robotic meta-operating system that provides hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. The evaluation of marginals in the MRF and the planner are integrated on the same platform, taking advantage of our implementation of the LBP algorithm, and the (re)planning loop uses the Lightweight Automated Planning Toolkit (Ramirez, Lipovetzky, and Muise 2014) run with a Serialized Iterated Width algorithm (Lipovetzky and Geffner 2012), that we adapted as a ROS independent planning package, that we made available to the community.

We ran all empirical evaluations using the MORSE simulator for academic robotics (Echeverria et al. 2011), which enables to perform software architecture-in-the-loop (SAIL) realistic simulations, i.e. to test the exact same functional architecture as the one that will be implemented on-board the real UAV, but replacing the physical sensors and actuators by simulated data. Interestingly, we can feed simulated sensors like cameras with real data such as true images, meaning that the image analysis algorithm can deal with the same kind of

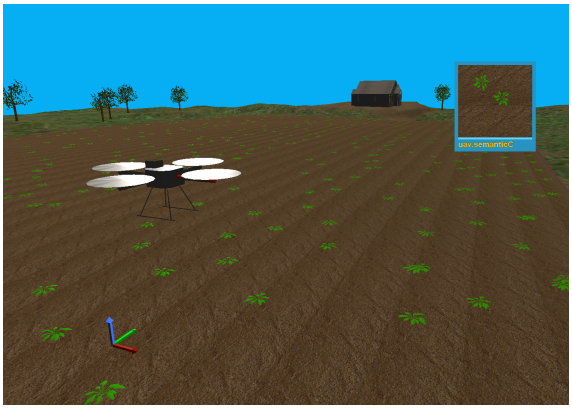


Figure 1: Platform simulation on MORSE. At the upper right corner, the UAV's semantic camera framing weeds.

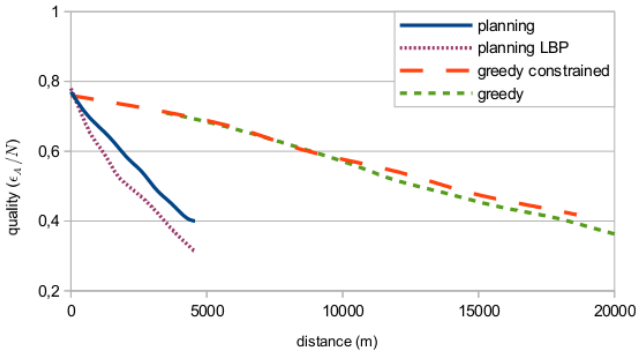


Figure 2: Plot of quality versus distance for model M1. Small dashed line is the initial greedy approach, dashed line is the updated greedy approach, plain line is the replanning approach, dotted line is the planning approach with improved LBP algorithm.

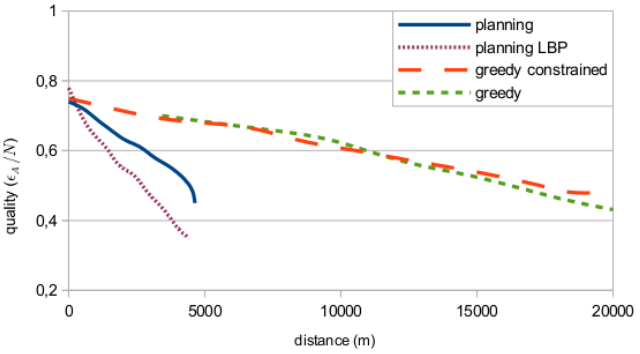


Figure 3: Plot of quality versus distance for model M2. Small dashed line is the initial greedy approach, dashed line is the updated greedy approach, plain line is the replanning approach, dotted line is the planning approach with improved LBP algorithm.

images in the simulation as during the real flight. We get weeds abundance classes in the field plots from the standard semantic camera sensor of MORSE (cf. Fig. 1) set to 50mm focal length. From a planning point of view, SAIL simulations allow us to test in realistic conditions.

We considered crop fields of size corresponding to an average field. A field was divided into a regular grid of 425 plots. Weeds can be structured into patches and depending on the weed species, the crop and the period of the year, these patches can be more extended into tillage direction (Johnson, Mortensen, and Gotway 1996), since dispersion is made easier. Therefore, we considered two MRF models of weeds spatial distribution: an isotropic model (M1) and its anisotropic version (M2).

$$\mathbf{M1}: \log[f_{i,j}(x_i, x_j)] = \beta \cdot \left(1 - \frac{|x_j - x_j|}{K}\right) \ , \ \beta \in \mathbb{R}$$

$$\mathbf{M2}: \log[f_{i,j}(x_i, x_j)] = \beta_t \cdot \left(1 - \frac{|x_i - x_j|}{K}\right) \cdot \mathbf{1}_{(i,j)\in E_t} + \\ + \beta_o \cdot \left(1 - \frac{|x_i - x_j|}{K}\right) \cdot \mathbf{1}_{(i,j)\in E_o} \ , \ (\beta_t, \beta_o) \in \mathbb{R}^2$$

where for model M2, E_t denotes the set of neighbours sites in the direction of tillage, while E_o is the set of neighbours sites in the orthogonal direction. With these two models, assuming no prior information is available about a dominant abundance class in the map, we considered that all order 1 potential functions, $f_i(x_i)$, are equal to one. Then, the maximal order 2 weight is given when neighbouring sites i and j are in the same state, and this weight decreases when the absolute difference between x_i and x_j increases. We considered 4 abundance classes ($K = 4$) and the parameters were fixed to the values of $\beta = 2$, and $\{\beta_t, \beta_o\}$ to $\{4, 1\}$, corresponding to realistic values for weeds maps split in plots of $9m^2$. 200 abundance maps of weeds were generated using the Gibbs sampling algorithm (Koller and Friedman 2009) for model M1 and for model M2. On each map we applied our on-line planning approach and we compared the map estimated from data sampled during the UAV trajectory with the real one.

We compared the performances of our replanning platform to the greedy approach of (Peyrard et al. 2013) on simulated crop fields. We show the figures for the distances and the corresponding quality in Figs.2 and 3, comparing the four approaches on model M1 and M2, with a numerical constraint on distance of 4500m, which roughly corresponds to 20 minutes of flight time. We recall that the planning model we use encodes side effects of observing on the expected quality gains, namely the neighbours of an observed site as not worth visiting. We improved the greedy model to include a similar modelling of the sampling: when selecting the next-to-visit site, the close neighbours of sites already visited were ruled out, even if observing the weed density there would lower the remaining uncertainty in the estimated complete map. This "greedy-constrained" algorithm ends up having similar quality performance on the final map than the original "greedy" approach, with the flown distance smaller, as the sampling process stops when 40-45% of the plots have been observed.

The planning approach using the customized LBP algorithm (denoted "planning LBP" in the graphs) improving on the "planning" results from (Albore et al. 2015) both in cal-

culus time and in the final map quality. The better results of the "planning LBP" algorithm over the other planning version that uses the LibDAI implementation (Mooij 2010) of LBP algorithm (Murphy, Weiss, and Jordan 1999), depend on the generally larger sensitivity perimeter used for appreciating the expected quality, resulting in a better selection of sampling sites (we used LibDAI with a fixed size for this perimeter for all models).

Crucially, both the replanning approaches end up with a much better map quality at the distance limit. The different behaviours in terms of ratio quality/distance of the two approaches can be ascribed both to the difference in the visiting sequence, clearly less expensive when using a planner, and to the selection criterion. More details of empirical evaluation can be found in (Albore et al. 2015).

The AMPLE robotic execution framework

We turn toward an execution framework that parallelises the evaluation of the marginal values in the MRF, the generation of a plan, and the execution of the plan. The AMPLE meta-planner (Chanel, Lesire, and Teichteil-Königsbuch 2014) is a configurable anytime meta-planner that can drive our planner, dealing with all pending and time-bounded planning requests sent by the execution framework from many reachable possible future execution states, in anticipation of the probabilistic evolution of the system.

AMPLE is designed as a configurable bi-threaded program: an "execution" thread reactively interacts with the execution engine by managing multiple present and future planning requests, while an "optimisation" thread deliberatively optimises planning problems in background. The optimisation thread of the meta-planner, in our case, is delegated to update the MRF in order to offer the most updated expected gains to the planning requests pending in the execution thread, even if we limit the recalculations of new expected quality gains, i.e. computing the values of all conditional marginals $\mathbb{P}(X_i = k|x_A)$ for each possible sample in each site, only when the accumulated error ϵ_A exceeds a given threshold, with $\epsilon_A = \sum_{A_i \in A} |x_{A^i} - x^*_{A^i}|$, and x_{A^i} and $x^*_{A^i}$ are respectively the observed and the expected values in site A^i.

In an endless loop, AMPLE looks at, and accordingly reacts to, the current execution state (matching environmental conditions) by launching MRF updates, anticipating replanning episodes depending on the accumulated error on the reconstructed map, while executing the current best sampling action. Planning requests management in the execution thread conforms to a formal model defined as a configurable finite state machine, whose particular configuration in conjunction with the optimisation thread algorithm actually yield to a well-defined planning algorithm (Chanel, Lesire, and Teichteil-Königsbuch 2014). The execution thread adds and removes planning requests according to the current execution state, and reads the action to execute in the current policy backed from the optimization thread or in a default policy if no optimized action could be found in the requested time for the current execution state. On top of dissimilar expectations on quality gains, we can elaborate in parallel separate plans that optimize different optimization criteria, delegating the AMPLEto select the most advantageous one.

To adapt our algorithms to the above architecture, we rewrite our routines to conform to the general algorithmic schema of the meta-planner AMPLE, meaning that we ensure to return an applicable *relevant* action in any possible given state at a precise time point, when required by the execution engine. In other words, the algorithm should be proved to be strictly anytime in the sense of policy execution under time constraints. This step is easily done by using an anytime version of the best-first search algorithm for our classical planner (E.A. Hansen 2007).

Conclusions

We have described here a novel AI planning-based approach to deploy autonomous UAVs on demand without any heavy logistics, in order to sample pests in a crop field and map their spatial distribution. Currently, the most common sampling method relies on a fixed choice of sampled plots in the field, visited by humans that assess the abundance class. This solution is time consuming and in practise only a limited number of plots can be sampled. On the contrary, our approach *autonomously* produces an estimated map of pest abundance in the fields. The platform integrates Markov random fields for knowledge representation, updated at runtime by the observations of the UAV's embarked sensors. We have illustrated the planning approach to the problem of weeds mapping in crop field, on a realistic SAIL simulation platform. When compared to a greedy approach that selects the next sites to be visited by the UAV without accounting for the future flight duration, we observe that the planning approach leads to results of similar quality but at much less cost (measured as the distance covered during the flight). This means that if the same distance is allocated to the two approaches, the planner will enable to sample more plots, and therefore to provide better quality estimated maps.

To embed multi-criteria planning, and real-time decision making based on gathered knowledge, we have proposed an instantiation of AMPLE, a framework for anytime anticipated optimisation of probabilistic planning problems and anytime execution of the resulting policy, that embeds our replanning platform. The meta-planner efficiently parallelises the navigation and sampling tasks, while anticipating as much as possible the expected quality gains and permits us, for instance, to plan from the most informative probabilistic distribution of the spatial phenomena observed.

AI planning has been recently used with success for UAV mission planning, i.e. Search-And-Rescue problems with low-cost quadcopters (Bernardini, Fox, and Long 2014) or Multi-Target Detection and Recognition with middle-size UAVs (Chanel, Teichteil-Königsbuch, and Lesire 2013). While the former application is more oriented towards target tracking, the latter is focused on dynamic data acquisition and environmental knowledge optimisation like the application we present in this paper. But while they rely on rather complex POMDP techniques, our approach assumes the use of small UAVs with limited resources, which requires light planning capabilities like our determinisation-based replanning method.

To the best of our knowledge, this approach to the pest mapping task is among the first applications of planning onboard UAVs, whose problems cannot be known prior to the flight, and whose policies are optimised and successfully executed during the flight.

Acknowledgements We thank Sylvain Jasson and Damien Leroux for their precious help in efficiently handling marginal calculations in MRFs. This research has been funded by Région Midi-Pyrénées (grant 13/05/12.05).

References

Albore, A., and Bertoli, P. 2006. Safe LTL assumption-based planning. In *Proc. of Int. Conf. on Automated Planning and Scheduling (ICAPS-06)*.

Albore, A.; Peyrard, N.; Sabbadin, R.; and Teichteil-Königsbuch, F. 2015. A online replanning approach for crop fields mapping with autonomous uavs. In *Proc. of Int. Conf. on Automated Planning and Scheduling (ICAPS-15)*.

Barto, A.; Sutton, R.; and Watkins, C. 1989. Learning and sequential decision making. In Gabriel, M., and Moore, J., eds., *Learning and Computational Neuroscience*. MIT Press.

Bernardini, S.; Fox, M.; and Long, D. 2014. Planning the behaviour of low-cost quadcopters for surveillance missions. In *Proc. of Int. Conf. on Automated Planning and Scheduling (ICAPS-14)*.

Bertoli, P.; Cimatti, A.; Pistore, M.; Roveri, M.; and Traverso, P. 2001. MBP: a model based planner. In *Proc. of the IJCAI-01 Workshop on Planning under Uncertainty and Incomplete Information*.

Bonneau, M.; Gaba, S.; Peyrard, N.; and Sabbadin, R. 2014. Reinforcement learning-based design of sampling policies under cost constraints in markov random fields: Application to weed map reconstruction. *Computational Statistics and Data Analysis* 72:30–44.

Bonneau, M.; Peyrard, N.; and Sabbadin, R. 2012. A reinforcement-learning algorithm for sampling design in markov random fields. In *Proc. European Conference on Artificial Intelligence (ECAI-12)*, 181–186.

Chanel, C. P. C.; Lesire, C.; and Teichteil-Königsbuch, F. 2014. A robotic execution framework for online probabilistic (re)planning. In *Proc. of Int. Conf. on Automated Planning and Scheduling (ICAPS-14)*.

Chanel, C. P. C.; Teichteil-Königsbuch, F.; and Lesire, C. 2013. Multi-target detection and recognition by uavs using online pomdps. In *Proc. of AAAI-13*.

Cimatti, A.; Clarke, E.; Giunchiglia, F.; and Roveri, M. 2000. NuSMV: a new symbolic model checker. *International Journal on Software Tools for Technology Transfer* 2(4):410–425.

Domshlak, C. 2013. Fault tolerant planning: Complexity and compilation. In *Proc. of Int. Conf. on Automated Planning and Scheduling (ICAPS-13)*.

E.A. Hansen, R. Z. 2007. Anytime heuristic search. *J. Artif. Intell. Res.(JAIR)*.

Echeverria, G.; Lassabe, N.; Degroote, A.; and Lemaignan, S. 2011. Modular openrobots simulation engine: Morse. In *Proceedings of the IEEE ICRA*.

Geman, S., and Geman, D. 1984. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 6:721–741.

Ivankovic, F.; Haslum, P.; Thiébaux, S.; Shivashankar, V.; and Nau, D. S. 2014. Optimal planning with global numerical state constraints. In *Proc. of Int. Conf. on Automated Planning and Scheduling (ICAPS-14)*.

Johnson, G.; Mortensen, D.; and Gotway, C. 1996. Spatial and temporal analysis of weed seedling populations using geostatistics. *Weed Science* 44(3).

Koller, D., and Friedman, N. 2009. *Probabilistic Graphical Models : Principles and Techniques*. MIT Press.

Lipovetzky, N., and Geffner, H. 2012. Width and serialization of classical planning problems. In *Proc. European Conference on Artificial Intelligence (ECAI-12)*, 540–545.

Mooij, J. M. 2010. libDAI: A free and open source C++ library for discrete approximate inference in graphical models. *Journal of Machine Learning Research* 11:2169–2173.

Murphy, K. P.; Weiss, Y.; and Jordan, M. I. 1999. Loopy belief propagation for approximate inference: An empirical study. In *Proc. of the conference on Uncertainty in Artificial Intelligence (UAI)*, 467–475. Morgan Kaufmann Publishers Inc.

Peyrard, N.; Sabbadin, R.; Spring, D.; Brook, B.; and Mac Nally, R. 2013. Model-based adaptive spatial sampling for occurrence map construction. *Statistics and Computing* 23(1):29–42.

Quigley, M.; Conley, K.; Gerkey, B.; Faust, J.; Foote, T.; Leibs, J.; Wheeler, R.; and Ng, A. Y. 2009. ROS: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3.

Ramirez, M.; Lipovetzky, N.; and Muise, C. 2014. Lightweight automated planning toolkit. Technical report, http://lapkt.org/.

Roth, D. 1996. On the hardness of approximate reasoning. *Artificial Intelligence* 82:273–302.

Heuristic Scheduling of Space Mission Downlinks: A Case study from the Rosetta Mission

Gregg Rabideau[1], Federico Nespoli[2], Steve Chien[1]

[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

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 is limited, this can be a challenging task.

This paper describes a formulation of the data downlink scheduling problem used for the Rosetta orbiter, a European Space Agency cornerstone mission currently investigating the comet 67P/Churyumov-Gerasimenko. We first describe the abstract problem and the Rosetta mission specific problem, along with desirable features of downlink schedules. We outline several algorithms (including the Rosetta operational algorithm) and we compare their performance on both actual mission data.

Introduction

Spacecraft enable us to measure and explore a wide range of targets spanning Earth, to the planets and bodies of our solar system, to bodies beyond our galaxy to the furthest reaches of the universe.

Mission planning and scheduling is an extremely challenging part of operating 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 as to the volume of acquired data. There may be certain types of data that have deadlines for downlink. We describe a particularly challenging downlink problem in which data generation may occur over extremely long periods of time overlapping downlink periods.

Problem Definition

We formalize the data downlink problem as follows:

Given:
a set of buffers $B = \{b_1, b_2, \ldots b_n\}$
where each b_i has
an initial fill state: $init\_fill_i$
(fill state of buffer at start of planning interval)
end fill limit: end_fill_i

(hard limit on fill state of buffer
at end of planning interval)
hard capacity: $capacity_i$
desired margin: $margin_i$
(soft limit on buffer fill volume
at any point in the interval)
and the operations plan dictates for each b_i in B, there is a

fill_function fill(b_i,t)→rate where rate is bits/s

and there is a set of downlinks D={d_1,...d_m} where each

d_i=<start_d_i, end_d_i, rate_d_i>

(we assume that no downlinks are overlapping)

for each downlink specify a downlink rate from each buffer such that the sum of all buffers downlink rates is <= downlink data rate

∀b_i, a function empty(b_i,t)→rate (bits/s) such that

∀downlink d_i
start_d_i ≤ t ≤ end_d_i Σ empty(b_i,t) ≤ rate_d_i
(i.e. at any point in time we can only downlink up to our downlink capacity)

∀t ∈ current_schedule current_fill_state(b_i, t) ≤ $capacity_i$

also it is desired that peak fill state and end fill state meet their targets. Specifically:

no peak margins are violated
∀t in current_schedule, for all b_i
max(current_fill_state(b_i, t)) ≤ $margin_i$

no end margins are violated
for t = end of current_schedule,
∀b_i current_fill_state(b_i, t)) ≤ end_fill_i

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_i,t) function is not as flexible as desired.

The Rosetta Downlink Scheduling Problem

The Rosetta onboard data storage is partitioned into a set of buffers, called packet stores, for different types of science and engineering data. Each instrument is designated a packet store with a specified hard upper volume limit that cannot be changed during routine scheduling.

The behavior of each downlink can only be controlled by two commands: SET_SCI_DW_LEVEL and STOP_DUMP.

- The first, SET_SCI_DW_LEVEL, is issued at the start of the downlink and assigns a priority to each of the packet stores.
- The second, STOP_DUMP, can be issued any time during the downlink and stops the downlink of data from the specified packet store for the remainder of the current downlink. Note that once a packet store has been stopped, it cannot be restarted for that downlink.

These two commands, along with their timing and parameters, make up the decision variables available to the scheduler for controlling the "empty" functions described earlier. 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 downlink software in the following set of rules.

- Two of the packet stores (used for high-priority engineering data) have fixed priorities and cannot be stopped with STOP_DUMP commands.
- A packet stores remains "active" until the STOP_DUMP command is issued, after which no data will be downlinked regardless of priority.
- When more than one active packet store has data waiting to be downlinked, the one with higher priority will be dumped first.
- If more than one active packet store all have the same priority, data will be downlinked in a round-robin fashion.
- Each packet store has a predefined packet size which defines the minimum amount of data that will be downlinked on each round-robin cycle.
- When a packet store contains less than one packet, downlink for that packet store will stop, possibly allowing downlink to start on the next highest priority packet store.
- If, at any time during the downlink, data is added to an empty but active packet store, downlink for that packet store will restart, preempting any downlink from lower-priority packet stores.
- Both the "active" state and the priority are reset at the end of the downlink.

Given the two available commands, and the set of downlink rules, the primary job of the downlink scheduler is to assign priorities and decide when to stop dumps in order to prevent overflow on all packet stores. The secondary goal of the scheduler is to make selections that prevent margin violations. Last, for some of the packet stores, there is a desire for the scheduler to keep margins as large as possible.

To achieve these goals, the scheduler must first model the behavior of the packet stores so that volume and overflows can be accurately predicted. Fill rates from observations, and dump rates from downlinks, are all provided as inputs to the scheduler. When constructing the schedule for the first time, the scheduler must decide on which observations to include as well as which downlink commands to issue to best satisfy science requests. In this paper, we focus on the scheduling of downlink commands only, assuming an observation schedule is fixed. Note that this type of re-scheduling of the downlink commands is necessary during short-term planning when certain last-minute changes must be made (e.g. due to the loss of a downlink). However, in the larger mission planning/scheduling process, observation scheduling and downlink scheduling are performed simultaneously.

With a model of how data is collected and downlinked, the scheduler can generate a profile for each packet store 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 packet store that will overflow sooner, or exceed its limit by more than any other packet store, then that packet store should be given higher priority or more time to downlink.

In our original implementation, we used a fixed set of pre-assigned priorities that were mostly unique, and selected only the length of time for each dump. Due to the serial nature of the resulting dump schedule, this method proved brittle to communication loss (packet stores scheduled near the end of the downlink would be unfairly impacted). To address this problem, we implemented the priority-based method, which assigned different priorities for each downlink but did not issue STOP_DUMP commands. Ideally, both SET_SCI_DW_LEVEL and STOP_DUMP would be used to select the best possible dump schedule, measuring both quality and robustness of the schedule. This is left for future work.

In this paper, we focus on the priority-based method, since this is the default method used in current operations. Therefore, in this formulation, the control variables are:

for each downlink: $d_1,...d_i$
for each packet store: $s_1,...s_j$
assign a priority to $P_{a,b}$ for a =1...i, and b = 1...j

Note that these priorities, together with the packet store initial states and incoming data, effectively define an empty(b_i,t) function.

Figure 1 contains the pseudo-code for scheduling downlink priority commands for the Rosetta spacecraft. The initial schedule contains only fill activities that generate data into packet stores with continuously increasing volumes (beyond their limits). Lines 3-6 heuristically assign a priority to each packet store of each downlink, generating a list of overflows that result. If an overflow occurs before the end of a downlink, the schedule will perform limited backtracking to reschedule at most two of the previous downlinks (line 8).

The function priorityHeuristic (lines 11-17) implements the operational heuristic for making priority-based buffer allocations for a given downlink. Here, packet stores are given a priority that is inversely proportional to the number of the downlinks that occur before the first overflow (lines 15-17). This ensures that high priority is given to packet stores with more urgent need for downlink. A similar heuristic is used to choose STOP_DUMP times in the time-based method. As an example, if two or more packet stores have future overflows at around the same time, then

```
1.  scheduleDownlinks(downlinks)
2.    sortByStartTime(downlinks)
3.    for each d in downlinks
4.      for each ps in packet stores
5.        p = priorityHeuristic(d, ps)
6.        setDumpPriority(d, ps, p)
7.      if overflows exist
8.        backtrack
9.    return overflows
10.
11. priorityHeuristic(d, ps)
12.    if ps has an immediate overflow
13.      return MAX_PRIORITY
14.    else
15.      o = findFirstOverflow(ps)
16.      n = numberOfDownlinksBetween(d, o)
17.      return MAX_PRIORITY - n
```

Figure 1: Scheduling algorithm

they will likely be assigned the same priority (or same amount of time). If an *immediate* overflow has been identified for the given packet store, then it will be assigned the highest priority (line 12-13). An immediate overflow is defined as one that occurs before the next downlink. Note that downlink parameters are chosen independent of other downlinks and packet stores. Choices made for one downlink have only an indirect impact on the choices that will be made for future downlinks.

For evaluation purposes only, we consider three additional heuristics for selecting priorities. First, as a baseline, we randomly select priorities. Second, we assign the highest priority to the packet store with the largest

volume measured as percent of capacity. The remaining buffers are assigned the lowest priority. Last, we implement a heuristic that assigns priorities by normalizing the percent full values across the available priorities (e.g. with 10 priority values, a packet store with volume <10% is assigned the lowest priority). All four heuristics are compared in the empirical evaluation section of this paper.

Finally, certain packet stores may contain time-sensitive data (e.g. data which may impact future plans). For these "urgent" packet stores, the downlink latency (i.e. time between collection and downlink) can be reduced by increasing the required margin. To find the largest margin without creating overflows, we wrap the scheduleDownlinks function in a binary search loop. Each iteration of the loop either increases or decreases the margin, depending on the existence of overflows. The result is a schedule with large margins, keeping the data volume low, and reducing the time that data waits in the packet store. This technique is limited, however, to data collection schedules that have feasible downlink schedules (i.e. a solution must exist with no overflows).

Estimated Computational Complexity of the Scheduling Algorithm

Our analysis of the above scheduling algorithm indicates the following factors in its computational complexity

scheduleDownlinks = O(D * P * F)

where
D = # downlinks,
P = # packet stores,
F = # fill rate changes

Finding the best margin only adds a constant factor lg100 (binary search on a percentage between 1 and 100).

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 packet stores, and for one MTP, there are typically 30+ downlinks and hundreds of fill rate changes.

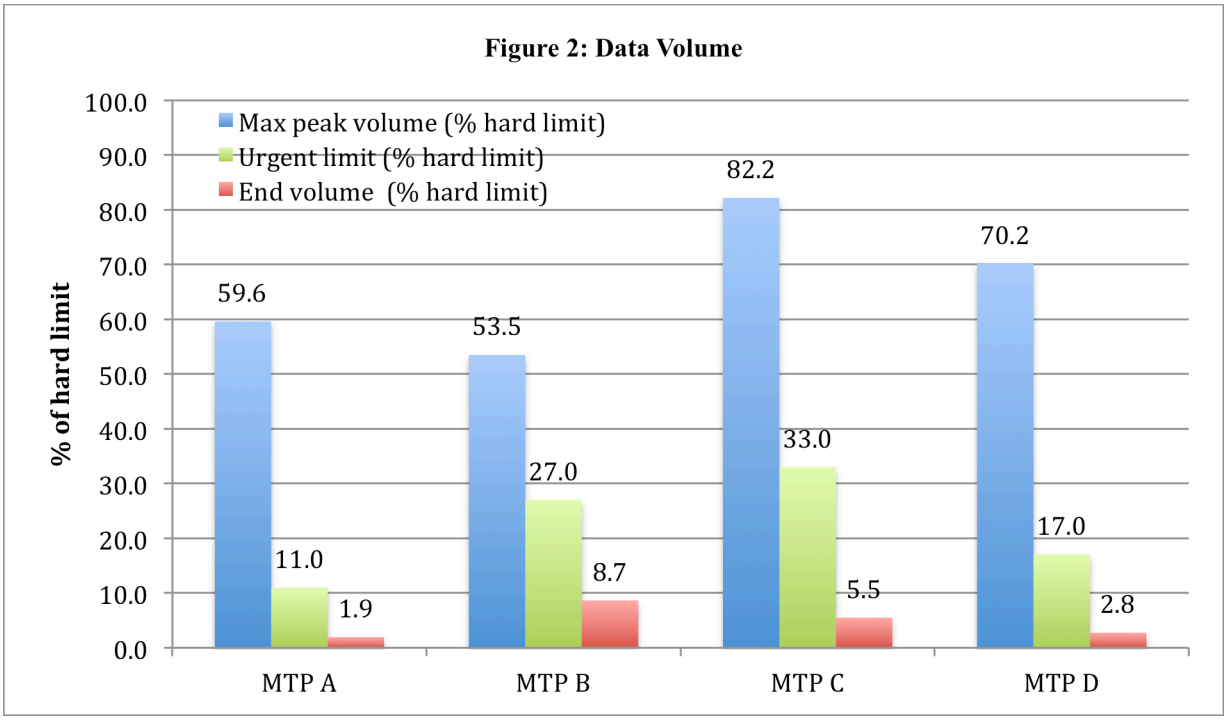
Empirical Evaluation of the Scheduling Algorithm

To date, we have conducted an empirical evaluation of the priority-based scheduling algorithm, the primary method used in operations. We use data from 4 medium-term planning (MTP) periods during the Rosetta mission, with each period spanning approximately 4 weeks. The CPU time required to generate each MTP downlink schedule was less than 1 minute running on a typical Windows laptop. The results are summarized in Figure 2 and Figure 3, with more details provided in Appendix A. The actual names of the packet stores and MTPs have been replaced for security reasons.

We evaluate its performance using the following metrics:

- Max peak volume percent: the maximum percent volume consumed for any packet store at any time during the MTP (no overflow if less than 100%)
- End volume percent: the percent volume consumed at the end of the MTP period
- Urgent limit: the smallest limit (i.e. largest margin) found by the schedule for the packet stores designated as containing urgent data
- Data collected: how much total data was collected from observations during the MTP

Figure 2: Data Volume



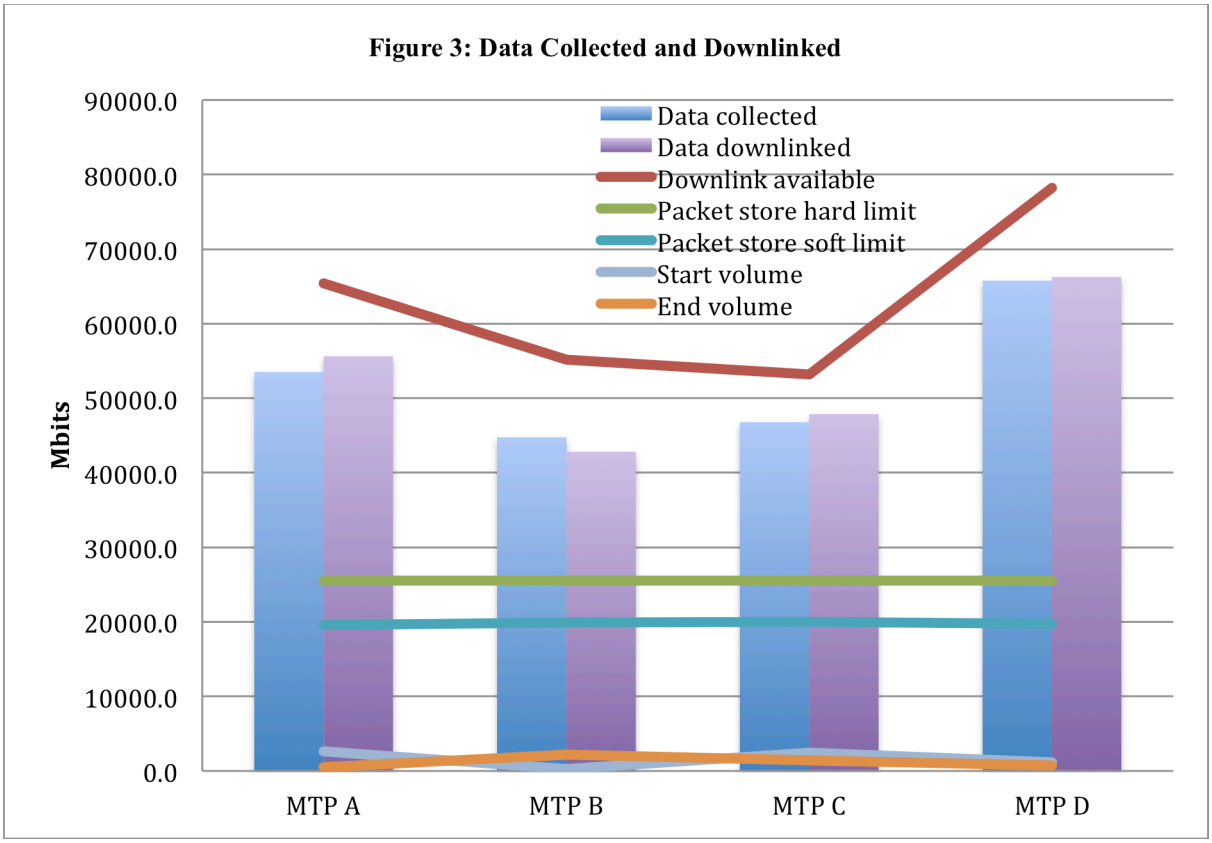
- Data downlinked: how much total data was downlinked out of the packet stores during the MTP
- Downlink available: how much downlink was theoretically available during the MTP (downlink duration multiplied by the bit rate).
- Packet store hard limit: what was the physical limit on the packet stores, and how do the data amounts compare
- Packet store soft limit: what was the operational limit imposed (including margin), and how do the data amounts compare
- Start and end volume: what was the volume of the packet store at the start and end of the MTP

The "Max peak volume" reported in Figure 2 give the maximum percent volume consumed for any packet store at any time during the MTP as a percent of the capacity for that packet store. This shows that, during the given 16-week period, at no point are any of the packet stores predicted to overflow. In addition, the data in Appendix A shows that at no point are any of the packet stores expected to exceed the desired "soft" limit. Some margin (typically 20%) on the packet store volume is maintained in order to account for uncertainties in the data collection and downlink model, which can occur from such things as variable data compression rates and communication outages. The "End volume" series in Figure 2 shows the percent volume consumed at the end of the MTP. Operationally, there is a preference to have minimal carry-over from one MTP to the next. For these four MTPs, the end volume stays below 10% of the capacity.

In each MTP, the data in two or three of the packet stores was considered urgent (designated with a '*' in the tables in Appendix A), and the downlink scheduler searched for the largest feasible margin on the packet stores. This meant keeping the volume low for the urgent packet stores without overflowing the other packet stores. In this way, the urgent data is not left to accumulate in the packet stores over long periods of time. The "Urgent limit" series in Figure 2 gives the smallest limit found by the scheduler for the packet stores containing urgent data.

In Figure 3, we see that the data collected amounts are very similar to the downlinked amounts for each MTP. It also shows that both values stay greater than 80% of the theoretical downlink available. The available downlink increases in the last MTP due to an increase in downlink rate, which occurs as the spacecraft exits solar conjunction. Finally, we can see that the data collected in each MTP is roughly between 2x and 3x the total packet store limit.

Figure 3: Data Collected and Downlinked



	Peak Volume			
	random	full	even	ops
MTPA	68.4	69.2	60.9	59.6
MTPB	64.7	52.9	54.4	55.3
MTPC	204.6	231.3	119.4	82.2
MTPD	77	69.5	66.1	70.2
Avg	103.675	105.725	75.2	66.825

Table 1: Peak Volume

	Urgent Limit			
	random	full	even	ops
MTPA	17.2	14.1	14.1	10.9
MTPB	31.2	32.8	29.7	28.1
MTPC	100	100	100	32.8
MTPD	32.8	18.7	20.3	17.2
Avg	45.3	41.4	41.025	22.25

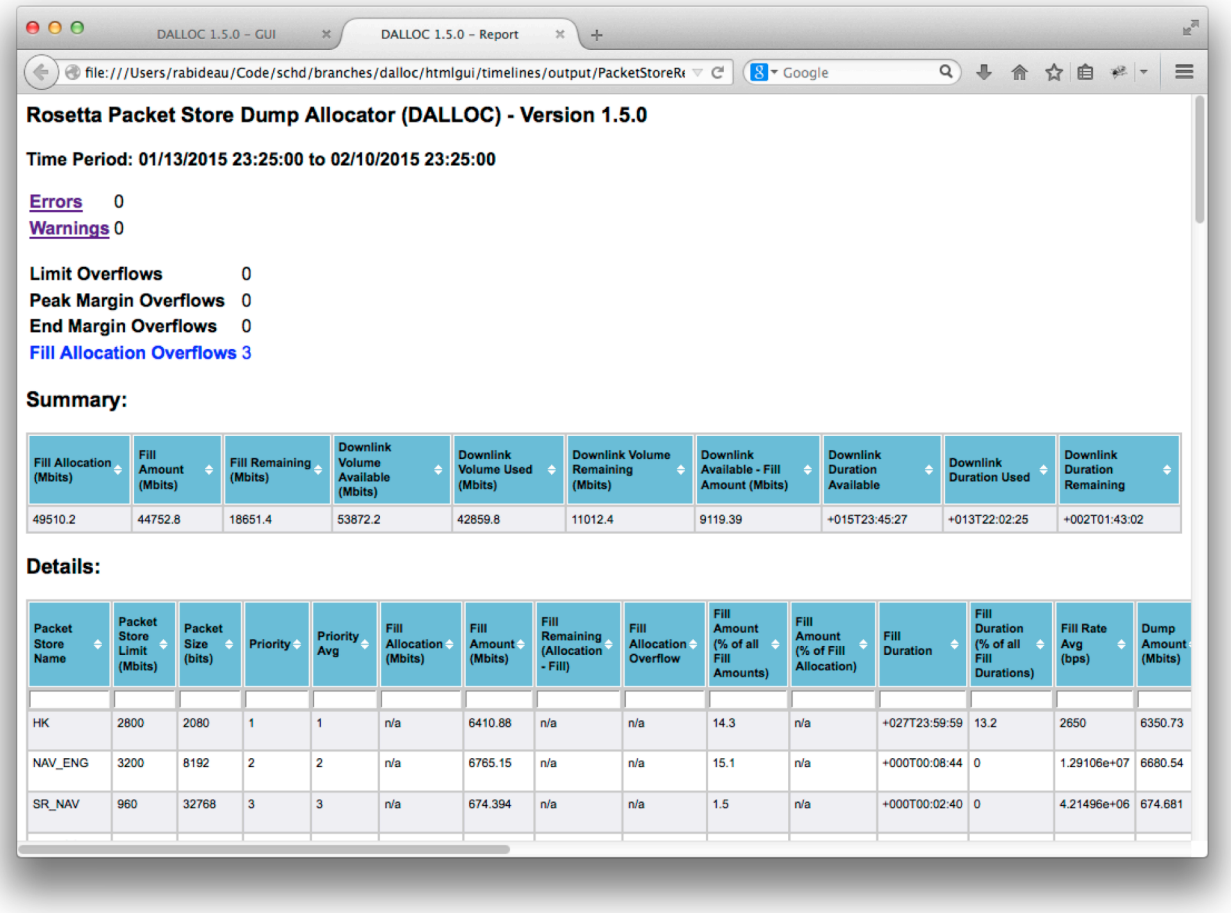
Table 2: Urgent Limit

We should mention that there are certain packet stores that are designated as containing high-priority, engineering data (marked with '**' in the tables in Appendix A). The data in these high-priority packet stores is downlinked first, and all of the data is downlinked before time is given to the other packet stores. Note that peak volume (column 3) stays very low for these packet stores.

Additional runs were performed to compare different heuristics for selecting priorities. Tables 1 and 2 show the results. In the "random" heuristic, priorities were selected at random to create a baseline for comparison. The "full" heuristic assigns the highest priority to the packet store with the highest volume measured as a percent of buffer capacity. The "even" heuristic assigns priorities by normalizing the fill percentages across the available priority values. The "ops" heuristic is the operational heuristic described in this paper. Table 1 records the maximum peak volume that resulted from applying each of the heuristics to each MTP. Table 2 records the smallest limit achieved for the "urgent" packet stores. The most

Figure 4: GUI

Figure 5: Report



significant difference can be seen in MTPC, which has the least amount of downlink available. In this case, only the operational heuristic generates a downlink schedule that does not result in packet store overflow. Note that the operational heuristic finds the smallest "urgent" limit in all four cases, but to achieve this, will sometimes create a higher maximum peak volume.

The GUI used in operations to evaluate the downlink schedule can be seen in Figures 4 and 5. The first is a zoom-able, scrollable, interactive graph that shows how the packet store volumes change over time within the given MTP. The bottom of the graph contains data on the various parameters that affect the volume, some of which the scheduler can control (e.g. priorities) and others that it cannot (e.g. downlink volume). The second page of the GUI provides a report on the resulting downlink schedule. The values in Appendix A were taken from this report.

In the future, we plan to evaluate the time-based scheduling method, as well as a combined method that selects both priority and stop time for each packet store, which in theory should produce the best results. Also, we plan to conduct scaling tests to validate our analysis of the computational complexity of the algorithm. We also plan on developing synthetic problem generators to further explore the performance of the downlink scheduling algorithms.

Discussion

Automated downlink planning is in operational use for the Mars Express [Cesta et al. 2007] mission. However in their data model observations produce data instantaneously, whereas in the Rosetta model data producing activities have rates that have significant temporal extent (such as engineering production continuously over the entire mission, and science observations doing the same). Interestingly, Cesta et al. characterize the problem as a planning problem (that of producing the sequence of downlink controlling commands). We take an opposing view that the end product is a scheduling/resource allocation problem, that of providing a downlink profile. MEX-MDP (Mars Express

Spacecraft Memory Dumping Problem) also takes into account the size of the plan (this is not an issue for Rosetta). 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. 2014]. 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 DALLOC).

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 additional constraints. We describe two implemented heuristic solutions to this problem and we present complexity analysis and empirical evaluation on actual Rosetta mission data.

Acknowledgements

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

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.

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.

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.

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.

S. Chien, G. Rabideau, D. Tran, J. Doubleday, D. Chao, F. Nespoli, M. P. Ayucar, M. C. Sitja, C. Vallat, B. Geiger, N. Altobelli, M. Fernandez, F. Vallejo, R. Andres, M. Kueppers, Activity-based Scheduling of Science Campaigns for the Rosetta Orbiter: An Early Report on Operations International Symposium on Artificial Intelligence, Robotics, and Automation for Space (i-SAIRAS 2014). Montreal, Canada. June 2014.

M. D. Johnston and G. Miller. "Spike: Intelligent scheduling of hubble space telescope observations." Intelligent Scheduling (1994): 391-422.

Russell Knight, Caroline Chouinard, Grailing Jones, Daniel Tran, Leveraging Multiple Artificial Intelligence Techniques to Improve the Responsiveness in Operations Planning: ASPEN for Orbital Express, AI Magazine, Vol 35, No 4, 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.

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.

Appendix A

	MTP A						
	#1	#2	#3	#4	#5	#6	#7
PS1**	2800.0	127.8	4.6	80.0	101.2	3.6	30.0
PS2**	3200.0	154.9	4.8	80.0	103.3	3.2	30.0
PS3**	960.0	67.5	7.0	80.0	0.0	0.0	30.0
PS4	800.0	273.3	34.2	80.0	10.8	1.4	30.0
PS5	50.3	21.0	41.7	80.0	12.3	24.4	30.0
PS6	160.0	91.8	57.4	80.0	17.5	10.9	30.0
PS7	320.0	86.8	27.1	80.0	4.8	1.5	30.0
PS8*	640.0	67.5	10.5	11.0	18.7	2.9	30.0
PS9*	320.0	33.8	10.6	11.0	18.8	5.9	30.0
PS10	1600.0	494.6	30.9	80.0	41.9	2.6	30.0
PS11	7516.2	3645.7	48.5	80.0	0.0	0.0	30.0
PS12	0.8	0.0	0.0	80.0	0.0	0.0	30.0
PS13	800.0	288.1	36.0	80.0	81.2	10.2	30.0
PS14	4000.0	2385.0	59.6	80.0	0.0	0.0	30.0
PS15	0.8	0.0	0.0	80.0	0.0	0.0	30.0
PS16	880.0	325.4	37.0	80.0	62.2	7.1	30.0
PS17	595.6	25.4	4.3	80.0	14.9	2.5	30.0
PS18	888.0	57.4	6.5	80.0	10.0	1.1	30.0
TOTAL/AVG	25531.7	8146.1	31.9	72.3	497.6	1.9	30.0

	MTP B						
	#1	#2	#3	#4	#5	#6	#7
PS1**	2800.0	173.2	6.2	80.0	131.2	4.7	30.0
PS2**	3200.0	206.6	6.5	80.0	193.2	6.0	30.0
PS3**	960.0	67.4	7.0	80.0	0.0	0.0	30.0
PS4	800.0	353.8	44.2	80.0	155.6	19.5	30.0
PS5	50.3	18.8	37.3	80.0	14.0	27.8	30.0
PS6	160.0	85.6	53.5	80.0	33.0	20.7	30.0
PS7	320.0	0.0	0.0	80.0	0.0	0.0	30.0
PS8*	640.0	168.5	26.3	27.0	6.7	1.0	30.0
PS9*	320.0	28.4	8.9	27.0	24.6	7.7	30.0
PS10	1600.0	193.8	12.1	80.0	57.2	3.6	30.0
PS11	7516.2	1839.7	24.5	87.0	729.1	9.7	30.0
PS12	0.8	0.0	0.0	87.0	0.0	0.0	30.0
PS13	800.0	174.4	21.8	80.0	75.1	9.4	30.0
PS14	4000.0	1799.1	45.0	75.0	753.2	18.8	30.0
PS15	0.8	0.0	0.0	75.0	0.0	0.0	30.0
PS16	880.0	304.2	34.6	80.0	28.0	3.2	30.0
PS17*	595.6	29.3	4.9	27.0	8.6	1.4	30.0
PS18	888.0	0.0	0.0	80.0	0.0	0.0	30.0
TOTAL/AVG	25531.7	5442.8	21.3	71.4	2209.5	8.7	30.0

#1 Packet store hard limit
#2 Peak volume
#3 Peak volume (% hard limit)
#4 Peak volume soft limit (% hard limit)
#5 End volume
#6 End volume (% hard limit)
#7 End volume soft limit (% hard limit)

* Contains urgent data
** Contains high-priority engineering data

	MTP C						
	#1	#2	#3	#4	#5	#6	#7
PS1**	2800.0	164.3	5.9	80.0	2.5	0.1	30.0
PS2**	3200.0	275.4	8.6	80.0	68.7	2.1	30.0
PS3**	960.0	337.3	35.1	80.0	0.0	0.0	30.0
PS4	800.0	516.4	64.6	80.0	89.3	11.2	30.0
PS5	50.3	27.2	54.0	80.0	18.7	37.2	30.0
PS6	160.0	87.3	54.6	80.0	58.3	36.4	30.0
PS7	320.0	0.0	0.0	80.0	0.0	0.0	30.0
PS8*	640.0	209.5	32.7	33.0	21.8	3.4	30.0
PS9*	320.0	80.7	25.2	33.0	28.2	8.8	30.0
PS10	1600.0	797.1	49.8	80.0	146.0	9.1	30.0
PS11	7516.2	6177.7	82.2	87.0	804.1	10.7	30.0
PS12	0.8	0.0	0.0	87.0	0.0	0.0	30.0
PS13	800.0	410.2	51.3	80.0	102.8	12.8	30.0
PS14	4000.0	2732.3	68.3	75.0	0.0	0.0	30.0
PS15	0.8	0.0	0.0	75.0	0.0	0.0	30.0
PS16	880.0	447.1	50.8	80.0	28.8	3.3	30.0
PS17*	595.6	43.4	7.3	33.0	16.4	2.8	30.0
PS18	888.0	25.0	2.8	80.0	17.6	2.0	30.0
TOTAL/AVG	25531.7	12330.8	48.3	72.4	1403.3	5.5	30.0

	MTP D						
	#1	#2	#3	#4	#5	#6	#7
PS1**	2800.0	172.0	6.1	80.0	46.7	1.7	30.0
PS2**	3200.0	232.9	7.3	80.0	77.5	2.4	30.0
PS3**	960.0	67.5	7.0	80.0	0.0	0.0	30.0
PS4	800.0	265.1	33.1	80.0	50.7	6.3	30.0
PS5	50.3	29.7	59.0	80.0	13.8	27.3	30.0
PS6	160.0	67.0	41.9	80.0	42.1	26.3	30.0
PS7	320.0	0.0	0.0	80.0	0.0	0.0	30.0
PS8*	640.0	83.4	13.0	17.0	20.2	3.2	30.0
PS9*	320.0	50.5	15.8	17.0	6.4	2.0	30.0
PS10	1600.0	424.1	26.5	80.0	32.1	2.0	30.0
PS11	7516.2	4465.6	59.4	87.0	0.0	0.0	30.0
PS12	0.8	0.0	0.0	87.0	0.0	0.0	30.0
PS13	800.0	217.5	27.2	80.0	29.1	3.6	30.0
PS14	4000.0	2806.2	70.2	75.0	330.0	8.2	30.0
PS15	0.8	0.0	0.0	75.0	0.0	0.0	30.0
PS16	880.0	337.0	38.3	80.0	27.4	3.1	30.0
PS17*	595.6	42.7	7.2	17.0	26.7	4.5	30.0
PS18	888.0	0.1	0.0	80.0	0.1	0.0	30.0
TOTAL/AVG	25531.7	9261.3	36.3	69.7	702.9	2.8	30.0

Heuristic Onboard Pointing Re-scheduling for an Earth Observing Spacecraft

Steve Chien, Martina Troesch
Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, USA
steve.chien@jpl.nasa.gov

Abstract

Prior space missions have not routinely used onboard decision-making. The Autonomous Sciencecraft (ASE), flying onboard the Earth Observing One spacecraft, has been flying autonomous agent software for the last decade that enables it to analyze acquired imagery on board and use that analysis to determine future imaging. However ASE takes approximately one hour to analyze and respond.

This paper describes a scheduling prototype for the Earth Observing Autonomy (EOA) project to increase the responsiveness of spacecraft flight software for onboard decision-making as well as to increase the capabilities of flight software. Specifically, we target onboard image analysis and response within a single orbital overflight at low Earth orbit (about eight minutes). We focus on the re-scheduling of the future image acquisitions in the context of an existing set of requests along with new requests based on onboard analysis of just acquired imagery. We describe a greedy, constructive, scheduler with $O(n^2)$ performance and present preliminary results on its performance.

Introduction

The Earth Observing Autonomy (EOA) project targets the development of a spacecraft autonomy capability to enable a wide range of Earth Observing, pointing spacecraft (e.g., Earth Observing One [Ungar et al. 2003], The Spot constellation [Wikipedia Spot 2015], Orbview Class spacecraft (such as Worldview-3) [Ball, 2015, Wikipedia Worldview-3 2015] to image, analyze the image, and re-image based on that analysis within a single overflight, imposing a responsiveness constraint of 5-8 minutes. This would represent a dramatic improvement over the current state of the art, ASE [Chien et al. 2005], which responds within roughly 1 hour.

We describe a software prototype of the EOA capability that includes several autonomy components:

1. *Onboard science processing algorithms.* Science analysis algorithms process onboard image data to detect science events and suggest reactions to maximize science return. Specifically we investigate the use of the Mixture –tuned Match Filter (MTMF) [Boardman and Kruse 2011] for onboard spectral analysis of acquired imagery. However ASE has already demonstrated the utility of thermal analysis for volcanoes and wildfires [Davies et al. 2006], spectral analysis for flooding [Ip et al. 2006], spectral analysis for cryosphere study [Doggett et al. 2006], as well as spectral unmixing for mineralogical analysis [Thompson et al. 2012].
2. *Onboard planning and scheduling software.* The Continuous Activity Scheduling Planning Execution and Replanning (CASPER) [Chien et al. 2000] combined with the Eagle Eye Mission Planning Software [Knight et al. 2013] system generates a baseline mission operations plans from observation requests. This baseline plan is subject to considerable modification onboard in response to data analysis from step 1. The model-based planning algorithms enable rapid response to a wide range of operations scenarios based on models of spacecraft constraints. However, in this paper we focus on a greedy, constructive, non-backtracking scheduler designed specifically for this application.
3. *Robust execution software.* The JPL core flight software [Weiss 2013] (CFS) expands the CASPER mission plans to low-level spacecraft commands and includes a powerful and expressive sequencing engine. The CFS sequencing engine monitors the execution of the plan and has the flexibility and knowledge to perform improvements in execution as well as procedural responses to execution anomalies.

One challenge to spacecraft autonomy is *limited computing resources*. An average spacecraft CPU offers 200 MIPS and 128 MB RAM – far less than a typical laptop computer. For the EOA prototype, we baseline a Rad 750 or Leon processor for all of the autonomy capability.

EOA demonstrates an integrated autonomous mission response capability using onboard science analysis, replanning, and robust execution. EOA performs intelligent science data analysis, and spacecraft retargeting. This capability can reduce data downlinked in cases where onboard analysis determines the data not of interest (e.g. search for active volcanos and return only images that contain active volcanos). This capability can also enable an increase in science return. In many cases, a mission is not limited by observation time, but rather by downlink volume. In these cases, if the spacecraft can acquire imagery searching for a specific signature and not return the data if the signatures not found, then search can be made much more efficient. Specifically, the spacecraft can search for active volcanoes a large amount of the time, and only pay the downlink cost proportional to the number of images with active volcanoes rather than the total number of images acquired searching for active volcanoes. In cases where phenomena may be short-lived, onboard detection may enable additional data to be acquired, gathering more science data on the scarce phenomena (e.g. when detecting an active volcano, add requests to image it more frequently and in greater detail).

The execution flow of the EOA software is shown in Figure 1. As the spacecraft overflys targets, it images them. As the imagery is acquired, it is processed onboard the spacecraft. Based on the operations policies of the missions, this analysis may result in new image requests. These image requests are folded into the prior image requests and a new schedule is constructed that may acquire the new image and may change other images acquired (such as pre-empting a less valuable target). Spacecraft execution then continues.

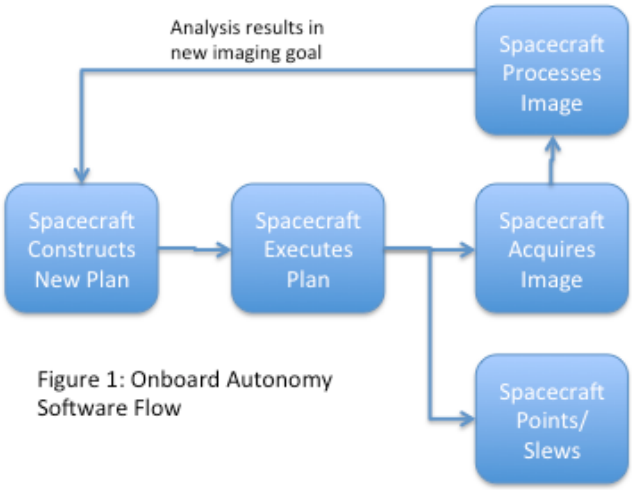


Figure 1: Onboard Autonomy Software Flow

These capabilities enable radically different missions with significant onboard decision-making allowing new ways to conduct science from space. The paradigm shift toward highly autonomous spacecraft will enable future space missions to achieve significantly greater science returns with reduced risk and reduced operations cost.

In this paper, as the meeting topic is planning and scheduling, we focus on the rescheduling portion of the overall responsiveness of the mission. We begin by describing the overall on board response scenario to show the overall mission timeline and the context of rescheduling.

Autonomous Science Scenario

Our onboard planning capability is designed to support an EOA mission scenario. While the EOA software is designed to support a wide range of spacecraft without any modification, in this section, we describe a scenario with a Worldview-3 like spacecraft [Ball 2015, Wikipedia 2015] to image science targets, process and analyze onboard image data, and re-plan operations based on science results.

For this demonstration we assume several baseline mission parameters.

Parameter	Value
Orbit	950 km Sun synchonous
Initial Science Images	30-40° lookahead from nadir
Response image	Nadir to 20° lookahead
Spacecraft slew rate	4.5° per second, instantaneous start and stop, no settle time
Imaging time	Dwell of 1s per image
Image request granule "footprint"	0.5 km along track x 4 km across track

In Figure 1 we highlight some of the geometry characteristics of the EOA scenario. As the spacecraft orbits the earth, it has several viewing windows. The first viewing window is the initial science image window which covers from 31 to 38° in front of the spacecraft. The second viewing window is the response image which covers from 0° lookahead (nadir) to 28° lookahead. As the spacecraft flies over the earth it is imaging in large number of locations in the initial science window. As it acquires this imagery, software analyzes the imagery onboard the spacecraft. This analysis indicates the possible need to take follow-up imagery (in the response imaging window). For example, in the initial science window we might search for the thermal signature of a volcanic eruption or wildfire. In the response window we might further image to precisely determine the extent of the lava flow and the exact temperature map of the flow.

The goal of the scheduler is to accommodate as many of the initial science and response imaging requests but is guided by the priority of the requests and restricted by the pointing and slewing capabilities of the spacecraft (as well

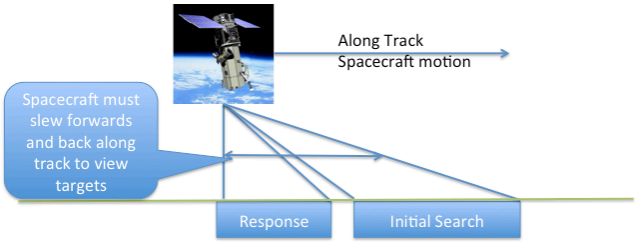


Figure 2: spacecraft must slew along track.
Distance between initial search and response windows large compared to distance between two requests in same window.

as any other operations constraints). As shown in Figure 2, from a side view, the speacecraft must slew forwards and backwards looking a variable amount ahead to view the image targets. At the same time, the spacecraft is moving forwards due to orbital motion (at approximately 7.5km per second). Because processing the images requires some time, the initial search window is significantly ahead of the response window. This enables initial search-ed images to be processed/analyzed in time to allow for scheduling of followup imagery in the response window. The response window does not extend behind the spacecraft in order to maintain consistent lighting conditions.

This slewing forwards and backwards along the spacecraft motion track is complicated by two things. First, the angle at which the spacecraft must look forward to view the target is a non linear function of when the spacecraft wishes to view the target. Specifically, at nadir, for the Earth, in a 950km orbit, 1 degree of lookahead corresponds to 16.6 km ahead of nadir in the ground track. However, at 37° of lookahead, 1° of further lookahead (e.g. to 38° lookahead) corresponds to 30.7 km ahead in the ground track. The second issue is that typically the slew rate of the spacecraft is not linear, there is a ramp up acceleration of the spacecraft to some maximum slew rate, a portion of the slew at the maximum rate, then a ramp down as the spacecraft arrives at the desired position.

Figure 2, Case 1 shows these two factors from the spacecraft pointing perspective. In this example the spacecraft is looking ahead and wants to view a target further ahead beyond the current look angle. The spaceraft could simply wait until the target comes into view, or it can slew ahead to meet the target. The blue line shows the track of a fixed point on the ground in terms of the look angle from the spacecraft as the spacecraft approaches the point. This line indicates that at time 0 the target is at 42° lookahead. The red line shows the angular position of the spacecraft reachable from the starting point of nadir as a function of time. The intersection of these two lines shows the earliest possible time that the spacecraft can view the target. The graph indicates that if the spacecraft begins slewing it will be able to reach the target but that the target will be at 38° lookahead when it is reached. In this case the motion of the spacecraft is helping us to meet the target earlier.

The right side of Figure 2 shows a different case, Case 2. In Case 2, the spacecraft is pointing at 38° lookahead, and

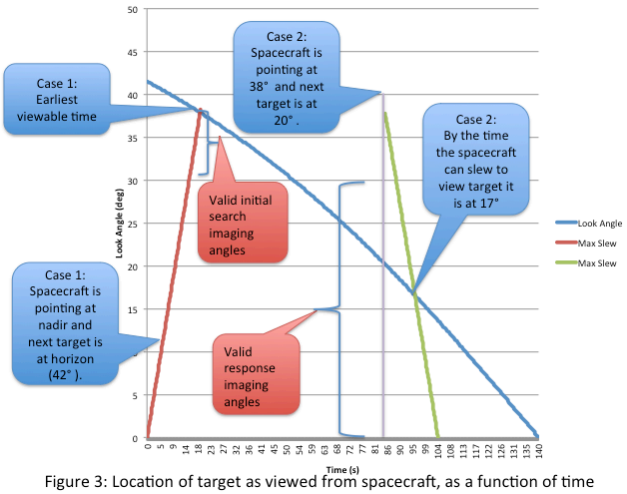


Figure 3: Location of target as viewed from spacecraft, as a function of time

wants to next view a target currently at 20° lookahead. In this case the spacecraft motion is carrying the target (relative to the spacecraft) away from the current spacecraft pointing and the slew must catch up. The graph shows that the by the time that the spacecraft can view the target it will be at 17° lookahead.

Figure 4 is a view from above the spacecraft looking down on the Earth. As the spacecraft moves along track (from left to right in Figure 4), the spacecraft must also slew across track (up and down in the Figure) as well as forwards and backwards along the ground track (left and right in the Figure) to image targets.

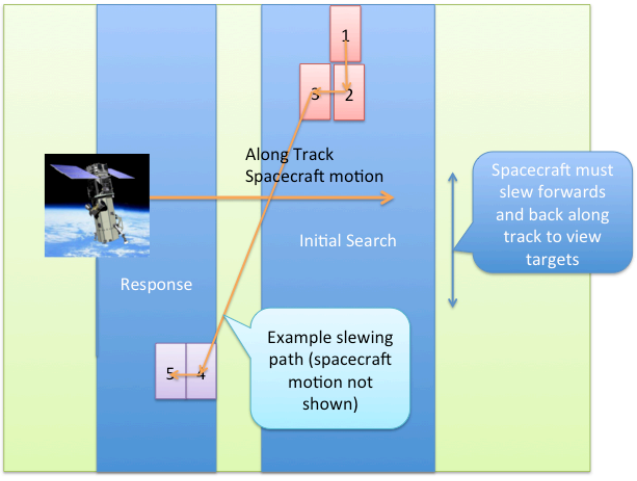


Figure 4: spacecraft must also slew across track.
Spacecraft moves along track as while slewing and imaging.

This scheduling problem is a challenging one for several reasons.

1. The spacecraft has limited ability to slew from one target to the next (e.g. each slew takes up valuable time).
2. Targets are distributed across the ground track of the spacecraft so that the amount of time required to image a target depends on the preceding and following (temporally) targets in the schedule.
3. Because the initial viewing and response doing windows are separated angularly, slewing back and forth between these windows can be wasteful of time.
4. Image analysis takes time. During this time spacecraft is moving towards the target(s). This is the reason why the initial image analysis and response image analysis windows are not overlapping, to allow the onboard software time to analyze the images.
5. Generating the schedule also takes time (the focus of this paper).
6. When calculating a start time to schedule an observation of a target, the spacecraft intercepts the target. The spacecraft must slew to a given position (of the target), reaching that position at the exact time that the target is in that position relative to the spacecraft. This requires an accurate model of the spacecraft slew time as well as the ability to project where relative to the spacecraft any target will be at any point in time.
7. In addition to pointing, the scheduler must consider other resources such as power, thermal, data volume (e.g. [Chien et al. 2010, Chien et al. 2012]). However in this paper we focus on the pointing and slewing aspect of the problem as the state and resource management aspect of the problem has been considered elsewhere.

In order to simplify the scheduling problem we first transform the image request locations from a <latitude, longitude, altitude> coordinate frame of reference to an <along track, across track> frame of reference (in this process using a model of the spacecraft orbit). From this <along the track, across track frame of reference>, combined with the spacecraft orbit, the set of valid times to view any target in the initial viewing window or response window is easily computed.

R = $\{r_1,\ldots r_n\}$ sorted from highest priority to lowest priority

```
achieved_requests = {}
best_solution = nil
for adding_request ∈{r1…ra}
 call schedule( achieved_requests ∪ {adding_request});
 if success then
   achieved_requests ← achieved_requests ∪ {adding_request}
   best_solution ← solution returned by schedule
```

```
Schedule(request_set = {r1…ra})
Sort request set by earliest start time
  (e.g. request with earliest start time is first in set)
current_solution = {}
for current_request ∈{r1…ra}
   attempt to add current_request to current_solution
   by scheduling it at the earliest possible time that
   it will fit into the schedule
   if cannot add return FAIL
   else (success} continue
return current_solution
```

This scheduling algorithm represents a greedy outer loop where we try to add requests in priority order. The inner loop is given a set of requests, and attempts to schedule them sweeping forward in time considering earliest possible start time requests first.

Figure 5 shows the inner loop of the scheduler. In figure 5a the two headed arrows indicate the earliest and latest possible times each image can be acquired. The longer intervals are response images and the shorter intervals are initial search requests. In Figure 5a the requests are sorted by earliest possible start time. Figure 5b shows the requests being scheduled. The software tries to add each request in the earliest start time sorted order, adding the request to the schedule as early as possible. The orange blocks indicate the slew time and the blue blocks indicate the imaging time. The imaging time is roughly constant but the slewing time is higher if the preceding image was of a different type (initial, response), this is because the spacecraft is generally slewing a greater distance (up to 0° → 38° lookahead) as opposed to from one initial search to another (maximum slew from 31° → 38°) or from one response to another (from 0° → 28°).

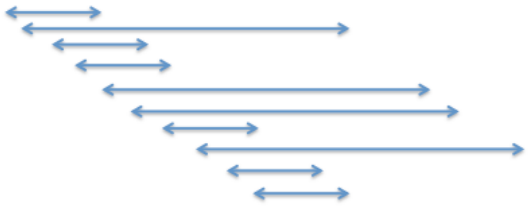


Figure 5a: Requests with possible scheduling times sorted by earliest possible start time

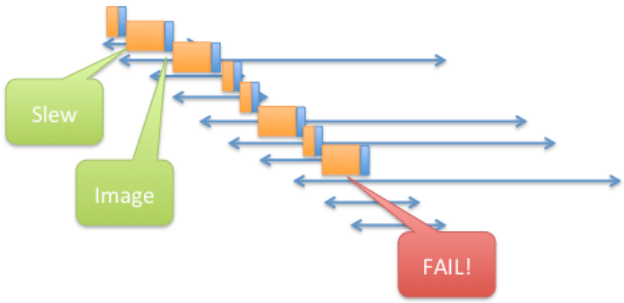


Figure 5b: Scheduling at earliest possible time until failing.

As the inner loop of the scheduler is attempting to insert the next request, it must repeatedly solve the problem of slewing forwards and backwards along track to view targets either ahead of or behind the current position. The scheduler solves this problem of the intersection of the earliest possible slew position curve intersecting the target position angle relative to the spacecraft using Newtons Method [Wikipedia Newtons 2015]. In computing this observation time the software must take the latest of: the spacecraft along track slew intersecting the target, the mission mode constraints (e.g. initial search allowed look angles, response image allowed look angles), as well as the across track slew time to view the target. Solving for the across track slew time is simpler than the along track problem – while the earths curvature does make the angular position a nonlinear function of the ground distance, there is no across track motion to compensate for and indeed this conversion from ground distance to angular distance can be pre-processed. In practice for our scenario slew times can range from a fraction of a second (for adjacent tiles) to 5-10 seconds.

While we currently use a simple constant slew rate model for our current prototype, a more realistic model would have:

1. an acceleration/decceleration profile,
2. a maximum rate,
3. a different model for different axes of the spacecraft,
4. a settling time for the spacecraft to stabilize after a slew in which the settling time depends on the parameters of the image being acquired as well as the velocity and acceleration profile of the slew

In our software architecture we treat the slew computation as a black box so that we can easily insert a high(er) fidelity model.

Estimated computational complexity of the scheduling algorithm

Our analysis of the above observation scheduling algorithm indicates several factors in its computational complexity.

Since we schedule from scratch each iteration we will always make R passes through the outer loop where R is the number of scheduling requests.

Each of the R passes through the outer loop makes a single call to the "schedule" function. The schedule function performs a computation to attempt to add a slew and image to the schedule each iteration. This effort to add a slew and image requires computation in worst case on the order of the number of items currently in the schedule. While the number of items currently in the schedule is certainly no worse than R above (total number of requests) if the number of requests that can actually fit into the schedule S_{max} is much smaller than R this will be a much lower bound.

For example, if the entire search window corresponds to 200s of observation time, and the minimum observation length is 2s $S_{max} < 100$ so if R is >> 100 it does not matter, the complexity is of order S_{max}. So overall the computational complexity of the scheduler is $RS_{max}C$ where R is the total number of requests and S_{max} is the number of requests that will actually fit in the schedule and C is the computation required to evaluate the feasibility of inserting a single request into the schedule.

Note that this algorithm does not take advantage of the fact that the set of changes to the request set is small compared to the size of the request set. An obvious optimization would be to only reschedule the portion of the schedule of lesser priority than the highest priority new request.

Empirical evaluation of the scheduling algorithm

In order to verify our analysis of the computational complexity of the algorithm, we also performed a limited empirical analysis of the algorithm. For this analysis we generated a synthetic data set using the following parameters.

Parameter	Value
Initial request probability	1-5%
Probability of a response image given a search request performed	25, 50, 75%
Scheduling horizon	45° lookahead

Figure 6: Preliminary run information

The empirical data is shown at the end of the paper. Graphs 1 and 2 show that the scheduler can only completely achieve a relatively small percentage of initial search requests (a few percent). Already if 2% of all possible tiles are requested for search with no responses many of the search requests are not being satisfied.

When response requests are included this further drives down the percentage of search requests that can be scheduled because response requests are higher priority and they preclude search requests. Graphs 3 and 4 show that as response images are added to the scheduling problem (at higher priority than search requests) the scheduler is able to accommodate fewer search requests. Graph 3 shows where 25% of searches yield a response

and Graph 4 shows where 75% of searches yield a response.

Graphs 5 and 6 show the CPU time required for the scheduler in VxSIM. The run-time data indicates that the scheduler is extremely fast, taking only a fraction of a second in the software simulation. While the flight processor is expected to be significantly slower, the scheduling algorithm is not optimized in any way. One obvious optimization is that the scheduler is solving the problem from scratch each invocation when the majority of the inputs have not changed. Clearly an incremental rescheduler offers great potential for efficiency gains.

Discussion

The Autonomous Sciencecraft (ASE) has been flying the CASPER continuous planner on board the Earth Observing One (EO-1)spacecraft for the past decade. However, the response time for CASPER on EO-1 is tens of minutes-in part due to the meager computation on board the spacecraft: 3 MIPS and no floating point computation in hardware for the RAD 3000 CPU on board. Additionally, the planning problem for EO-1 does not involve significant geometric issues. The spacecraft generally only images using its push broom imager and a fixed angle relative to nadir, therefore there is no flexibility in the imaging time for any target. The problem is rather one of which combination of images should be acquired. The same issue of computing which combination of images and slews is feasible is challenging (and solved on the ground). The EO-1 pointing problem is complex because EO-1 only has three reaction wheels, therefore as further observations are required momentum builds up on the reaction wheels that restricts later pointings of the spacecraft due to maximum rates that the reaction wheels can achieve. While this momentum can be relieved using a magnetic torque bar, this is a very slow process so observations are quite constrained by builtup angular momentum (for further details see [Chien et al. 2010]).

The CLASP [Rabideau et al. 2010, Doubleday et al, 2014] and Eagle eye [Knight et al. 2013] planners solve geometric coverage planning problems from a ground context. These systems can incorporate more complex geometric constraints but also have better computational resources and less time constraints.

AEOS [LeMaitre et al. 2002] is another project to perform automatic observation planning on the ground. AEOS solves a much more complicated and expressive problem in which the spacecraft slews while imaging to cover target polygons and the direction of the slew can be optimized to cover the polygon as efficiently as possible. In contrast we assume a framing imager and that the alignment of the imager is in a fixed aspect ratio with respect to a long track and across track to simplify the problem. We do this because our onboard computation capabilities are necessarily limited and also our response time for the scheduler correspondingly constrained.

In the future we plan on further maturing this work, refining the scheduling algorithms as well as bringing the work into a relevant hardware testbed.

Summary

We describe an overall software architecture for onboard imaging, image analysis, operations scheduling, and re-imaging within a realistic flight software operating system and flight hardware performance environment. This prototype demonstrated the feasibility of performing such functions autonomously within a low earth-orbiting environment (roughly 5-8 minutes overflight time). Future efforts will further mature this concept and software by bringing the prototype into a relevant flight hardware testbed.

Acknowledgements

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

Ball Aerospace, The WorldView spacecraft series, http://www.ballaerospace.com/page.jsp?page=294, *retrieved 18 Feb 2015.*

J.W.Boardman and F.A.Kruse,"Analysis of imaging spectrometer data using N-dimensional geometry and a mixture-tuned matched filtering (MTMF) approach," *TGARS*, vol. 49, no. 11, pp. 4138–4152, 2011.

DigitalGlobe, Worldview-3: DigitalGlobe, http://worldview3.digitalglobe.com**,** *retrieved 18 Feb 2015.*

S. Chien; Sherwood, R.; Tran, D.; Cichy, B.; Rabideau, G.; Castano, R.; Davies, A.; Mandl, D.; Frye, S.; Trout, B.; Shulman, S.; Boyer, D. 2005. Using Autonomy Flight Software to Improve Science Return on Earth Observing One. Journal of Aerospace Computing, Information, and Communication 2(4): 191–216.

S. Chien, R. Knight, A. Stechert, R. Sherwood, and G. Rabideau, "Using Iterative Repair to Improve Responsiveness of Planning and Scheduling," *Proceedings of the 5th International Conference on Artificial Intelligence Planning and Scheduling*, Breckenridge, CO, April 2000.

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.

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 2012, Stockholm, Sweden, June 2012.

Davies, A. G., S. Chien, V. Baker, T. Doggett, J. Dohm, R. Greeley, F. Ip, R. Castano, B. Cichy, R. Lee, G. Rabideau, D. Tran and R. Sherwood (2006) Monitoring Active Volcanism with the Autonomous Sciencecraft Experiment (ASE). Remote Sensing of Environment, Vol. 101, Issue 4, pp. 427-446.

T. Doggett, R. Greeley, A. G. Davies, S. Chien, B. Cichy, R. Castano, K. Williams, V. Baker, J. Dohm and F. Ip (2006) Autonomous On-Board Detection of Cryospheric Change. Remote Sensing of Environment, Vol. 101, Issue 4, pp. 447-462.

F. Ip, J. M. Dohm, V. R. Baker, T. Doggett, A. G. Davies, R. Castano, S. Chien, B. Cichy, R. Greeley, and R. Sherwood (2006) Development and Testing of the Autonomous Spacecraft Experiment (ASE) floodwater classifiers: Real-time Smart Reconnaissance of Transient Flooding. Remote Sensing of Environment, Vol. 101, Issue 4, pp. 463-481.

R. Knight, A. Donnellan, J. Green, Mission Design Evaluation Using Automated Planning for High Resolution Imaging of Dynamic Surface Processes from the ISS, *International Workshop on Planning and Scheduling for Space (IWPSS 2013).* Moffett Field, CA. March 2013.

Michel Lemaître,, Gérard Verfaillie, Frank Jouhaud, Jean-Michel Lachiver, and Nicolas Bataille. "Selecting and scheduling observations of agile satellites." *Aerospace Science and Technology* 6, no. 5 (2002): 367-381.

D. R. Thompson, B. Bornstein, S. Chien, S. Schaffer, D. Tran, B. Bue, R. Castano, D. Gleeson, A. Noell, Autonomous Spectral Discovery and Mapping Onboard the EO-1 spacecraft, IEEE Transactions on Geoscience and Remote Sensing. 2012.

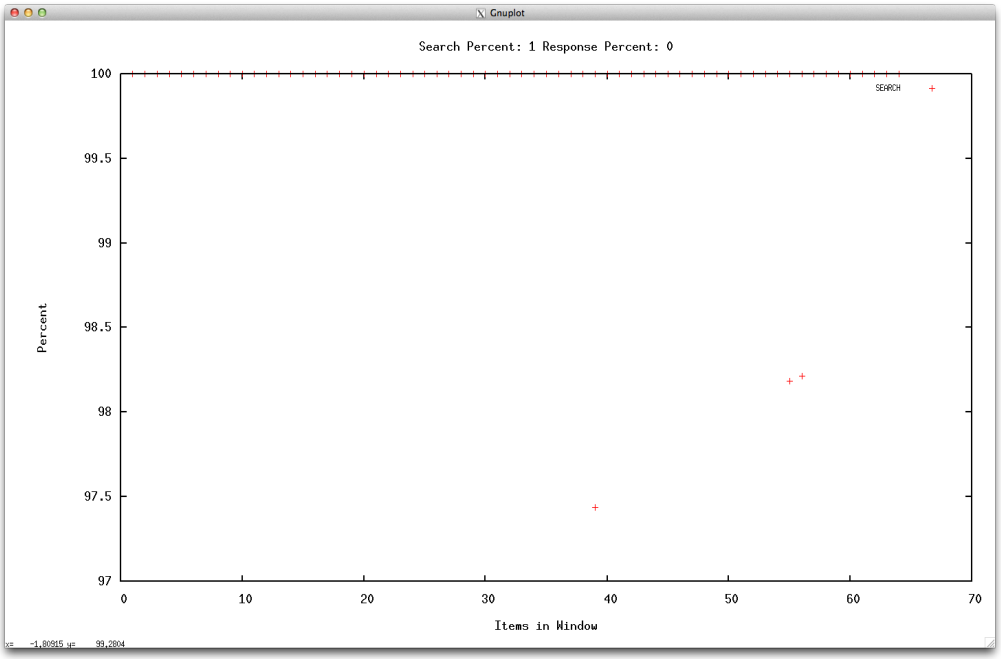
Stephen G. Ungar, Jay S. Pearlman, Jeffrey A. Mendenhall, and Dennis Reuter. "Overview of the earth observing one (EO-1) mission." *Geoscience and Remote Sensing, IEEE Transactions on* 41, no. 6 (2003): 1149-1159.

K. Weiss, "An Introduction to the JPL Flight Software Product Line", 2013 Workshop on Spacecraft Flight Software (FSW-13), Pasadena, CA, December 2013.

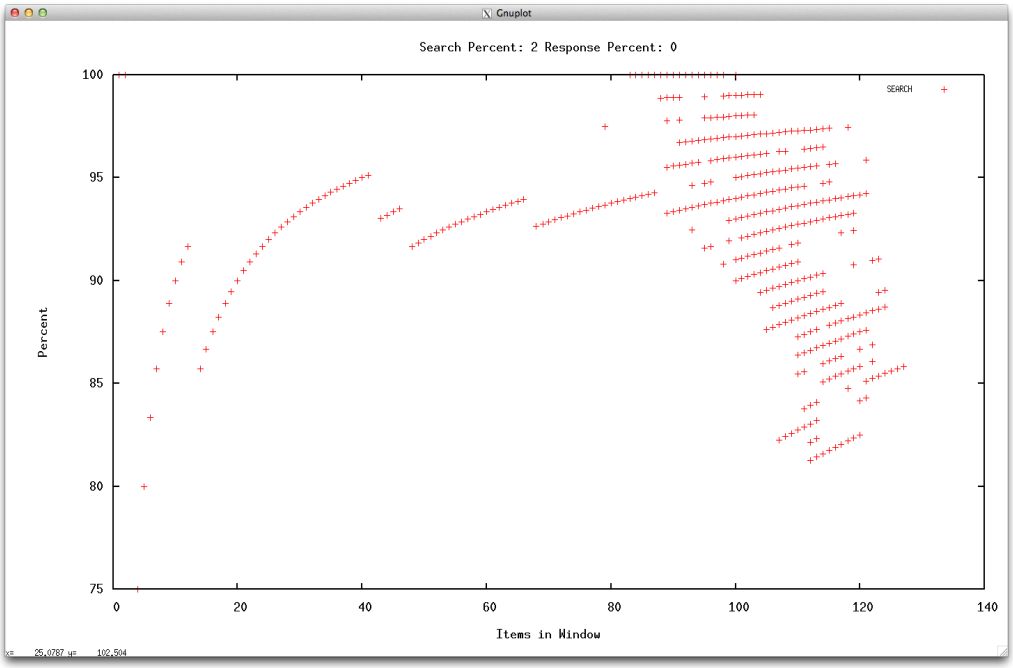
Wikipedia, Newtons Method, http://en.wikipedia.org/wiki/Newton%27s_method **,** *retrieved 10 Apr 2015.*

Wikipedia, Spot(satellite), http://en.wikipedia.org/wiki/SPOT_%28satellite%29**,** *retrieved 10 Apr 2015.*

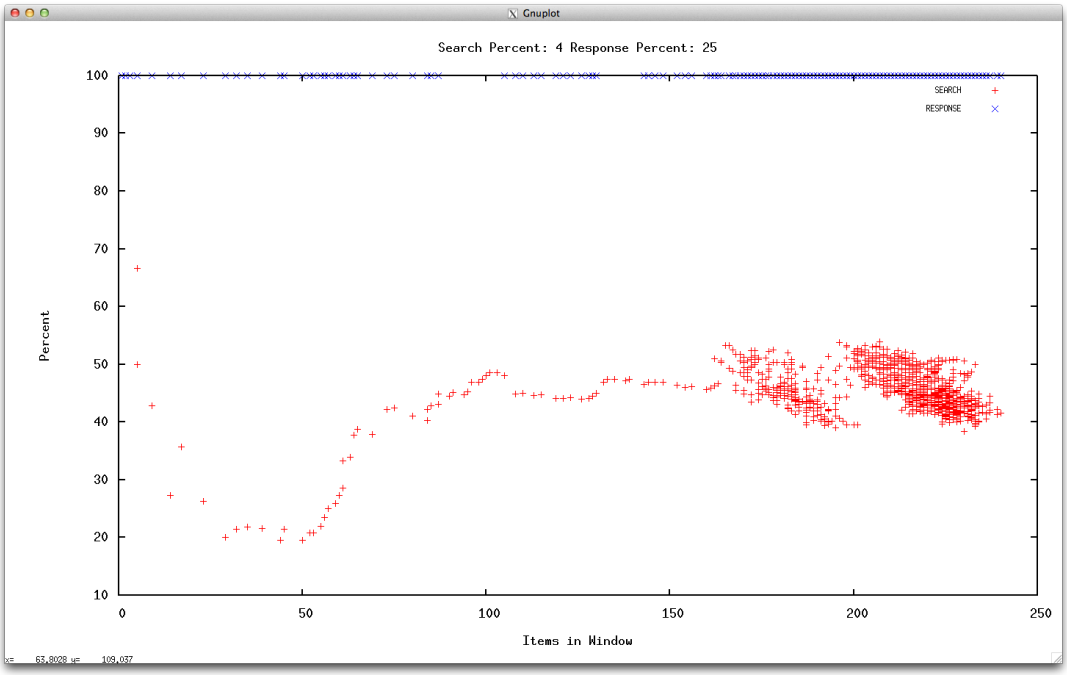
Wikipedia, Worldview-3, http://en.wikipedia.org/wiki/**WorldView-3,** *retrieved 18 Feb 2015.*



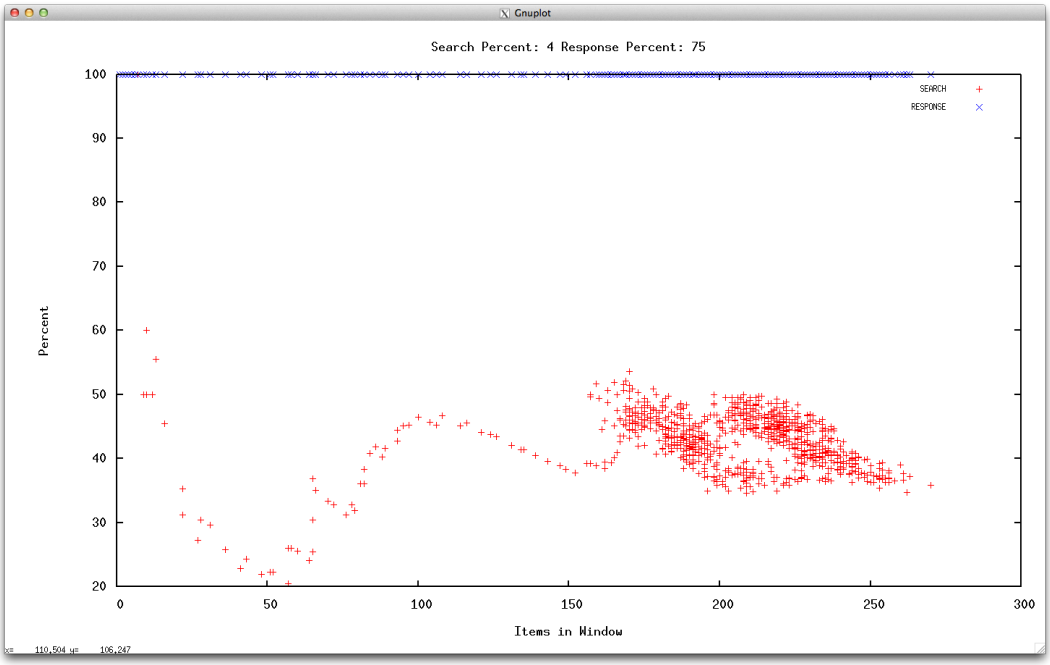
Graph 1: 1% search 0% response – almost all search requests scheduled
X axis is number of requests presented to scheduler



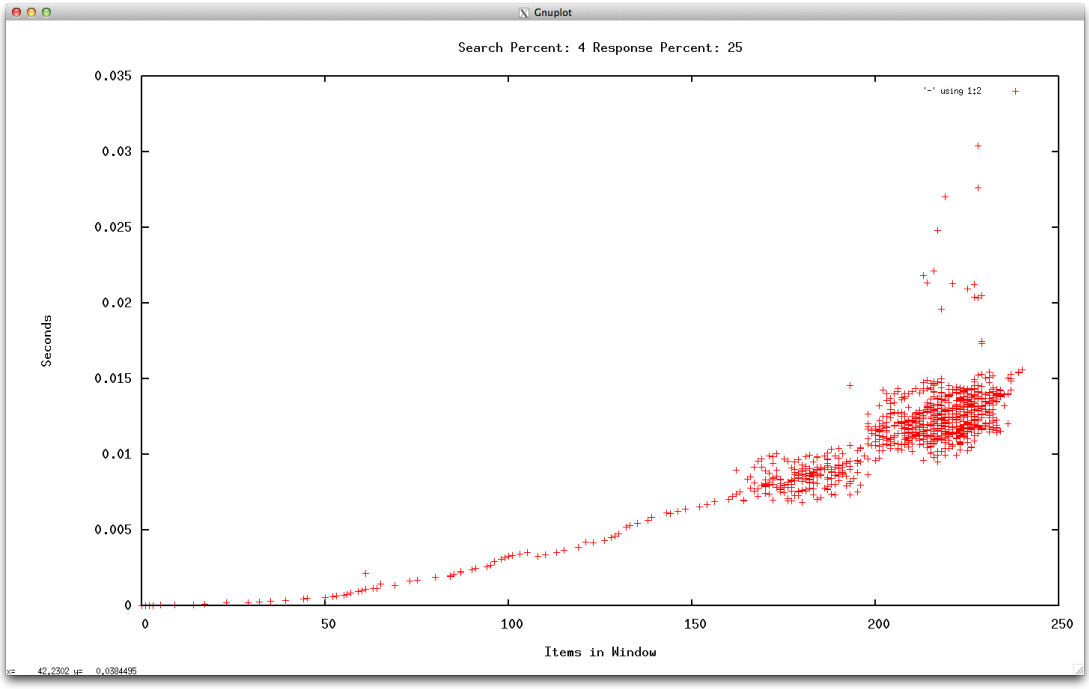
Graph 2: 2% search 0% response – some search requests already are not achieved.
X axis is number of requests presented to scheduler
Y axis is %-age of search requests scheduled



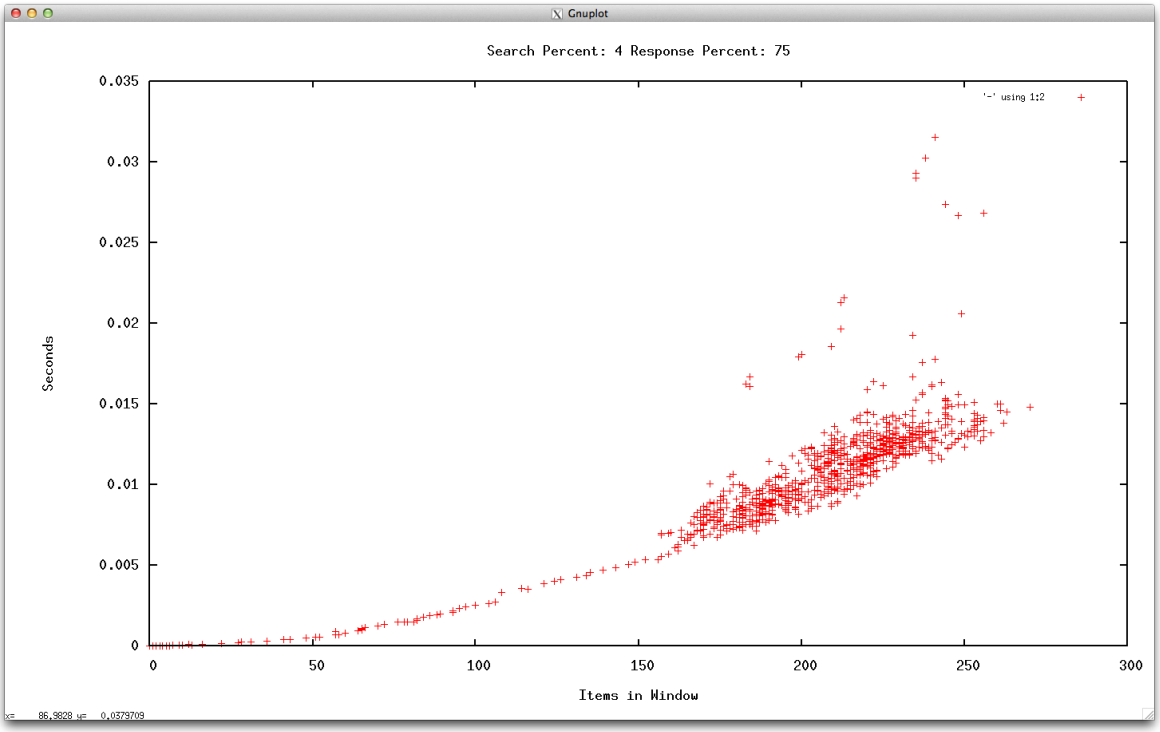
Graph 3: 4% search 25% response – response images are higher priority so all are getting scheduled. Search images are pre-empted and s/c loses time slewing between search and response windows.
X axis is number of requests presented to scheduler
Y axis is %-age of search (red) or response (blue) requests scheduled



Graph 4: 4% search 75% response – more response images are pre-empting search images.
X axis is number of requests presented to scheduler
Y axis is %-age of search (red) or response (blue) requests scheduled



Graph 5: 4% search 25% response
X axis is number of requests (search + response) presented to scheduler
Y axis is CPU time to construct schedule



Graph 6: 4% search 75% response
X axis is number of requests (search + response) presented to scheduler
Y axis is CPU time to construct schedule

COURSR: Scheduling Composite Educational Objects

Konstantinos Agnantis and Ioannis Refanidis

University of Macedonia, Dept. of Applied Informatics
Thessaloniki, Greece
kagnadis@gmail.com, yrefanid@uom.gr

Abstract

This paper presents COURSR, an online deployed system aiming at helping trainees (students, lifelong learners, etc) to decide whether they are able to attend an educational object (e.g., a university class, an online series of lectures, etc), taking into account their other commitments. COURSR encompasses a detailed description of every aspect of the educational object that demands effort, commitments or simply time from the trainee. For example, a course may have lectures, with constraints on their attendance or not, labs, mid-term exams, final exams, homework, projects and, of course, home reading. All this information is represented using a suitable ontology that has been developed in order to allow easy information exchange with other systems. Taking into account all this information, COURSR is able to analyze whether it is possible for the trainee to schedule a new course or a set of courses within his calendar or not. This is achieved using an existing powerful personal activities scheduling system, named SELFPLANNER. Taking informative decisions is important for the trainee, both for financial and for psychological reasons.

Introduction

Organizing personal time nowadays has become increasingly complex, due to the many tasks, commitments, deadlines and interleaving constraints that govern our life. Traditional calendars, either electronic or paper-based, have already reached the limits of their capabilities. Busy people maintain numerous of electronic calendars, trying to classify their tasks according to their purpose, e.g., business, leisure, family, personal, etc. People spend a lot of time to maintain all this information, which is a significant task by itself. Calendar information maintenance has two aspects. The first has to do with entering the information concerning the user's activities and commitments. The second has to do with rescheduling events, as new requests arise. Both these tasks are performed manually by most modern commercial electronic calendar applications.

In the recent years, there are research efforts that try to automate the scheduling and rescheduling part of an electronic calendar's maintenance. Research has focused on enhancing electronic calendars with powerful scheduling engines, which are capable to provide high-utility valid schedules, taking into account the constraints imposed by the user and the tasks, as well as the user's preferences. One example of such effort is the SELFPLANNER system (Refanidis and Alexiadis, 2011; Refanidis and Yorke-Smith, 2010; Refanidis, 2007), which works on top of Google Calendar, allowing the user to describe activities using a rich model of attributes, while it takes into account events directly added to the user's Google Calendar. Recent results have further increased the quality of the plans produced by the underlying scheduling engine (Alexiadis and Refanidis, 2013; 2015), using stochastic post-processing local search optimization methods.

However, while trying to alleviate the effort needed for scheduling and rescheduling a user's activities over his calendar, the effort needed to enter all the necessary information for each activity, in order for the scheduler to be able to schedule it, increases drastically. Particularly, the minimum information needed for an activity so that the scheduler to be able to schedule it, is its temporal domain (usually a set of temporal intervals), its duration and its location. Additional information that is useful in order to produce high utility plans includes its utility, potential user preference over the way the activity is scheduled by itself or in conjunction to other activities, the possibility to be scheduled in parts (preemptive scheduling) or in parallel to specific types of other activities, etc. Despite the fact that an intelligent user interface has been developed for the task of entering all the necessary information for each activity (Alexiadis and Refanidis, 2009), this task constitutes the main reason why end users are hesitant in adopting an intelligent calendar application like SELFPLANNER to organize their everyday activities.

One approach that has been adopted by some systems to overcome the problem of entering all the necessary information needed by the scheduler to produce useful and valid plans is to provide ready-to-use activity descriptions for specific application domains. This approach has been used by the MYVISITPLANNER system (Refanidis et. al., 2014), an intelligent application that helps visitors and residents of the area of Northern Greece to select cultural activities and schedule them within their calendar. Activity descriptions in MYVISITPLANNER are relatively simple, comprising a temporal domain, a duration range and a location. Complex activities take only the form of bundles of simple activities, e.g., visiting two museums within a specific time period (e.g., the same day). MYVISITPLANNER employes the scheduling engine of SELFPLANNER,

provided as a web service, to solve the underlying scheduling problem.

This paper presents COURSR, another prototype system that adopts a similar approach to MYVISITPLANNER, but applies it in the educational domain. Activities in the educational domain are inherently more complex than, e.g., visiting a museum. A single educational activity may include synchronous and asynchronous tasks, deadlines, temporal constraints, hard and soft constraints, cardinality constraints (not even supported by SELFPLANNER), etc. Describing all this information for an educational object is a tedious task and, furthermore, this information is not usually provided by those offering the educational activity. Thus, even if the user wanted to enter all this information into his calendar, this information would not be available.

An educational activity may range from simple ones, like attending a one-day tutorial (either with physical presence or online) or reading a book, to very complex ones, like attending a university class, enrolling to a master degree or learning a foreign language or a musical instrument. Having access to this information before deciding to adopt an educational activity is crucial for the trainees in order to take informative decisions. Problems that arise from non-informative decisions range in two dimensions (Moka and Refanidis, 2010): First, enrolling with a course usually incurs paying significant tuition fees for the student, so a potential failure implies financial harm. However, of equally importance is the psychological harm that a student experiences in case he cannot successfully complete the undertaken course.

The COURSR system tries to solve exactly this problem. It allows the course instructors or the owners of simple educational objects to describe in fine-grained detail all sorts of time requirements incurred for a student if he decides to attend one or more courses. This is achieved through a rich ontology for describing all temporal aspects of a course. Furthermore, COURSR allows exporting such descriptions in various ontology formats, in order to be used or simply validated by other systems. By having access to all this information, the potential student is able to perform a what-if analysis, that is, check whether adding one or more educational objects into his calendar is feasible. For this purpose, COURSR employs the SELFPLANNER scheduling engine, offered as a web service that returns either a valid plan or failure.

The rest of the paper is structured as follows: First we outline how educational objects are described within COURSR and we present a typical student's session with the system. Next, we describe the integration of SelfPlanner scheduling services to the system. Then we give some insight of the underlying ontology and the COURSR SPARQL endpoint. Finally, we conclude the paper and identify directions for future research.

System Overview

COURSR is currently available at http://coursr.herokuapp.com/. Its starting page is shown in Figure 1.

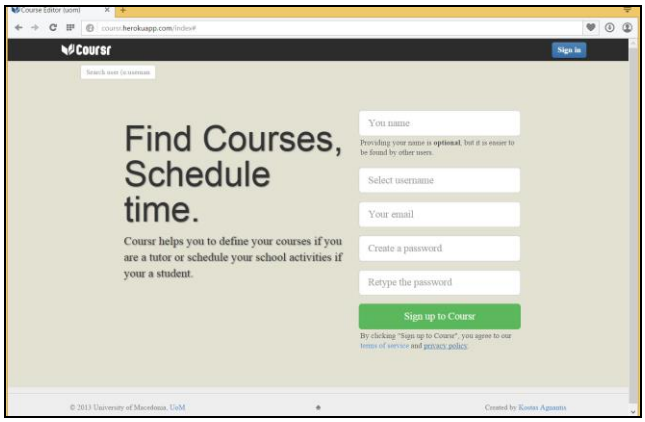


Figure 1: The start page of COURSR

Users can create local accounts (SignUp) or login using an Open ID account (Google, Facebook, LinkedIn and StackExchange are supported). In its current version the system has some limitations, which we are working constantly to overcome. The most severe of them is that there is a single interface for tutors and students, that is, everybody can add new courses into the system, as well as inspect and schedule existing courses (added by other users) within his calendar. Another limitation concerns the number of calendars supported for every user: Currently, a single calendar per user is only supported, so if a user maintains events in multiple calendars, these are ignored while scheduling courses.

The main screen of COURSR, after logging in, shows the dashboard containing recent actions performed by the user, like the new courses he created (if he is a tutor) or the new courses that he (as a student) enrolled in. (Figure 2).

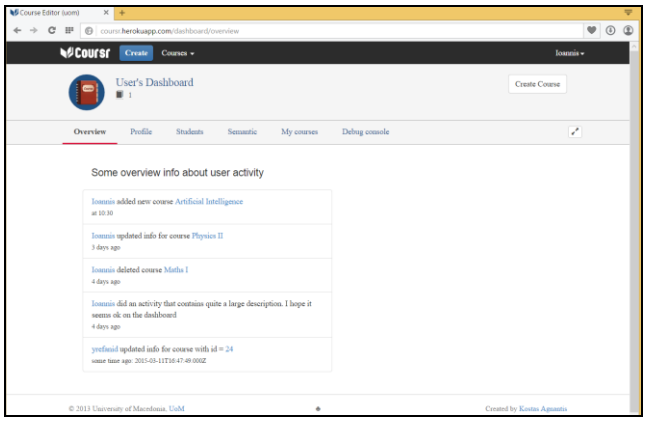


Figure 2: The COURSR dashboard

In order to create a new course, the user starts by clicking the "Create Course" button. In the first screen, the basic course information should be added, namely its title, the name of the instructor, the semester when the course is offered, its start and end dates, as well as an optional description. Note that no course activity is allowed to take place outside the temporal window defined from the "starts on" and "ends on" fields, including exams (Figure 3).

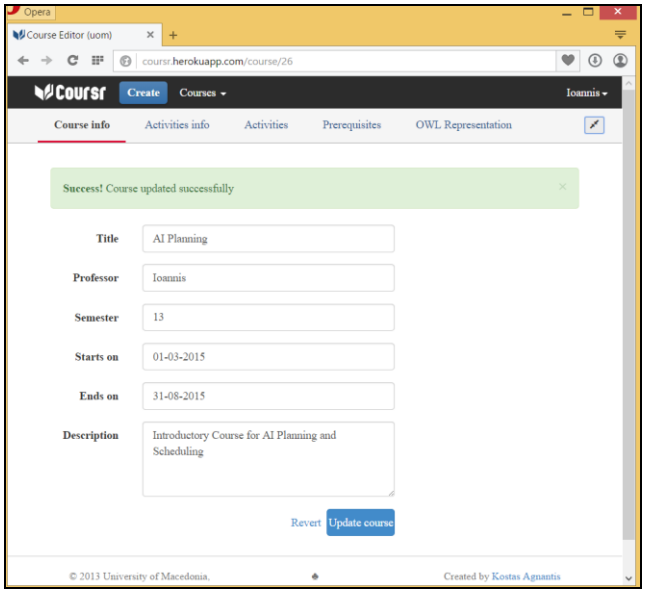


Figure 3: Basic information for the newly created course

Having inserted the basic course information, the next step is to describe its individual activities. By selecting "Activities Info" in the main window, the user can add any number of activities for the course. COURSR supports six (6) types of activities:

- Lecture
- Lab
- Midterm exam
- Final exam
- Presentation
- Homework

For each one of these types of activities, the user can enter the following information:

Workload (in mins): The time required by the student to get prepared in advance, in order to be ready for the activity. For example, for a weekly lecture, the student may need 60 mins for preparation, in order to take the most from attending the lecture, whereas for the final exams the student may need 12 hours to prepare in order to success in the exams. This workload concerns asynchronous time, that is, a student could allocate the workload at the time of his convenience, taking into account his other commitments.

Duration (in mins): The actual duration of the activity. For example, a lecture may last 90 mins, whereas the final exams 180 mins. This is synchronous time, that is, a student has no flexibility as of when to schedule this time.

Periodicity: In this field the instructor defines the periodicity of the specific activity of the course. This is a text field and the text entered by the instructor is analyzed by a dedicated parser in order to extract the intended periodicity. Examples of valid periodicity descriptions are the following:

- Every Monday at 15:00
- Every second Monday at 12:00
- Every Monday at 12:00 and Tuesday at 14:00
- Every Monday at 15:00 up to 31/5/2015
- Every Monday at 18:00 from 10/2/2015 to 20/4/2015
- On 13/4/2015 at 20:00
- Every Monday at 12:00 excluding 27/4/2015 including 28/4/2015 at 20:00

For each valid periodicity description, in the field "Auto-completed dates" are shown the dates of the individual occurrences of the event (Figure 4).

The instructor can create multiple activities for a course, of any one of the six available types, which are shown at the bottom area of Figure 4. They can be edited, updated and deleted at any time.

From the tab "Activities" the instructor can edit each individual activity, by assigning a title, a description and a comment to it. For example, concerning the multiple lectures created in Figure 4, an individual title, description and comment could be added to any one of them (Figure 5).

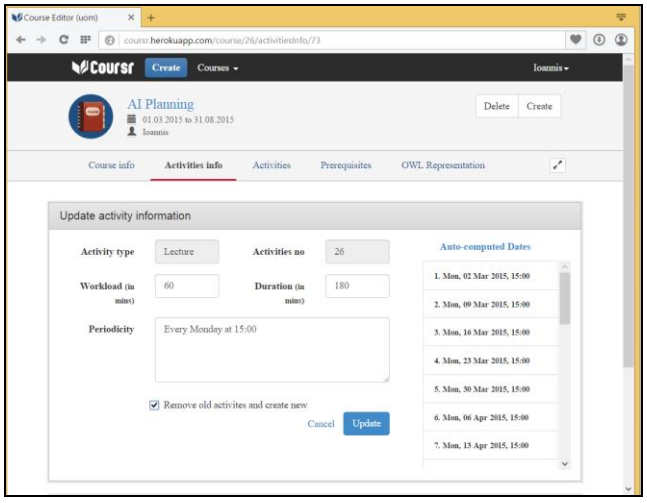


Figure 4: Duration and periodicity of an activity that is part of a course

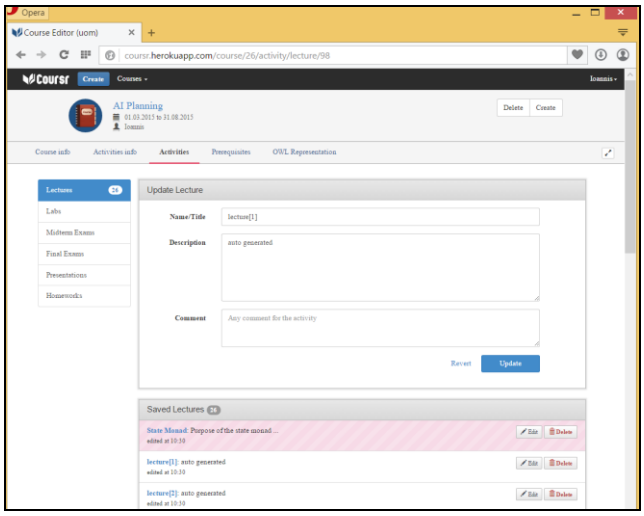


Figure 5: Editing individual activities

The instructor can also add special requirements in order for successfully completing a course. Three types of requirements are supported by the current version of COURSR:

- Course prerequisites
- Attendance prerequisites
- Activity prerequisites

Course prerequisites concern the fact that in order for a student to attend this course, he should attend another course in advance. In the field "Course prerequisites" (Figure 6) the instructor has simply to write the name (s) of the pre-required course (s).

Attendance prerequisites concern the minimum attendance requirements per activity in order for the attendance of a course to be considered successful. For example, for a course with 13 lectures, an attendance prerequisite may impose that the students should attend at least 10 out of the 13 lectures, or that the students should deliver at least the second and third homework otherwise they fail.

Activity prerequisites allow the instructor to impose ordering constraints between asynchronous activities of the course. As an example, in order for a student to take part to the mid-term exam, he should have participated to at least 3 lab sessions.

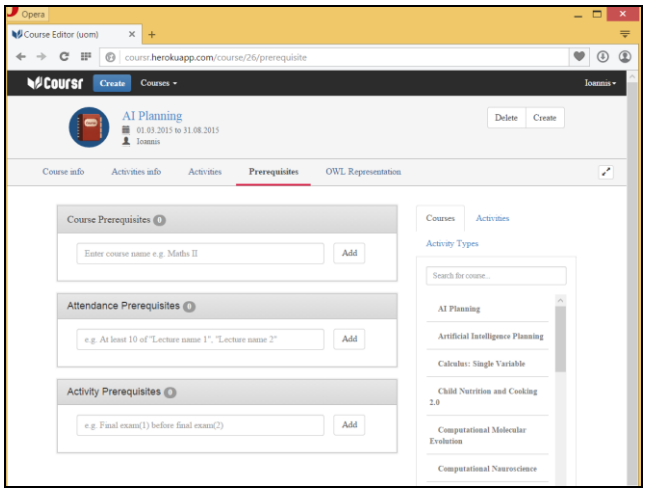


Figure 6: Imposing prerequisites in order to successfully complete a course

Finally, the instructor can export the course description, with all of its activities and prerequisites, in some ontological format (Functional, Turtle, RDF/XML and simple XML are supported).

Currently an RDBMS (PostgreSQL) is used to store all the data of the application, mainly for performance and efficiency reasons. However, the application gives the option to the tutor to additionally persist all the information of each course to an on-disk triple-store in RDF format, using the Sesame API (Brickley and Miller, 2012). By doing that, this information becomes available to be queried on later, by not only the users of the system, but by external users or other system, through the SPARQL endpoint provided by COURSR.

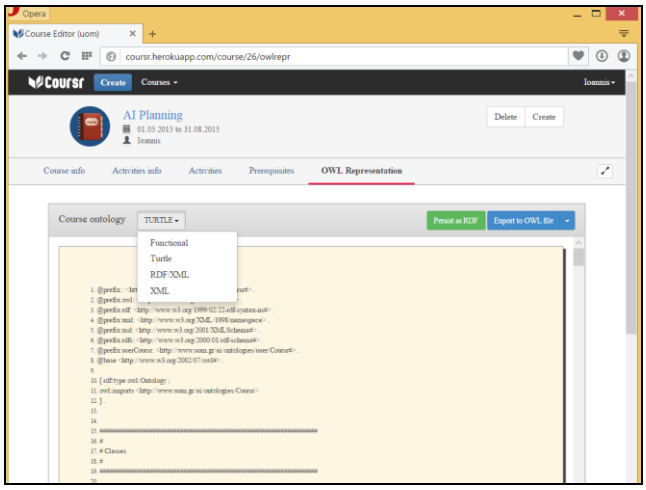


Figure 7: Exporting a course description in ontological format

Student experience

Students using COURSR need to grant the system access to one of their Google calendars. This is important for two reasons: First, COURSR will use this calendar in order to put there information about the scheduled activities of the selected courses, that is, when the student needs to accomplish a synchronous or an asynchronous activity. Second, COURSR reads other (non-COURSR) activities already in this calendar, and considers them as busy time, thus avoiding to schedule there any course-related activity.

To inform COURSR about his calendar, the user selects "Profile" through a popup menu, by pressing on his name in the upper right corner of the application. In Figure 8, the user has the option to give access to his basic Google calendar, provided that he has one, by just giving the Calendar in the relevant field. ID.

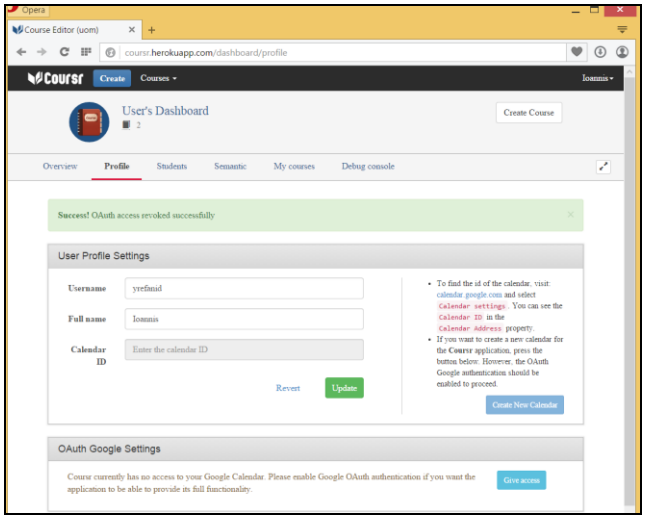


Figure 8: Editing a user's profile

Alternatively, if the user does not want to use his existing Google calendar, he can create a new one and grant access to it. In either case, this access to the calendar can be revoked at any time.

Students are interested in enrolling to courses. To see a list of available courses, they have to select "Students" from the main menu of Figure 8, in order to retrieve the courses list. They can filter the list of available courses by using the related fields in the top-right corner and they can subscribe to any number of courses by clicking on "Subscribe" buttons.

Having subscribed to a number of courses, the student can focus on them through the tab "My courses" (Figure 9).

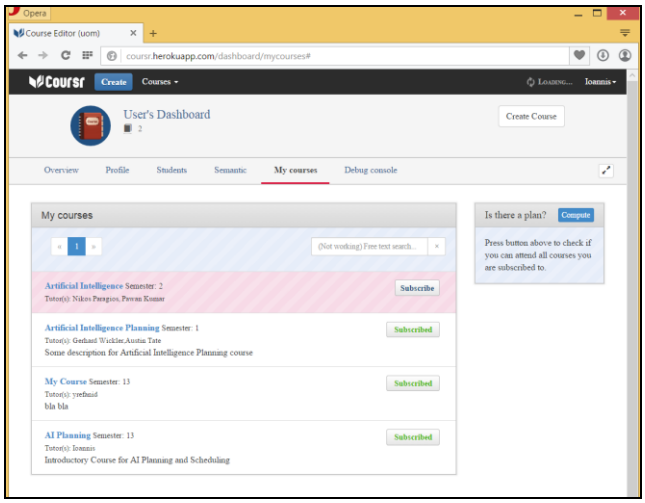


Figure 9: List of subscribed courses and consistency check

By pressing the button "Compute" in Figure 9, COURSR sends all the necessary information for the subscribed courses to the scheduling engine of SELFPLANNER and receives either a plan with all the necessary course activities scheduled in it, or a failure. In this way, the student is informed about whether he is capable to subscribe (enroll) simultaneously in the selected courses or not, a piece of information that is of great value for his subsequent decisions. In case the student receives a negative answer, he can unsubscribe from some of the selected courses and try to schedule the remaining courses again. In this way, he can examine several scenarios concerning what are the combinations of courses he can afford, from a time availability point of view.

SELFPLANNER Integration

COURSR takes advantage of the powerful scheduling engine SELFPLANNER to provide plans to students. SELFPLANNER is a general-purpose activity scheduler unaware of the specific prerequisite Coursr is using in course definitions. However, it supports various types of unary and binary constraints that might apply to single activities and pairs of activities, respectively (Refanidis and Yorke-Smith, 2010). *Ordering constraints* are used when an activity needs to be scheduled before another, *implication constraints* might exist between activities, usually donating prerequisites (e.g. Traveling to Jerusalem, presumes booking a flight reservation) and *proximity constraints* which imply minimum/maximum temporal distance between a pair of activities.

COURSR is using its pre-processing engine to convert course prerequisite to unary and binary constraints supported by SELFPLANNER. COURSR *Activity Prerequisites* can be encoded using the *implication constrains* supported by SELFPLANNER. For example, if the student should have taken the Mid-term exam in order to

be eligible to participate to the Final exam, then we define the implication rule "`Final-exam Activity ⇒ Mid-term exam Activity`". This constraint conveys to SELFPLANNER the information that in order to produce a valid plan, it must add the *Mid-term exam* activity, in case *Final-exam* activity is also present in the final plan.

There are activities that should be scheduled, before, after or between other activities. For example, *Studying for the Final exam* should always proceed the *Final exam* activity, while *Studying for Lecture C* should be scheduled some time between the end of *Lecture B* and the start of *Lecture C*. We can pass these kind of information to the SELFPLANNER using one of the following ways:

- Pre-computing the valid time domain of the activity to be scheduled and give it directly to the SELFPLANNER.
- Using *Ordering constraints* mechanism to indirectly limit the time domain of the activity and let SELFPLANNER handle the constraint for us.

The use of ordering constraints relieve from the burden of pre-computing the domain of the activity. A problem with how ordering constraints work in SELFPLANNER, is that in case of a constraint of the form `A < B < C` (A proceeds B and B proceeds C), if SELFPLANNER does not insert to a plan the activity A, this will indirectly change the temporal domain of the activity B; The starting time of B's temparal domain will become the same as the starting time of the plan domain itself. For example, activity B may be scheduled 2 months before the scheduled time of the activity C. To overcome this problem, *proximity constraints* are used in conjunction with *ordering constraints*, by defining the max distance in time between activity B and activity C to be less than a specific time (e.g. a week in case of week lectures)

The majority of the activity types COURSR is using, have inelastic temporal domains, meaning that they should be scheduled only on specific time and with specific duration. For example *Lecture C* will takes place at 13 May 2015, at 13:00 and its duration will be 2 hours. In these cases, COURSR will create and pass to SELFPLANNER an activity with a strict temporal domain (13/5/2015,13:00 - 13/5/2015,15:00). SELFPLANNER will either add the activity in that time spot (if that is possible), or it will not include it at all to the final plan.

A feature provided by COURSR but not supported by the SELFPLANNER specifications, is the ability to define cardinality constraints. For example, in order to participate to the *Final exam*, the student should have submitted *3 out of the 5 homework* throughout the semester. Currently, there is no way to communicate this information to the SELFPLANNER; however in the aforementioned example a plan with 3 homework activities should be considered valid, while a plan with should not.

A way to approach this desirable behavior is by applying different *utility* values for each activity. SELFPLANNER always tries to compute a plan with the highest possible *cumulative utility* (i.e., the sum of the utilities of all activities included in the plan). The *utility* value which COURSR will give to an activity will be determined by the significance of the activity to a valid plan. For example, the activities of the prerequisite "*Student needs to submit 8 out of 10 homeworks*", will have larger utility values, from the ones of the prerequisite "*Student needs to attend at least 3 out of 10 lectures*". Activities of the first case will have larger priority, compared to the ones of the second prerequisite during the plan creation, increasing the probability the plan to satisfy both prerequisites. In case of a failure in finding a valid plan, COURSR will update the *utility* values of the activities and re-submit a request for a plan. COURSR users can specify (through application preferences) the max number of tries, before COURSR stop requesting for new plans.

The COURSR Ontology

The COURSR ontology was developed as an essential part of the COURSR system and aims at representing and describing the various activity types of an educational domain (e.g. lecture, exam), their relationships (e.g. to enroll in course B, you should have completed successfully course A) and the information that defines them (e.g., what is the subject of a course or lecture, when is the final exam date). The ontology is consisted of:

- 41 classes (owl:Class)
- 53 named individaul objects - including examples (owl:NamedIndividual)
- 33 object properties (owl:ObjectProperty)
- 37 data properties (owl:DataProperty)
- 49 subclass relations (rdfs:subClassOf)
- 14 equivalent classes (owl:equivalentClass)

Some of the more prominent classes of the ontology are the following:

Course: This class can be considered as the "root" of the whole ontology, as it uses explicitly or implicitly almost all the other classes. It is used to encode all the information a course may consist of, like the various activities it includes, the starting/ending dates of the course etc.

SchoolActivity: This class represents a generic school activity. As previously described, in our domain we define 6 types of activities, each one being a subclass of the main class *SchoolActivity*.

SchoolActivitySynopsis: Contrary to *SchoolActivity* class, this class is used to represent information not about a single activity, but meta-data info relative to the whole set of all activities of the same type. For example, a *LectureActitySynopsis* (subclass of *SchoolActivitySynopsis*) contains information like the number of a Lectures of a specific course, the periodicity of all lectures, or the

duration of them. As with *SchoolActivity*, for each activity type there is a corresponding *ActivitySynopsis* subclass.

Periodicity: This class encodes the various periodicity types. Currently, our ontology supports 4 types of an event (activity) periodicity: One time, some-times, week and month periodicity.

Prerequisite: This class contains 3 subclasses, each one representing a different type of prerequisite (course, activity and attendance) our application supports.

User: This class is used to represent information about the two types of users: Student and Tutor. In the future, we intend to integrate this class with elements of other established ontologies like FOAF (Broekstra, Kampman and Harmelen, 2002).

Figure 10 gives an overview of COURSR ontology.

Figure 10: A graphical representation of the COURSR ontology

SPARQL endpoint

COURSR provides a SPARQL endpoint, in order to give the users the option to access its database in an ontological way, using SPARQL queries (Figure 11). The full SPARQL language is supported, with a link to its documentation. In the default page, the user can see the first 100 triples stored on the RDF database, and the respective SPARQL query. Of course he can change the query at will. Additionally, all results presenting in the result table are clickable. By clicking on a subject, property or object the query is being updated, returning all the statements where the specific resource is the subject of them. Additionally, the user can get the results of a query in csv format.

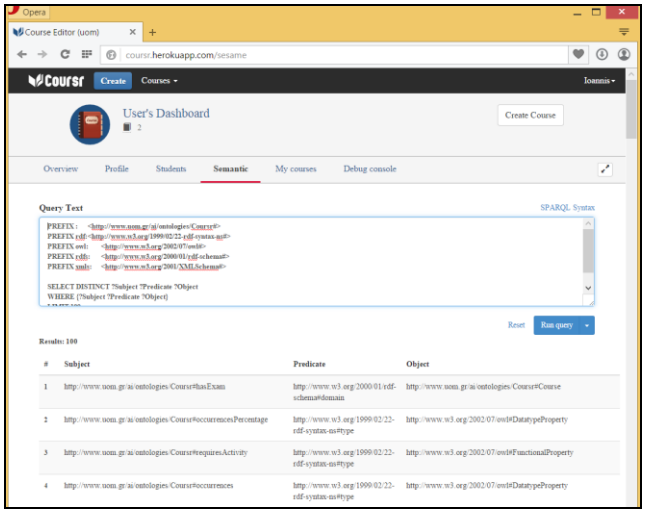


Figure 11: Accessing COURSR RDF database through its SPARQL endpoint

Related Work

To the best of our knowledge there is no other mature research results or just some research prototype trying to focus on describing the workload imposed by educational objects and exploit this information through a powerful scheduling engine in order to allow the prospective student to examine various scenarios concerning whether he has the necessary resources to undertake new educational commitments or not.

Of course, there are several research efforts trying to help people to organize their time. One of them, already mentioned in this paper, is SELFPLANNER, which serves as the scheduling engine of COURSR. There are many systems developed over the last two decades that mainly focus on automated meeting scheduling. Some of them concentrate on specific aspects of this problem (Garrido and Sycara 1995; Jennings and Jackson 1995; Sen and Durfee 1994; Sen and Durfee 1998). More recent efforts tend to incorporate learning components or to integrate with the

Semantic Web. For example, RCal (Singh 2003) is an intelligent meeting scheduling agent that assists humans in office environments to arrange meetings. RCal agents negotiate with each other on behalf of their users and agree on a common meeting time that is acceptable to all users and abides by all the constraints set by all the attendees. RCal supports parsing and reasoning about semantically annotated schedules over the web, such as conference programs or recurring appointments (Payne, Singh and Sycara 2002).

CMRadar (Modi et al. 2004) is an end-to-end agent for automated calendar management that automates meeting scheduling by providing a spectrum of capabilities ranging from natural language processing of incoming scheduling-related e-mails, to negotiate with other users or making autonomous scheduling decisions.

PTIME (Berry et al. 2006) is an ongoing effort being developed under the CALO project (Myers 2006), that aims at facilitating meeting scheduling. The innovation of PTIME lies at its capability to learn the user's preferences thus adapting its future behavior, whereas it emphasizes in adopting natural language for interfacing with the user. Part of the work in the PTIME thrust is Emma, a personalized calendar management assistant designed to help its user handle email meeting requests, reserve venues and schedule events. Emma interfaces with commercial enterprise calendaring platforms and operates seamlessly with users who do not have Emma. It is designed to learn scheduling preferences, adapting to its user over time. The system is in initial deployment at SRI International.

Another effort within the CALO project concerns Towel (Conley and Carpenter 2007), an initial attempt towards an intelligent to-do list. Towel allows the user to organize to-dos (group, tag, check, etc) as well as delegate them to other users or agents. Although to-dos can be seen as tasks, Towel emphasizes on to-dos manipulation rather than in solving the scheduling problem associated with actually performing these to-dos.

Conclusions and Future Work

COURSR is an innovative prototype application aiming at integrating educational data with intelligent calendar applications. Intelligent calendar applications face the problem of lacking data of enough expressivity, in order to apply their sophisticated constraint optimization algorithms. People usually consider entering all this necessary information as not worthy the time needed to enter this information in order to get better schedules. So, they avoid using intelligent calendar applications, preferring to stay with traditional electronic calendars, even with paper-based ones.

The solution to this problem lies in defining standards that can be used to provide ready-to-use activity descriptions for a plethora of application domains. This resembles existing calendar standards that are used for exchanging calendar information, like iCal, however with much more expressive power, in order to give people the ability to describe complex activities with a variety of interactions among them, as well as end user constraints and preferences over the way the activities are scheduled in time and space. Having such standards in hand, it would be easy to define complex activity descriptions for a variety of domains like, e.g., going shopping, attending classes or having some leisure time.

In this work we identified the education domain as a promising candidate to be used for describing temporal activity descriptions with complex structure. Educational activities may have significant structure and complexity, which is very hard to be entered manually in a user's calendar. So, providing ready-to-use activity descriptions for the educational domain may greatly facilitate the students to keep track of their duties, manage their time effectively and take informative decisions.

Our future plans concerning COURSR include extending its database with more educational objects, by trying to collect such data using crowd-sourcing techniques. Particularly, the best candidate to feed COURSR's database are the instructors themselves. By providing them a simple widget to be included in the courses' web pages, allowing them to insert workload data for their courses and having them stored in COURSR's database, would be beneficial for all involving parts (students, instructors and COURSR).

Other potential extensions include underlying COURSR's ontology, which could be connected with other well-known ontologies and be extended even further to express more complex interactions between parts of an educational object.

Acknowledgement

This research has been co-financed by the European Union (European Social Fund – ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Program: Heracleitus II. Investing in knowledge society through the European Social Fund.

References

Alexiadis, A. and Refanidis, I. Post-Optimizing Individual Activity Plans through Local Search. In Proceedings of ICAPS 2013 Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems (COPLAS), pp. 7-15. Rome, 2013.

Alexiadis, A., and Refanidis, I. "Defining a Task's Temporal Domain for Intelligent Calendar Applications". In Proceedings of the 5th IFIP Conference on Artificial Intelligence Applications & Innovations (AIAI 2009), pp. 399-406, Springer. Thessaloniki, Greece.

Alexiadis, A., and Refanidis, I. Optimizing Individual Activity Personal Plans through Local Search. Accepted for publication by AI Communications, IOS Press, 2015.

Berry, P., Conley, K., Gervasio, M., Peintner, B., Uribe T. and Yorke-Smith, N. Deploying a Personalized Time Management Agent, 5th International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-2006), Industrial Track, pp. 1564-1571.

Brickley, D. and Miller, L.: "FOAF vocabulary specification 0.98." Namespace document 9 (2012).

Broekstra, J., Kampman, A., and Harmelen, F.V.: "Sesame: A generic architecture for storing and querying rdf and rdf schema." The Semantic Web—ISWC 2002. Springer Berlin Heidelberg, 2002. 54-68.

Conley, K., and Carpenter. Towel: Towards an Intelligent To-Do List, AAAI Spring Symposium on Interaction Challenges for Artificial Assistants, Stanford, CA, March 2007.

Garrido, L. and Sycara, K. Multi-agent meeting scheduling: Preliminary experimental results, 1st International Conference on Multi-Agent Systems (ICMAS), 1995.

Jennings N.R. and Jackson, A.J. Agent based meeting scheduling: A design and implementation, *IEE Electronic Letters*, 31(5):350-352, 1995.

Joslin, D.E., and Clements, D.P. "Squeaky Wheel" Optimization, *Journal of Artificial Intelligence Research*, vol. 10 (1999), 375-397.

Modi, P.J., Veloso, M., Smith, S.F. and Oh, J. CMRadar: A Personal Assisstant Agent for Calendar Management, Workshop on Agent Oriented Information Systems (AOIS), New York, 2004.

Moka, E. and Refanidis, I. "Towards Intelligent Management of a Student's Time". In Proceedings of the 6th Greek Conference on Artificial Intelligence (SETN-2010), Athens, Greece. Springer LNAI 6040, pp. 383–388.

Myers, K. Building an Intelligent Personal Assistant, AAAI Invited Talk, 2006.

Payne, T.R., Singh, R. and Sycara, K. Calendar Agents on the Semantic Web, IEEE Intelligent Systems, vol. 17(3), pp84-86, May/June 2002.

Refanidis, I. Managing Personal Tasks with Time Constraints and Preferences, 17th International Conference on Automated Planning and Scheduling Systems (ICAPS-2007), Providence, Rhode Island, 2007.

Refanidis, I., and Alexiadis, A., "Deployment and Evaluation of SelfPlanner, an Automated Individual Task Management System". *Computational Intelligence*, 27(1), 41-59, 2011.

Refanidis, I., and Yorke-Smith, N. "A Constraint Based Programming Approach to Scheduling an Individual's Activities". *ACM Transactions on Intelligent Systems and Technologies*, vol. 1 (2), 2010.

Refanidis, I., Emmanouilidis, C., Sakellariou, I., Alexiadis, A., Koutsiamanis, R.-A., Agnantis, K., Tasidou, A., Kokkoras, F., and Efraimidis, P.S. MYVISITPLANNER^{GR}: Personalized Itinerary Planning System for Tourism. In Proceedings of the Hellenic Artificial Intelligence Conference (SETN), Ioannina, Springer-LNCS 8445, pp. 615-629, 2014.

Refanidis, I., McCluskey, T.L. and Dimopoulos, Y. Planning Services for Individuals: A New Challenge for the Planning Community, Workshop on Connecting Planning Theory with Practice, Whistler, British Columbia, Canada, 2004.

Sen, S. and Durfee, E.H. A formal study of distributed meeting scheduling, *Group Decision and Negotiation*, vol.7, pp. 265-289, 1998.

Sen, S. and Durfee, E.H. On the design of an adaptive meeting scheduler, 10th International Conference on Artificial Intelligence for Applications, pp. 40-46, 1994.

Singh, R. RCal: An Autonomous Agent for Intelligent Distributed Meeting Scheduling, master's thesis, tech. report CMU-RI-TR-03-46, Robotics Institute, Carnegie Mellon University, December, 2003.