28th International Conference on
Automated Planning and Scheduling

June 24-29, 2018, Delft, the Netherlands

SPARK 2018

Proceedings of the 11th Workshop on

Scheduling and Planning Applications woRKshop (SPARK)

Edited by:

Sara Bernardini, Simon Parkinson, Kartik Talamadupula

Preface

This volume contains the papers presented at SPARK 2018: 2018 Scheduling and Planning Applications woRKshop held on June 24-29, 2018 in Delft. There were 11 submissions. Each submission was reviewed by at least 3 program committee members. The committee decided to accept 9 papers. The program also includes 2 invited talks.

Application domains that entail planning and scheduling (P&S) problems present a set of compelling challenges to the AI planning and scheduling community that from modeling to technological to institutional issues. New real-world domains and problems are becoming more and more frequently affordable challenges for AI. The international Scheduling and Planning Applications woRKshop (SPARK) was established to foster the practical application of advances made in the AI P&S community. Building on antecedent events, SPARK 2018 is the eleventh 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 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 2018 workshop and publication chairs for their support.

Sara Bernardini, Simon Parkinson, and Kartik Talamadupula
The SPARK 2018 Organizers

June 14, 2018 Sara Bernardini
London

Program Committee

Laura Barbulescu	Carnegie Mellon University
Anthony Barrett	NASA
Sara Bernardini	Royal Holloway University of London
Mark Boddy	Adventium Labs
Lukas Chrpa	Czech Technical University in Prague
Gabriella Cortellessa	CNR-ISTC, National Research Council of Italy
Minh Do	NASA
Simone Fratini	European Space Agency - ESA/ESOC
Mark Giuliano	Space Telescope Science Institute
Christophe Guettier	SAFRAN
Patrik Haslum	ANU
Angelo Oddi	ISTC-CNR, Italian National Research Council
Simon Parkinson	University of Huddersfield
Nicola Policella	ESA/ESOC
Cédric Pralet	ONERA Toulouse
Riccardo Rasconi	ISTC-CNR
Bram Ridder	King's College London
Kartik Talamadupula	IBM
Mauro Vallati	University of Huddersfield
Tiago Stegun Vaquero	NASA Jet Propulsion Laboratory, Caltech
Neil Yorke-Smith	Delft University of Technology
Terry Zimmerman	University of Washington -Bothell

Author Index

Keyword Index

Table of Contents

Personalised Medication Planning using PDDL+

Fares K. Alaboud and **Andrew Coles**
Department of Informatics, King's College London, UK
email: firstname.lastname@kcl.ac.uk

Abstract

Prescription medication is typically prescribed with a standardised set of instructions, to be followed regularly, with the aim being to manage symptoms while remaining within safe dosage limits. The caveat of such standardisation is that it is not tailored to the needs of the patient, in terms of their activities. In this paper, we take the first steps towards modelling medication pharmacokinetics as a PDDL+ hybrid planning problem. As pharmacokinetics are inherently non-linear, we present a planner-independent linearise–validate cycle, where tasks can be solved by iterative refinement of a linear approximation of the domain, by validation against the full non-linear semantics.

1 Introduction

One of the largest problems in healthcare is the incorrect consumption of medication. It is estimated that half of patients that are prescribed medication for chronic conditions do not consume their medication correctly (The Academy of Medical Sciences 2014). Most medication is prescribed in a way that expects the patient to follow a standard routine. This is done in order to help the patient stay compliant and at the same time to consume the medication in a way that does not endanger the patient – often, when patients are given a regular dose, it is to keep things simple. For example, paracetamol (acetaminophen) is usually given in doses of 500mg per pill. The standard dose is two pills to be taken every four to six hours, with a maximum consumption of eight pills per day. Higher levels may give more pain relief, but the rate at which it is metabolised gives a risk of paracetamol toxicity if these limits are exceed – the spacing between doses, and daily limit, avoid excess exposure.

To address the challenge of effectively managing patients' medication usage, one option is to produce personalised medication plans. Personalised medicine is defined by the as providing "the right patient with the right drug at the right dose at the right time" (Sadee and Dai 2005). Historically, the scope for this has been limited to where it is essential (for instance, personalised insulin regimes for diabetics) but is recently becoming more viable through the uptake of technology – at one extreme, with the use of a drug dosage printer to 'print' drugs with accurately specific doses (Hirshfield et al. 2014).

In this paper we present the possibility of using PDDL+ to personalise medication schedules by modelling the problem as a hybrid planning domain, determining an effective schedule for a patient depending on their varying pain relief needs throughout the day. As the metabolism of medication is non-linear (negative exponential), and many otherwise-effective PDDL planners do not support non-linear domains, we explore the use of an iterative piece-wise linear approximation process to allow a broader range of planners to be used as a kernel within this process; and hence find solutions that are valid when considering the full non-linear pharmacokinetics. We present an initial evaluation of this approach using the planner OPTIC (Benton, Coles, and Coles 2012), as extended to support PDDL+ (Coles and Coles 2014), and discuss the future direction of the work and limitations of PDDL+ for modelling desirable objective functions in this domain.

2 Background

When consumed, medication is metabolised in the body over time, leading to a decay of the active medication level. Whilst pharmacokinetics are complex, a reasonable model is to assume negative-exponential (i.e. first-order) decay, with drug-dependent half-lives depending on the rate at which the active ingredients are metabolised (Geenen et al. 2013). Returning to the example of Paracetamol, the half life is up to 3 hours. That means if someone takes 1000mg of paracetamol at 12.00pm, there will be 500mg of the drug left in three hours (i.e. at 3.00pm). In another three hours (i.e. 6.00pm), there will be 250mg of the drug left, and so on. The question then, returning to the topic of this paper, is how these pharmacokinetics can be modelled in PDDL.

Medication levels changing over time are an example of *continuous numeric change*. The capability for these was first added to PDDL in PDDL2.1 (Fox and Long 2003), as part of 'layer 5' – durative actions can have continuous numeric effects that occur during their execution. As drug metabolism is not something that one can *choose* to occur in a plan, but is rather something that occurs exogenously in the world, a better fit is to encode it as a process, expressed in PDDL+ (Fox and Long 2006). PDDL+ provides a language for defining hybrid planning problems, where the state trajectory for a solution plan contains the effects of the planned actions (as in PDDL2.1) but also the consequences of ex-

ogenous processes and events. The key distinction is that a process occurs *whenever* its conditions are true: it stops and starts, outwith the direct control of the planned actions; and during its execution, it effects continuous numeric change upon the world. Analogously, PDDL+ events are instantaneous actions that occur *whenever* their conditions are true: when they fire, the world is immediately updated according to their instantaneously effects. By combining these, PDDL+ provides a useful toolkit for expressing hybrid domain models for a range of problems (Piotrowski et al. 2015). In this paper, as we will illustrate, we combine these to provide the necessary exogenous context in which to plan personalise medication consumption schedules.

3 High-level Problem Description

The decision-making constraints for a given medication can be defined using a number of constants:

- $t_{1/2}$, its biological half life. This is the time taken for the plasma level of the medication to halve.
- B, the typical dose amount consumed.
- G, the amount of time one needs to leave between two doses.
- m, the maximum number of doses within the planning horizon (e.g. 24 hours).

Given the half life, and the plasma level of a drug, D, the rate r at which the plasma concentration of the drug is decreasing can be calculated as:

$$r = D \cdot \frac{\ln 2}{t_{1/2}}$$

Taking the integral of this, the level of a drug at a time t ($D(t)$) after some reference point (D_0) can be written as:

$$D(t) = \frac{D_0}{2^{\frac{t}{t_{1/2}}}}$$

As we are planning the consumption of medication, we can think of the trajectory of drug levels during a solution plan as comprising time points at which medication is taken (when the drug level increases), interspersed with intervals in which the medication level reduces with rate r. More formally:

- At time t_0, the medication level takes some pre-defined value D_0 (the initial plasma level).
- At subsequent ordered time points $[t_1..t_{n-1}]$ an amount of medication $[B_1..B_{n-1}]$ is taken. For this paper, we assume these values are either 0 or B – the dose that could in principle be consumed.
- At time t_n, no medication is taken – this represents the end of a finite horizon over which the plan must succeed.

With this representation, we can define the constraints on medication consumption. First, if a dose is taken at time t_i, the next dose cannot be taken at a time before $t_i + G$:

$$\underset{i \in [1..n]}{\forall} \left(B_i = 0 \vee \left(\underset{j \in [i+1..n]}{\forall} (B_j = 0 \vee t_j - t_i \geq G) \right) \right)$$

Second, the maximum number of doses within the planning horizon is m:

$$|\{i \cdot i \in [1..n-1] \wedge B_i \neq 0\}| \leq m$$

A medication plan can be said to be safe if it satisfies these two constraints. A separate matter is what therapeutic benefit is provided: taking no medication at all never exceeds the limits of the prescription, but of course is of no benefit to the patient. The medication level $D(t_i)$ at time t_i ($i > 0$) can be defined as:

$$D(t_i) = \frac{D(t_{i-1}) + B_{i-1}}{2^{\frac{t_i - t_{i-1}}{t_{1/2}}}}$$

The interaction between planning decisions and plasma levels arises when considering how a plan may constrain what are permissible drug levels at different points within the planning horizon.

In this paper we will focus on pain relief management with a single painkiller as an exemplar problem, so can discuss drug levels in terms of desired levels of pain relief (*pr*).

In the simple case, the schedule of desired pain relief is static: at defined intervals throughout the day, the pain relief must be no lower than a minimum threshold. For instance, suppose a patient goes to work at 9am every morning and finishes work at 5pm every evening. We could then expect the minimum pain relief to be higher within this interval than at other times.

More persuasively, from a planning point of view, the desired pain relief is dynamic, depending on the actions used in the plan. For instance, during some actions (shopping for groceries, walking, and so on) a greater level of pain relief may be needed than at other times. A plan then must have a reasonable causal structure – as would be the case for planning a user's day modulo medication requirements – but additionally, the actions may have preconditions referring to pain relief that must hold during their execution. To meet these, in turn, requires the use of actions that correspond to taking medication. The resulting plan then gives the patients a personalised schedule of times at which they should take their medication, along with a plan for the day for their other activities, to ensure they get the right pain relief at the right time.

4 Modelling Pharmacokinetics in PDDL+

Having now set out the mathematical model of medication levels that we will use, we now map this to PDDL+.

First, to model the pharmacokinetics, a process is used. This runs whenever there is a non-zero amount of medication in the bloodstream; and decreases the medication level at rate r. In our example of single-medication pain relief, the variable pr represents pain relief, and ke represents the elimination constant for the medication:

$$ke = \frac{\ln 2}{t_{1/2}}$$

The process can then be written as follows:

```
(:process decay
 :parameters ()
 :precondition
   (> (pr) 0)
 :effect
   (decrease (pr) (* #t (* (pr) (ke))))
)
```

For clarity, the 'decrease' line can be read as:

$$-\frac{d\text{pr}}{dt} = \text{pr} \cdot ke$$

Alongside this process, there is one action that changes pr: ***consume***. This is a durative action with duration G and has the following preconditions and effects:

- To start the action, a proposition *safe-to-consume* (true initially) must be true; and a variable *doses* (0 initially) must be less than m (the maximum number of doses).
- When started, *safe-to-consume* is deleted; *doses* is increased by 1; and the pain relief level pr is increased by B – the dose of medication.
- At the end of the action, *safe-to-consume* is added

Effectively, *safe-to-consume* and *doses* perform the requisite book-keeping to enforce the constraints on maximum medication consumption. *safe-to-consume* acts as a semaphore: no two consume actions can overlap; and the duration G serves to ensure the minimum specified time between doses is thereby respected. *doses* is a simple counter, to ensure that if a dose is to be taken, the maximum safe limit for the period over which we are planning cannot be exceeded.

Having defined the pharmacokinetics, and constrained medication taking, what is left is to define the minimum pain relief. For the case of a static minimum pain relief schedule, the minimum pain relief level $minpr$ can be set using Timed Initial Fluents (Piacentini, Fox, and Long 2015) (TIF) – these specify the new value for $minpr$ at each time it needs to change. For instance, taking plan time units to be minutes counting from midnight:

```
(= (minpr) 0)
(at 420  (= (minpr) 100))
(at 540  (= (minpr) 200))
(at 1020 (= (minpr) 100))
(at 1320 (= (minpr) 0))
```

...sets the minimum pain relief to 100 at time 420 (7:00), to 200 at time 540 (9:00), then back down to 100 and 0 later in the day.

Having set the schedule, to ensure at all times $pr \geq minpr$, we use a PDDL+ event that fires at the first time this is not the case:

```
(:event prfailure
 :parameters ()
 :precondition (and
   (< (pr) (minpr))
   (min-check-passed)
 )
 :effect
   (not (min-check-passed))
)
```

The proposition *min-check-passed* is true in the initial state, required as a goal, and not added by any other action. Hence, if at any point in the plan the value of pr falls below $minpr$, the proposition is deleted, and a dead-end is reached: it is impossible to re-achieve this goal.

For dynamic pain relief levels, as set by actions, this is somewhat simpler: if an action requires some level of minimum pain relief during its execution, then this can be added as an `over all` condition, for instance:

```
(over all (>= (pr) 300))
```

...will ensure that the pain relief level is at least 300 during the execution of the action. The 'TIF plus event' model of a static pain relief schedule is unaffected, as this condition alone ensures that there is enough pain relief during the execution of the action.

The practical upshot of this PDDL+ encoding is that it allows a pharmacokinetic model to be specified as a background context into which a planning model can be specified.

5 Planning using a Linearise → Validate Cycle

In our initial experimentation with our PDDL+ model as specified thus far, a restricted range of planners were found that could reason with the negative-exponential numeric change induced by the process (e.g. UPMURPHI (Penna et al. 2009) and DINO (Piotrowski et al. 2015)); and some that could reason with processes, but only if the effects are linear (e.g. the extension of OPTIC described in (Coles and Coles 2014)). There is something of a trade-off between these two classes of planner. UPMURPHI *et al.* are capable of handling negative-exponentials and other non-linear domain features, and have been used in a number of applications including battery load management (Fox, Long, and Magazzeni 2012). Conversely, if linear change is sufficient, OPTIC's heuristic forward-search approach is a good choice; but, assuming pharmacokinetics are linear is not reasonable[1].

Desiring to maintain the benefits of using OPTIC, we present an iterative approach where a linear approximation is incrementally refined until the plans found are valid according to the non-linear domain model. This is related to the discretise–validate approach of UPMurphi, but instead of discretising time (notionally, on the X-axis), we linearise by segmenting the values of variables (on the Y-axis).

[1]Zero-order pharmacokinetics are rare; one notable exception is ethanol, but this has a narrow range of medical applications.

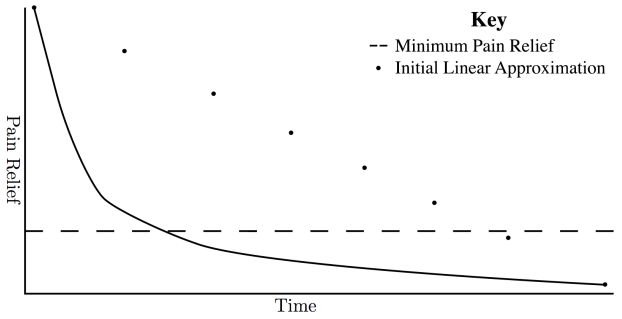


Figure 1: Initial Linear Approximation

As a starting point, we must devise an initial linear approximation. For this, we refer to:

- lb, the lower bound on what is an interesting medication level. This cannot be 0, as mathematically, a negative exponentially decreasing drug plasma level will only reach 0 as $t \to \infty$. We instead use a nominal value of 1% of a dose (for paracetamol, a value of 10, i.e 10mg).
- ub, the upper bound on what is an interesting medication level. This corresponds to taking the maximum number of doses m in succession, with each dose separated by the minimum time between doses, G.
- $\overline{r_{ub,lb}}$, the average medication decay rate over the time taken for the plasma level to fall from ub to lb:

$$\overline{r_{ub,lb}} = \frac{ub - lb}{\frac{-t_{1/2}}{\ln 2} \cdot \ln(\frac{lb}{ub})}$$

This is depicted in Figure 1. The solid line shows the negative-exponential change in pain relief, assuming the pain relief at time 0 is ub. The x-axis ranges from 0 until the time at which the pain relief level would reach lb. The dotted line shows a linear approximation spanning this time, with gradient $\overline{r_{ub,lb}}$.

A substantial caveat is that the initial linear approximation substantially over-estimates pain relief. If there is a minimum pain relief threshold (e.g. the dashed line in Figure 1), then actual pain relief will fall below the threshold far sooner than would be considered to be the case according to the linear approximation. But, we do get a simple linear process, with a constant (linear) effect:

```
(:process decay_ub_lb
 :parameters ()
 :precondition
   (and (>= (pr) lb) (<= (pr) ub))
 :effect
   (decrease (pr) (* #t r_ub_lb))
)
```

Solving this linearised problem, then validating the plan against the non-linear model using VAL (Howey, Long, and Fox 2004), will identify where the inaccuracies inherent in the linearisation have caused issues. This is evidenced in one of two ways:

Algorithm 1: Linearise–Validate Cycle

Data: P, the non-linear planning domain and problem;
$bounds = \{ub, lb\}$, the initial bounds
Result: A solution plan, Π

1 $P_{base} \leftarrow P$, with the decay process removed;
2 **while** *true* **do**
3 $sorted \leftarrow [bounds$, sorted in descending order$]$;
4 $P' \leftarrow P_{base}$, with a linear process for each successive pair in $sorted$;
5 $\Pi \leftarrow$ solve P' using planner;
6 **if** Π *is a valid solution to* P **then return** Π;
7 **foreach** $pr \in$ *VAL's diagnostic trace for* Π *using model* P **do**
8 $bounds \leftarrow bounds \cup \{pr\}$

- The event `prfailure` occurred, deleting `(min-check-passed)` – as this is a goal, the solution plan is invalid.
- A precondition on an action referring to pr was unsatisfied at some time – and hence the solution plan is invalid.

In both of these cases, VAL produces a diagnostic trace: a time-stamped progression through the plan, including what the value of pr was at each happening in the plan, as evaluated against the non-linear domain. With this information, we can refine the linearisation: instead of having a single-segment linear process spanning the whole range ub to lb, we can have several processes each covering one segment of this range.

Our motivation for refining the linearisation to give the right value of pr at happenings is based the observation that error in the linearisation is acceptable, so long it gives the right value when it matters; i.e. when `prfailure` would fire, or a precondition referring to pr would be violated. Hence, we ensure that on each iteration, the plan found with the previous linearisation will not be admitted by the new linearisation. This does not guarantee that a solution to the non-linear model will be found on the second iteration, but it does mean the model is iteratively refined to exclude apparently attractive but actually infeasible solutions.

Our approach is shown in Algorithm 1. We begin by running the planner to find a solution based on the initial approximation; i.e. starting with the initial values of ub and lb. Hence, at line 4, when the linearised model P' is generated, we have only one pair of bounds in the list, and generate a single process covering this range, with an effect with gradient $\overline{r_{ub,lb}}$. A solution Π to P' is then found.

As noted earlier, it is likely that when using the initial linearisation, Π will not be a solution to P – as P' overestimates the actual pain relief throughout the day. Hence, to refine the linearisation, we refer to the happenings in the diagnostic trace from VAL, an example of which can be seen in Figure 2, and keep all calculated values of pr (marked in boldface); i.e. for the plan Π, what values of pr were seen according to the model P. Each of these is added to the set of bounds (line 8). With this updated set of bounds, the loop starts again, generating an updated linear problem P',

```
Checking next happening (time 240)
Updating (pr) (1300) by 515.905 assignment

Checking next happening (time 240)
Adding (safe-to-consume)

Checking next happening (time 486.079)
Updating (pr) (515.905) by 200 assignment

EVENT triggered at (time 486.079)
Triggered event (prfailure)
Deleting (min-check-passed)

Checking next happening (time 544.824)
Updating (pr) (200) by 159.509 assignment

...

Checking next happening (time 784.824)
Updating (pr) (1159.51) by 460.152 assignment
```

Figure 2: Example Stack Trace from VAL

and once again attempting to find a solution plan. For the updated problem, the bounds are sorted, and a process generated for each adjacent pair of bounds, each with its own $\overline{r_{ub,lb}}$ value.

6 Evaluation

As an initial evaluation of our approach, we generated a sequence of problems with increasing planning horizon and a fixed $minpr$ level, thereby necessitating increased number of doses as problem sizes increases. The planning horizons tested started at 540 minutes (i.e. 9 hours), and increased by 60 minutes each time.

The reference drug used was paracetamol, with the following initial parameters:

- $t_{1/2}$ (half life): 180 minutes
- ***B*** (dose amount): 1000mg
- ***G*** (gap between doses): 240 minutes
- ***m*** (max doses): 4 doses
- ***lb***, (lower bound): 10mg
- ***ub***, (upper bound): 3000mg
- ***minpr***: 200mg.

Our linearise–validate cycle was implemented in a planner-independent way, but the only candidate planner that yielded solutions was the extension of OPTIC to support linear PDDL+ (Coles and Coles 2014). A number of other planners were considered (Cashmore et al. 2016; Piotrowski et al. 2015; Penna et al. 2009), but publicly available implementations of planners were unable to solve problems (non-linear and linear). The results for this configuration are shown in Table 1. All problems were solved in at most two iterations, with the time for the first iteration shown in the row $t_{initial}$, and for the refined iteration in the row $t_{refined}$.

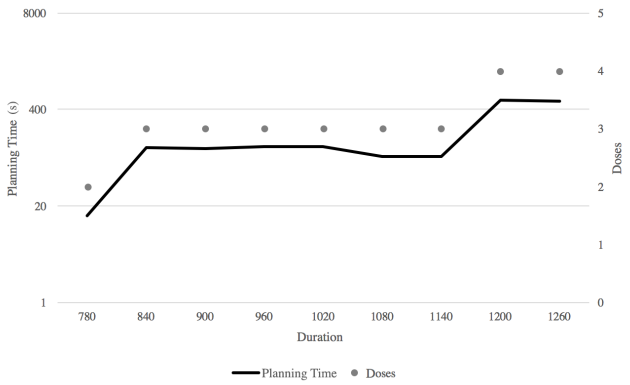


Figure 3: Comparison Between Doses Needed and Planning Time (in seconds)

For shorter planning horizons, we see some interesting results.

- For a horizon of 540 minutes, the solution for the initial linearisation used only a single dose of medication; the resulting solution did not validate, so the linear model was refined, and a new plan found.
- For horizons in the range 600–720, the solution for the initial linearisation recognised the need to take two doses of medication; and these were taken back-to-back. The resulting plan was incidentally a valid solution for the non-linear model.
- For a horizon of 780, two doses were needed (as with 600–720) but the timing of the doses is more important: the second dose needs to be delayed to come somewhat more than four hours after the first. This was not recognised by the initial linearisation (which over-estimated pain relief at time 780); but was compensated for in the refined linearisation by appropriately delaying the second dose.

For the larger problems, the initial linearisation was never valid. The planning time for the refined model was strongly correlated with the number of doses to be taken – this is shown more clearly in Figure 3, where the planning time (left Y-axis) tracks the number of doses needed (right Y-axis).

To gain further insights into the linearisation process, we looked at the solution plan found for the longest planning horizon (1260), after the first and second iteration. These plans were validated against three models:

- The refined linearisation
- The non-linear model (i.e. exponential decay)
- The initial linearisation

Figure 4 shows the calculated value of pr for the plan after the first iteration, validated against each of these. The first plan took three doses back-to-back. With reference to the initial linearisation (the dashed line), this is reasonable – a fourth dose was not necessary to complete the plan. As can be seen, the initial approximation is as expected extremely

	Planning Horizon (minutes)												
	540	*600*	*660*	*720*	*780*	*840*	*900*	*960*	*1020*	*1080*	*1140*	*1200*	*1260*
$\mathbf{t_{initial}}$	0.05	0.03	0.03	0.04	0.03	0.04	0.03	0.03	0.06	0.06	0.06	0.09	0.09
$\mathbf{t_{refined}}$	0.71	-	-	-	11.72	164.19	162.99	161.29	160.09	110.05	111.2	654.77	649.86
total	0.76	0.03	0.03	0.04	11.75	164.23	163.04	161.32	160.12	110.11	111.26	654.83	649.95
doses	2	2	2	2	2	3	3	3	3	3	3	4	4

Table 1: Planning times (seconds) for fixed $minpr$, increasing plan horizon

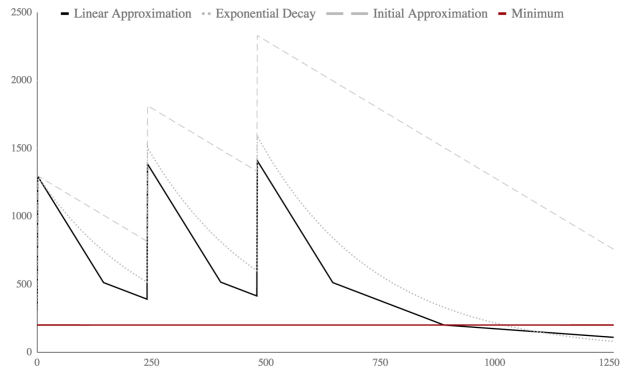


Figure 4: Pain Relief Levels with Three Models, using the Initial Approximation Plan

Figure 5: Pain Relief Levels with Three Models, using the Refined Approximation Plan

ub	3000.0	515.905	515.901	200.0
lb	515.905	515.901	200.0	10.0
$\overline{r_{ub,lb}}$	5.43373	1.986648	1.283748	0.2442325

Table 2: Example bounds after one iteration (horizon 1260)

optimistic, with the calculated value of pr significantly exceeding the actual negative-exponential value (the dotted line). The refined linearisation (solid line) avoids falling into the same trap: a three-dose solution would cross the $minpr$ threshold, so would never be returned as a solution by the planner.

An analogous graph for the plan after the second iteration, found by the planner using the refined linearisation, is shown in Figure 5. Crucially, a fourth dose is now taken. In particular, the planner scheduled doses to be taken at the earliest possible instance for the first three doses (at time 0, 240 and 480) and waited until the latest possible time to take the fourth dose; i.e. medication was consumed just before the $minpr$ threshold was crossed, the point at which the `prfailure` event would otherwise have fired.

For reference, with a horizon of 1260, five bounds were used: the initial bounds of 3000 and 10, and a further three in between. This yielded four linear processes, whose parameters are shown in Table 2.

To test whether the planner could scale over a horizon beyond 1260 minutes, we created a problem with a horizon of 1500 minutes (25 hours), using a Timed Initial Literal (Hoffmann and Edelkamp 2005) to mark the point at which the day changed (to limit doses consumed per 24 hours). Although the planner was able to find a solution, needing five doses, it took almost two hours – adding the Timed Initial Literal to switch from one day to the next had a substantial effect on the size of the search space.

7 Discussion and Future Work

With our our linearise–validate approach, we have shown we can solve problems in this domain, and have presented an evaluation to show its efficacy on fixed-minimum-pain-relief tasks. The evaluation problem as it stands could be seen as a scheduling problem rather than a planning problem, but our motivation for doing this within PDDL+ is to allow task and activity planning to take place in the context of medication scheduling.

We will now briefly discuss some of our future research directions.

7.1 Polymedicine

Our evaluation here considered only a single drug; the next step is to look at polypharmacy. As almost a quarter of the UK population are on at least three prescriptions (Scholes, Faulding, and Mindell 2014), this is a substantial area of interest.

In the case of painkillers, these are often complementary. If a patient was only on one drug, it may be difficult to give adequate pain relief due to the constraints of the drug itself. For example, paracetamol is an effective painkiller, but the dosage restrictions mean it is a challenge to use it as a monotherapy to give sufficient pain relief for a patient's routine. Thus, if we take into account multiple painkillers (for

example paracetamol and ibuprofen), a combined schedule of the two gives greater potential for pain management (taking both at the same time), and greater flexibility (taking them at different times). The modifications to the model to support this are relatively straightforward: rather than using a single pr variable, use multiple such variables (one for each painkiller) and define conditions on pain relief to refer to a weighted sum of these.

A more challenging case is where there are interactions between medication. Ideally, two adversely interacting medications would not be taken concomitantly, but it is sometimes unavoidable. To handle pharmacokinetic interactions (one drug affects the rate of metabolism of another), the drug decay processes would need to be updated. How to do this well remains an open challenge.

7.2 Planning to avoid side effects

As discussed earlier, activity-specific drug plasma level requirements can be incorporated into the preconditions of actions. This provides a mechanism for allocating doses of medication around a patient's daily routine.

A further consideration is the side-effects of medication, as well as their desired effects. These side-effects place various constraints on how medication should be taken. For instance, some medication require activities to take place before or after consumption. For example:

- Ibuprofen cannot be taken on an empty stomach, or it will cause irritation, so an ideal plan would include meal-times as well as medication times.
- Codeine causes drowsiness, so patients should avoid taking it before driving or operating machinery.
- Paracetamol conversely has no such constraints, but the daily maximum dose is quite limited.

Considering a full plan of action for the day for a patient, covering a wide range of their daily activities, there is good scope to improve the management of their medication to reduce adverse effects.

7.3 Plan quality metrics

Thus far, our discussion has been on finding plans that meet hard constraints, in terms of drug plasma levels during times of the day, or during activities. We could hope to improve a patient's quality of life further by finding plans that are of good quality.

The question then is what the quality metric should be. In the context of pain relief, whilst a minimum pain relief may be specified, the patient may for the sake of comfort wish to avoid their pain relief getting quite down to this minimum. A good candidate for a plan quality metric is to maximise the minimum gap between pr and $minpr$ seen during the plan. With reference to Figure 5, this would have the effect of delaying the second/third doses to avoid pr going quite so low before the fourth dose was taken: the same medication is taken, but the plan is subjectively better.

The caveat here though is that whilst 'max of min' or 'min of max' metrics are common in various scheduling problems (e.g. Job-Shop Scheduling) they are non-standard in planning, and cannot be elegantly expressed in PDDL+. As OPTIC is already using a MIP to check that the preconditions on solution plans hold, it already has the framework to use more powerful plan metrics, by setting these as the MIP objective function; we will be exploring this in our future work.

8 Conclusions

In this paper, we presented a PDDL+ model of drug pharmacokinetics, to provide a context in which to find solution plans that consider, *inter alia*, the consumption of medication. As heuristic forward-search planners do well in terms of causal reasoning, but often handle only linear dynamics, we devised a linearise–validate approach for solving these problems, by iteratively refining a linear approximation of the domain using the diagnostic trace returned by the plan validator, VAL.

An initial evaluation, implemented as a wrapper around the planner OPTIC, demonstrates the feasibility of this approach. The results are exciting, opening up the opportunity for future work both in terms of more comprehensive model development, and more generally improving the range of quality metrics that can be handled by PDDL-based planners in hybrid domains.

References

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

Cashmore, M.; Fox, M.; Long, D.; and Magazzeni, D. 2016. A Compilation of the Full PDDL+ Language into SMT. In *Proceedings of the Twenty-Sixth International Conference on International Conference on Automated Planning and Scheduling (ICAPS)*.

Coles, A. J., and Coles, A. I. 2014. PDDL+ Planning with Events and Linear Processes. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS)*.

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

Fox, M., and Long, D. 2006. Modelling Mixed Discrete-Continuous Domains for Planning. *Journal of Artificial Intelligence Research*.

Fox, M.; Long, D.; and Magazzeni, D. 2012. Plan-based policies for efficient multiple battery load management. *Journal of Artificial Intelligence Research*.

Geenen, S.; Yates, J. W. T.; Kenna, J. G.; Bois, F. Y.; Wilson, I. D.; and Westerhoff, H. V. 2013. Multiscale modelling approach combining a kinetic model of glutathione metabolism with pbpk models of paracetamol and the potential glutathione-depletion biomarkers ophthalmic acid and 5-oxoproline in humans and rats. *Integrative Biology* 5:877–888.

Hirshfield, L.; Giridhar, A.; Taylor, L.; Harris, M.; and Reklaitis, G. 2014. Dropwise additive manufacturing of pharmaceutical products for solvent-based dosage forms. *Journal of Pharmaceutical Sciences* 103:496–506.

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

Howey, R.; Long, D.; and Fox, M. 2004. VAL: Automatic Plan Validation, Continuous Effects and Mixed Initiative Planning using PDDL. In *The 16th IEEE International Conference on Tools with Artificial Intelligence*, 294–301.

Penna, G. D.; Magazzeni, D.; Mercorio, F.; and Intrigila, B. 2009. UPMurphi: A Tool for Universal Planning on PDDL+ Problems. In *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS)*.

Piacentini, C.; Fox, M.; and Long, D. 2015. Planning with numeric timed initial fluents. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI)*.

Piotrowski, W.; Fox, M.; Long, D.; Magazzeni, D.; and Mercorio, F. 2015. Heuristic Planning for Hybrid Systems. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI)*.

Sadee, W., and Dai, Z. 2005. Pharmacogenetics/genomics and personalized medicine. *Human Molecular Genetics*.

Scholes, S.; Faulding, S.; and Mindell, J. 2014. Use of prescribed medicines. In *Health Survey for England – 2013*. chapter 10.

The Academy of Medical Sciences. 2014. Patient Adherence to Medicines.

Using Squeaky Wheel Optimization to Derive Problem Specific Control Information for a One Shot Scheduler for a Planetary Rover

Wayne Chi, Steve Chien, Jagriti Agrawal
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109
{firstname.lastname}@jpl.nasa.gov

Abstract

We describe the application of using Monte Carlo simulation to optimize a schedule for execution and rescheduling robustness and activity score in the face of execution uncertainties. We apply these techniques to the problem of optimizing a schedule for a planetary rover with very limited onboard computation. We search in the schedule activity priority space - where the onboard scheduler is (a) a one shot non-backtracking scheduler in which (b) the activity priority determines the order in which activities are considered for placement in the schedule and (c) once an activity is placed it is never moved or deleted. We show that simulation driven search outperforms a number of alternative proposed heuristic static priority assignment schemes. Our approach can be viewed using simulation feedback to determine problem specific heuristics much like squeaky wheel optimization.

Introduction

Embedded schedulers must often perform within very limited computational resources. We describe an approach to automatically deriving problem specific control knowledge for a one-shot (non-backtracking) scheduler intended for a planetary rover with very limited computing. In this application, the onboard scheduler is intended to make the rover more robust to run-time variations (e.g., execution durations) by rescheduling. Because the general structure of the schedule is known a priori on the ground before uplink, we use both analysis of the schedule dependencies and simulation feedback to derive problem specific control knowledge to improve the onboard scheduler performance.

The target onboard scheduler is a one-shot limited search scheduler. Because the scheduler does not backtrack across activity placements, the order in which it considers activities heavily influences generated schedule quality. In our approach, we search the space of activity priorities which determine the order in which the scheduler considers activity placement. At each step in the priority search, a Monte Carlo simulation is conducted to assess the likelihood of an activity being executed. Using an approach analogous to squeaky wheel optimization, these runs are automatically analyzed and used to feed back into adjustments to the activity priorities (and hence the order in which they are considered for inclusion in the schedule for both initial schedule generation and rescheduling). This search in the activity priority space continues until all requested activities are included or a resource bound is exceeded. We call this method *Priority Search* and we present empirical results that show that *Priority Search* outperforms several static priority assignment methods (those that do not use Monte Carlo feedback) *including manual expert derived priority setting*.

We study this problem in the context of setting activity priorities as part of the ground operations process for a one-shot, non-backtracking scheduler (Rabideau and Benowitz 2017) designed to run onboard NASA's next planetary rover, the Mars 2020 (M2020) rover (Jet Propulsion Laboratory 2017a). For our problem, the onboard scheduler is treated as a predetermined "black box".

The remainder of the paper is organized as follows. First we describe our formulation of the scheduling problem, metrics for schedule goodness, and the onboard scheduling algorithm. Second, we describe several static approaches to priority assignment as well as our *priority search* approach that leverages Monte Carlo simulation feedback. Third, we describe empirical results demonstrating the efficacy of *priority search* over static methods, evaluating on *sol types*, the best available anticipated operations plans for the M2020 planetary rover mission. Finally, we describe related and future work and conclusions.

Problem Definition

For our defined scheduling problem (Rabideau and Benowitz 2017), the scheduler is given

- a list of activities
  $A_i\langle p, R, e, dv, \Gamma, T, D\rangle \ldots A_n\langle p, R, e, dv, \Gamma, T, D\rangle$
- where p is the scheduling priority of the activity, and
- R is the set of unit resources $R_1 \ldots R_m$ that the activity will use (up to project limitations - 128 for M2020), and
- e and dv are the rate at which the consumable resources energy and data volume respectively are consumed by the activity, and
- Γ are non-depletable resources used such as sequence engines available or peak power, and

- T is a set of the activity's optional a) start time windows $T_{i\_start} \dots T_{i\_end}$ and b) preferred schedule time $T_{i\_preferred}$, and
- D is a set of the activity's dependency constraints from $A_j \rightarrow A_k$ [1]

All activities are *Mandatory Activities*. These are activities, $m_1 \dots m_j \subseteq A$, that must be scheduled as long as the given set of inputs are *valid*. In order for a set of inputs to be considered *valid*, there must exist a valid (e.g. constraint satisfying) schedule - in the context of the scheduler - that includes all of the mandatory activities. Note that the M2020 Onboard Scheduler is an incomplete algorithm. As a result, there could be a set of inputs where valid schedule exists and a complete scheduler would place all mandatory activities, but the Onboard scheduler would not. Since not all input sets will be *valid*, it is important for us to modify the input sets (e.g. changing priorities) to allow all mandatory activities to be scheduled.

In addition, activities can be grouped into *Switch Groups*. A *Switch Group* is a set of activities where exactly one of the activities in the set must be scheduled. The activities within a switch group are called *switch cases* and vary only by how many resources (time, energy, and data volume) they consume. Switch groups allow us to schedule a more resource-consuming activity if it will fit in the schedule. For example, one of the M2020 instruments takes images to fill mosaics which can vary in size; for instance we might consider $1x5$, $3x5$, or $5x5$ mosaics. Taking larger mosaics might be preferable, but taking a larger mosaic takes more time, takes more energy, and produces more data volume. These alternatives would be modeled by a switch group that might be as follows:

$$SwitchGroup = \begin{cases} Mosaic_{1x5} & \text{Duration=100 sec} \\ Mosaic_{3x5} & \text{Duration=200 sec} \\ Mosaic_{5x5} & \text{Duration=400 sec} \end{cases} \quad (1)$$

In the above example, the scheduling priority order would be $Mosaic_{1x5}$ the lowest of the three, then $Mosaic_{3x5}$, and $Mosaic_{5x5}$ the highest. The desire is for the scheduler to schedule the activity $Mosaic_{5x5}$ but if it does not fit then try scheduling $Mosaic_{3x5}$, and eventually try $Mosaic_{1x5}$ if the other two fail to schedule. The challenge for the scheduler is that getting a preferred switch case is not deemed worth forcing out another mandatory activity from the schedule. Because the normal approach to handling such interactions is to search, this introduces complications into the scheduling algorithms but these are the subject of a different paper.

The charter of the scheduler is to produce a grounded time schedule that satisfies all of the above constraints.

We also make the following assumptions:

1. There exists a set of activity scheduling priorities that would allow all mandatory activities to be scheduled by the scheduler [2].
2. The prior schedule is executed while the scheduler is running (Chi et al. 2018).
3. Activities do not fail.
4. No preemption (activities are only preempted as a major failure case for M2020).
5. The onboard scheduler is a "black box" - the onboard scheduler algorithm (Algorithm 1) is fixed.

The goal of the scheduler is to schedule all mandatory activities and better switch cases [3] while respecting individual and plan-wide constraints.

The goal of the priority setting algorithm is to derive a set of priorities that will best allow the scheduler to achieve that goal. Not only that, but we must derive that set of priorities in the shortest amount of time possible in order to satisfy daily mission time constraints.

Scheduler Design

Algorithm 1 Onboard Scheduler

Input:
$A\langle p, R, e, dv, \Gamma, T, D\rangle$: List of activities with their individual constraints
C: Constraints for the whole plan (e.g. available cumulative resources)
S: Current state of the spacecraft (state of charge, data volume, activity status)
Output:
U: Resulting schedule

Sort(A) ▷ Sorted by highest to lowest priority.
for each $a \in A$ **do**
  $P \leftarrow \emptyset$ ▷ Some activities may require automatically generated preheats
  $M \leftarrow \emptyset$ ▷ Some activities may require automatically generated maintenances
  $I \leftarrow [a.earliest\_start\_time, a.latest\_start\_time] \bigcap find\_valid\_intervals(a.unit\_resources) \bigcap find\_valid\_intervals(a.activity\_status) \bigcap find\_valid\_intervals(a.data\_volume)$
  if $requires\_preheat(a)$ **then**
    $P \leftarrow generate\_preheat\_activities(a)$
    $M \leftarrow generate\_maintenance\_activities(a)$
  end if
  $I \leftarrow I \bigcap find\_valid\_intervals(a.energy, P, M) \bigcap find\_valid\_intervals(a.peak\_power, P, M)$
  $awake \leftarrow generate\_awake\_activity(a, I)$
  if $I \neq \emptyset$ **then**
    $schedule\_activity(a, I)$
    $schedule\_activity(awake, I)$
    for each $p \in P$ **do**
      $schedule\_activity(p, I)$
    end for
    for each $m \in M$ **do**
      $schedule\_activity(m, I)$
    end for
  end if
end for

The Mars 2020 onboard scheduler (Algorithm 1) is a single shot, non-backtracking scheduler that schedules (consid-

[1] $A_j \rightarrow A_k$ means the scheduled end time of A_k must be before the scheduled start time of A_j.

[2] Since our algorithm includes an incomplete scheduler, our assumption of a valid set of inputs can only hold true for our particular scheduler

[3] See Evaluating a Schedule for more information

ers activities) priority first order and never removes or moves an activity after it is placed during a single scheduler run. It does not search except when considering valid intervals for a single activity placement and when scheduling sleep and preheats [4] (Rabideau and Benowitz 2017).

Due to the greedy, non-backtracking nature of the onboard scheduler, the order in which activities are scheduled can greatly impact the quality of the schedule.

Evaluating a Schedule

In order to evaluate the goodness of a particular priority assignment, we have developed a scoring method based on how many and what type of mandatory and switch group activities are able to be scheduled successfully by the scheduler. The score is such that the value of any single mandatory activity being scheduled is much greater than that of any combination of switch cases (at most one activity from each switch group can be scheduled). This ensures the following strict ordering:

$$V(m \in M) \gg \sum_{i=1}^{n_S} V(s \in S_i) \tag{2}$$

where $V(x)$ is the value of activity x being scheduled, M is the set of all mandatory activities, n_S is the number of switch groups, S_i is switch group i, and s is a switch case in switch group S_i.

Static Algorithms for Activity Priority Assignment

We have developed several static algorithms which set the priorities of activities based on various activity ordering criteria. These algorithms do not consider Monte Carlo simulations of plan execution where activities may end early or late while determining priorities, unlike our Priority Search approach. We will later compare these to our Priority Search approach to gain a better understanding of how well it performs. Activities which must begin at a particular time (e.g. data downlink) are always given the highest priority and thus are not affected by the static algorithms described.

The following four methods are used to initialize activity priorities:

- *Equal Priorities.* All activities have equal priorities.
- *Random Assignment.* Each activity is given a random priority.
- *Latest Start Time.* The activity priorities are ordered by the latest time they are allowed to start. The activity with the earliest such time has the highest priority.
- *Human Expert.* Each activity is assigned a priority based on the start time of the activity in a schedule constructed by a human expert. The activity with the earliest start time in this schedule has the highest priority.

The following two methods are applied to the priorities after they have been initialized in one of the four ways described above:

[4] Sleep and preheats are activities automatically generated and scheduled by the scheduler.

- *Dependencies.* $A \rightarrow B$ means that B must execute successfully before A can start. To generate a schedule that respects this,

$$A \rightarrow B \Rightarrow priority_A < priority_B \tag{3}$$

where higher priority means an activity is considered for scheduling earlier.

- *Tie Breaker.* If activities have the same priority assignment the activity with earliest latest allowed start time is of higher priority. If they also have the same latest allowed start time then the longer activity has the higher priority. If all of these attributes are equal then the higher priority activity is chosen lexicographically based on each activity's unique identifier.

Priority Search

In order to determine a set of priorities which will allow the scheduler to generate a schedule better than our static heuristics, we attempt to search the priority space in an approach similar to Squeaky Wheel Optimization (SWO) as described in Joslin and Clements 1999 (Joslin and Clements 1999). Squeaky Wheel Optimization usually involves a constructor, an analyzer, and a prioritizer. The constructor generates a schedule, the analyzer determines problem areas and assigns "blame" to certain elements in the schedule, and the prioritizer modifies the order in which the elements are considered. This process repeats until a satisfactory result is reached or allotted time runs out. However, our scheduling problem is intrinsically tied to execution and analyzing the initial schedule generated by itself is not satisfactory. Our approach (Figure 1) builds upon the usual SWO approach by incorporating a simulation of execution and Monte Carlo to build an execution sensitive result. We call our approach Priority Search as it searches the priority space using Monte Carlo simulation feedback to find a good set of priorities, unlike the static algorithms.

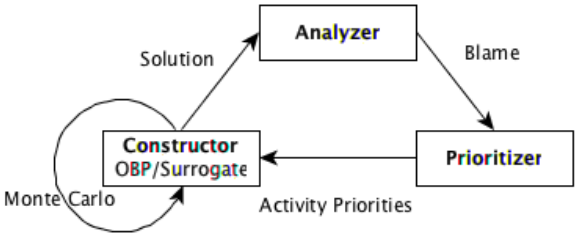


Figure 1: Squeaky Wheel accounting for Execution

Constructor

Typically, the constructor generates a schedule as the solution, which is then fed into the analyzer. However, our scheduling problem must be taken in context with execution. Activities may finish early or late which affect how many and which activities can be scheduled. In order to take this into account, we generate the final schedule of a

(lightweight) simulation of the entire plan execution. This is simulated by letting activities finish early or late by a variable amount based on a probabilistic model of plan execution [5]. However, the probabilistic model may promote misleading results if only sampled once. As a result, our constructor (Algorithm 2) runs a Monte Carlo and simulates multiple plan executions, passing on all of the executed plans as the solution to the analyzer.

Algorithm 2 Monte Carlo Constructor

Input:
$A\langle p, R, e, dv, \Gamma, T, D\rangle$: List of activities with their individual constraints
C: Constraints for the whole plan (e.g. available cumulative resources)
N: Number of runs in the Monte Carlo
Output:
S: List of all final schedules after simulating execution
1: $i \leftarrow 0$
2: **while** $i < N$ **do**
3: $schedule \leftarrow$ simulation(A, C)[6]
4: $S_i \leftarrow schedule$
5: $i \leftarrow i + 1$
6: **end while**

Priority Analyzer

The analyzer (Algorithm 3) takes the solution and assigns blame to problem areas. Since our objective is to schedule all mandatory activities and better switch cases, we blame all activities that are not scheduled. Since the solution is multiple schedules, there may be some Monte Carlo runs where the activities do not succeed or fail to be scheduled. For simplicity, we chose to blame any activity that was unscheduled in any of the schedules, but other approaches may assign blame according to how many times an activity was not scheduled.

Algorithm 3 Monte Carlo Analyzer

Input:
$A\langle p\rangle$: List of activities with priorities
S: List of all final schedules after simulating execution
Output:
U: List of all unscheduled activities
$score$: Score (objective function)
1: **for each** $S_i \in S$ **do**
2: $U \leftarrow U \bigcup \{\forall a \in A | a \notin S_i\}$
3: $score \leftarrow score +$ get_score(S_i)
4: **end for**

Constant Step Prioritizer

A simple way to re-prioritize is to increase the blamed (unscheduled) activities' priorities by a constant step size s.

Typically, activities have varying degrees of flexibility due to their constraints (resources, dependencies, time, etc.).

[5] See Empirical Results for how that probabilistic model was generated.

[6] The final schedule after simulating execution.

Algorithm 4 Constant Step Reprioritization

Input:
$A\langle p\rangle$: List of activities with priorities
U: List of all unscheduled activities (from analyzer)
$step$: Constant step size
Output:
A: Best relative ordering of activities found
1: **for each** $a \in U$ **do**
2: incrementRelativePriority(a, $step$, A)
3: **for each** $d \in a.dependents$ **do**
4: incrementRelativePriority(d, $step$, A)
5: **end for**
6: **for each** $sg \in a.switchGroup$ **do**
7: incrementRelativePriority(sg, $step$, A)
8: **end for**
9: **end for**

Higher priority activities can consume resources (unit resources, energy, and data volume) or change state in a way that prevents lower priority activities from scheduling such that their constraints are satisfied. Increasing the blamed activities' priorities allows them to schedule earlier (scheduling order) which means they have more "slack" to satisfy their constraints. The goal is that the algorithm will slowly promote less flexible activities to the top so that their constraints can be satisfied, and demoted activities are flexible enough to be scheduled in a more constrained plan.

When increasing the relative priorities of blamed activities, the existing relative priorities between certain groups of activities must remain enforced.

First, each switch group must maintain the relative priorities between each activity in the grouping. For each switch group, the activities $(s_1, \ldots, s_n)$ must be ordered such that those with higher resource consumption (time, energy, and data volume) have higher priorities as well.

Second, dependency relationships must be enforced such that (3) is held true.

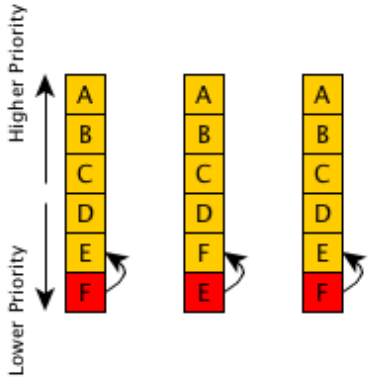


Figure 2: Cycle in the Constant Step approach. Red activities were unable to be scheduled and assigned blamed.

There is one main issue with the Constant Step approach

- it is extremely susceptible to cycles. One common cause for cycles is that a set of activities needs to be promoted beyond a particular activity together, but the constant step size prevents this from ever occurring. For example, in Figure 2 activity F is unschedulable and assigned blame. Its priority is increased, but this causes activity E to fail to schedule. Activity E is then promoted in the next iteration, causing F to fail to schedule and the process repeats. In reality, both E and F have to be promoted above D, but because the step size is constant, they will never achieve that and form a cycle. The situation where activities are unable to be promoted above an activity blocking it can be extended to any constant step size less than the maximum step size [7].

Stochastic Step Reprioritization

Algorithm 5 Stochastic Step Reprioritization

Input:
$A\langle p\rangle$: List of activities with priorities
U: List of all unscheduled activities (from analyzer)
Output:
A: Best relative ordering of activities found

```
step ← random(1, A.length)
for each a ∈ U do
    incrementRelativePriority(a, step, A)
    for each d ∈ a.dependents do
        incrementRelativePriority(d, step, A)
    end for
    for each sg ∈ a.switchGroup do
        incrementRelativePriority(sg, step, A)
    end for
end for
```

Injecting randomness to the step size allows the algorithm to become robust to cycles. In each iteration of the priority setting algorithm, a random step distance between 1 and N, where N is the number of activities in the plan, is assigned to all of the blamed activities. This lets the scheduler always have the possibility of being promoted above a resource constraining activity, while still allowing smaller step size priority permutations.

The main issue that lies with a random approach is that empirically [8] it finds the global maximum score slower than desired. This is further exacerbated by the fact that each iteration of our SWO cycle takes a non-negligible amount of time (a few seconds) due to the need to run a lightweight simulation and Monte Carlo.

Max Step Reprioritization

Stochastic Step Reprioritization (empirically) produced results slower than desired. Max Step Reprioritization seeks to solve both of those issues by always promoting blamed activities to have the highest scheduling priorities. The earlier an activity is considered for scheduling, the more flexibility that activity has to be scheduled. Therefore, if an activity is first to be considered for scheduling, but still cannot be successfully scheduled, there is no other scheduling priority that would allow the activity to be scheduled. Knowing this, by promoting blamed activities to have the highest scheduling priorities we can attempt to avoid iterations that fail to schedule the same blamed activities, thereby speeding up the overall algorithm.

Since the blamed activities will have the highest scheduling priorities, cycles such as those seen in Figure 2 can be avoided. However, Max Step Reprioritization doesn't prevent cycles entirely and they still pose an issue when encountered.

Algorithm 6 Max Step Reprioritization

Input:
$A\langle p\rangle$: List of activities with priorities
U: List of all unscheduled activities (from analyzer)
Output:
A: Best relative ordering of activities found

```
for each a ∈ U do
    step ← A.length
    incrementRelativePriority(a, step, A)
    for each d ∈ a.dependents do
        incrementRelativePriority(d, step, A)
    end for
    for each sg ∈ a.switchGroup do
        incrementRelativePriority(sg, step, A)
    end for
end for
```

Empirical Evaluation

In order to evaluate how well our Priority Search algorithm is able to generate a priority assignment which results in an optimal schedule, we have applied the algorithm to various sets of inputs comprised of activities with their constraints and priorities and compared against various static algorithms. The inputs are derived from *sol types*. *Sol types* are currently the best available data on expected Mars 2020 rover operations (Jet Propulsion Laboratory 2017a). In order to construct a schedule and simulate plan execution, we use the *M2020 surrogate scheduler* - an implementation of the same algorithm as the M2020 onboard scheduler (Rabideau and Benowitz 2017), but implemented for a Linux workstation environment. As such, it is expected to produce the same schedules as the operational scheduler but runs much faster in a workstation environment. The surrogate scheduler is expected to assist in validating the flight scheduler implementation and also in ground operations for the mission (Chi et al. 2018).

Each input file contains approximately 40 activities. We use a probabilistic execution model based on operations data from the Mars Science Laboratory Mission (Jet Propulsion Laboratory 2017b; Gaines et al. 2016a; 2016b) in order to simulate activities completing early by a reasonable amount. In our model to determine activity execution durations, each of the actual execution durations provided in MSL data is first divided by the corresponding predicted execution dura-

[7] See section Max Step Reprioritization
[8] More information can be found in Empirical Evaluation.

tion. Then, we use a linear regression on the scaled values to obtain a mean and standard deviation presuming the ratio of predicted to actual execution times is normally distributed. The value representing the actual execution duration on the regression line for the given conservative duration is used as the mean. A scaled prediction of the actual duration is generated from a a normal distribution using the derived mean and standard deviation. Finally, this value is scaled back by multiplying by the given conservative duration. Note that we do not explicitly change other activity resources such as energy and data volume since they are generally modeled as rates and changing activity durations implicitly changes energy and data volume as well.

Using each of the sol types, we create variants by adding two switch groups to a set of inputs. Each switch group contains three switch cases where the switch cases differ in duration in a manner similar to the one described in (1). Each of the two switch groups are as follows:

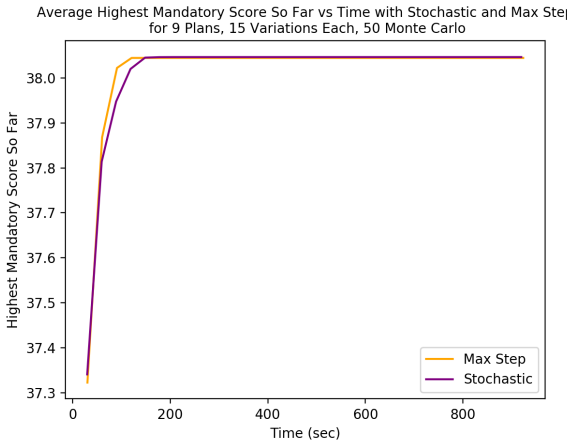
$$SwitchGroup = \begin{cases} Activity_{original} & \text{Duration=}x \text{ sec} \\ Activity_{2x} & \text{Duration=}2x \text{ sec} \\ Activity_{4x} & \text{Duration=}4x \text{ sec} \end{cases} \tag{4}$$

Due to the inequality in (2), a successfully scheduled mandatory activity is of much higher value than a successfully scheduled longer switch case. Therefore, the mandatory activity score is weighted at a much larger value then the switch group score. Each mandatory activity that is successfully scheduled is given one point which contributes to the mandatory score. If a switch case with a duration that is 2 times that of the original activity is able to be scheduled, then it contributes $1/5$ to the switch group score. If a switch case that is 4 times the original duration is able to be scheduled, then it contributes $2/5$ to the switch group. Since there are two switch groups in each variant, the maximum switch group score for a variant is $2 * (2/5) = 4/5$. In the following empirical results, we average the mandatory and switch groups scores over all Monte Carlo runs of execution.

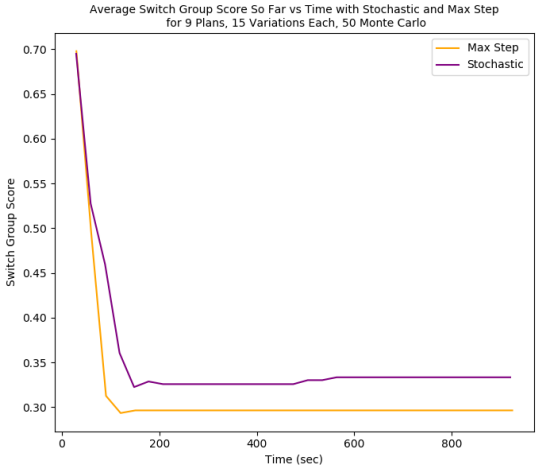
Also, in each of our variants we set the preferred schedule time of each activity to the earliest time the activity is allowed to start.

We first compare the different approaches to implementing Priority Search to understand which performs better.

The highest score so far is a combination of the mandatory score and the switch group score where the mandatory score is weighted at a much higher value than the switch group score. In Figure 3 we plot how the mandatory and switch case components of the highest score achieved up to the current time change over time using both the Stochastic method and the Max Step method. We do not consider the Constant Step method since it is so highly susceptible to cycles. For both methods, as the score for mandatory activities increases, the score for switch groups largely decreases until the highest mandatory score is reached. This is a reasonable outcome because as more mandatory activities are scheduled, the schedule likely becomes more constricted, thus making it more difficult to schedule longer switch cases. Since the mandatory score contributes much more to the total score than the switch group score and the mandatory sore



(a) Mandatory score component of highest score so far vs Time averaged across sol type variants using both priority search methods.



(b) Switch group score component of highest score so far vs Time averaged across sol type variants using both priority search methods.

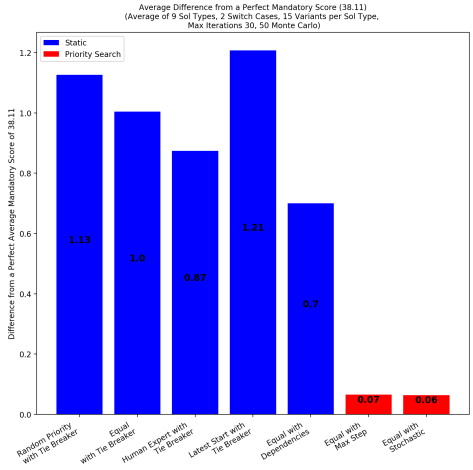
.

Figure 3: Plot of the highest score so far separated by mandatory score (3a) and switch group score (3b) over time using the Stochastic Step method and the Max Step method averaged over 9 sol types, each with 10 variants each containing 2 switch groups. Each iteration of Priority Search was run with 10 Monte Carlo runs and with 30 iterations of Priority Search alloted for each run of the algorithm.

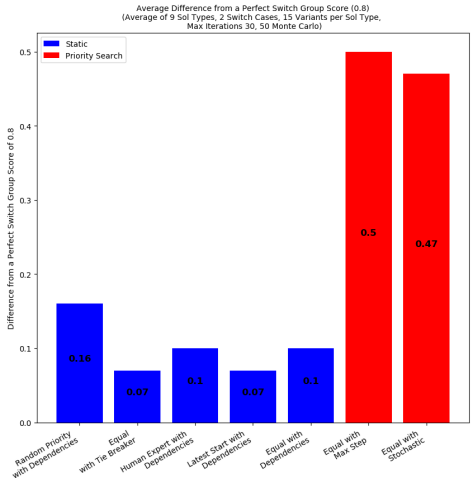
is increasing in both figures, the total highest score so far is always increasing over time, as it should be.

Figure 3a shows that Stochastic Step reaches its highest mandatory score that is ever achieved over the time span of approximately 920 seconds (30 iterations of the priority search algorithm) in 207.58 seconds. The highest mandatory score achieved at this time and onwards is 38.047. The high-

est mandatory score using the Max Step method is reached at 120.59 seconds and has a value of 38.044. Figure 3b shows that the highest switch group score after the point at which the highest mandatory score is reached is 1.67 at 568.16 seconds using the Stochastic method and 1.48 at 150.87 seconds using the Max Step method. Therefore, we conclude that using the stochastic method results in a marginally higher total highest score but it takes less time to reach the highest score using the Max Step method.



(a) Difference from perfect mandatory score averaged across sol type variants for various scheduling methods.



(b) Difference from perfect switch group score averaged across sol type variants for various scheduling methods.

Figure 4: The difference from a perfect mandatory score of 38.11 and perfect switch group score of 1.0 using various scheduling methods is averaged over 9 sol types where 15 variants are derived from each sol type and each variant contains 2 switch cases. Each iteration of the Priority Search algorithm is run with 50 Monte Carlo runs of execution

Figure 4 shows the results of comparisons between Priority Search and other static priority setting algorithms. Since the scheduling of mandatory activities and switch groups are not weighted equally, we have constructed two separate plots to show the results for each. Both methods of Priority Search, in red, result in fewer unscheduled mandatory activities and consequently a lower difference from the perfect mandatory score. This implies they set the priorities such that more mandatory activities are able to be scheduled over multiple Monte Carlo runs compared to how the static algorithms set the activity priorities. As shown in 4b, they result in a higher number of unscheduled switch cases, likely because if more mandatory activities were scheduled it becomes more difficult to schedule longer switch cases. Due to the strict inequality described in (2), even though fewer longer switch cases are scheduled, the total scheduling score is still higher when using Priority Search. Thus, we conclude that both Priority Search methods outperform the static algorithms. Among the static algorithms, running the Dependencies algorithm with Tie Breaker on equal priorities performs the best as it results in the highest mandatory score while running Tie Breaker after setting the priorities based on the latest start time performs the worst.

Related Work

Our Priority Search approach is inspired by Squeaky Wheel Optimization (SWO). Typically, SWO uses a constructor and analyzer, and prioritizer for the next iteration of schedule generation (Joslin and Clements 1999). Priority Search differs in that the intent is not to generate a good schedule but rather to set priorities that perform well in execution and rescheduling. Therefore the Priority Search constructor must use the scheduler through multiple runs of execution (where each run of execution incurs multiple scheduler invocations) to assess priority assignment performance.

Generating schedules that are robust to execution run time variations (Leon, Wu, and Storer 1994) is a mature area of work. However, the topic usually revolves around developing a scheduler that can generate robust schedules. In our case, the scheduler is a) a fixed ”black box” that we have no control over and b) robust to execution run time variations mainly through rescheduling (Chi et al. 2018). As a result, rather than developing a scheduler itself, we’re developing a methodology that is able to generate a set of priorities for a fixed scheduler that enables it to be robust to rescheduling due to runtime variations.

Other approaches (Drummond, Bresina, and Swanson 1994; Washington, Golden, and Bresina 2000) use branching to increase robustness - these differ from our work that adjusts priorities and allows rescheduling.

A number of other spacecraft (Muscettola et al. 1998; Pell et al. 1997; Chien et al. 2005; 2016) and rover (Woods et al. 2009; Gregory et al. 2002) autonomy systems have included planning, but these differ in that we are deriving control information specific to scheduling for a limited context - e.g. one rover sol. temporal schedule.

Discussion and Future Work

While we have focused on the impact of activity priority on the scheduler (and hence rescheduling during execution), there is often an execution system that may also have some flexibility to add robustness to the overall system (Chi et al. 2018). For the empirical evaluation described above, we ran without such an execution system. In the future, we could consider the execution system in the schedule and Monte Carlo analysis and potentially derive information usable by the execution system (e.g. allow an activity to run late but only until time T). This paper describes initial work to determine priorities for scheduler activity consideration ordering to optimize scheduler execution results for an embedded scheduler. However, this work is still preliminary with many other ideas to be explored as described below.

First, more sophisticated critique/blame assignment methods should be explored. Currently, priorities of activities that are not executed are modified, but more sophisticated analysis of scheduler runs could provide greater insight into how the priorities should be modified. Prior work in Process Chronologies (Biefeld and Cooper 1991) has been used to focus scheduling tactics by finding regions where time constraints or high demand for some resource results in conflict. By evaluating which periods of time or what resources are over-subscribed using Capacity/Over-Subscription Analysis, we can pinpoint which activities are more tightly constrained and increase their priorities. Prior work in Oversubscribed Scheduling Problems (Kramer and Smith 2006) show that scheduling according to maximum-availability (least subscribed) allows a suitable schedule to be generated. Similar analysis could be used to determine which activities to assign blame to and by how much to promote the blamed activities. We can also consider precedence constraints when deciding by how much to promote activity priorities. For every blamed activity, there is likely a scheduled activity that is using resources needed by the blamed activity. Precedence constraints could help discern which activity is using those resources. The blamed activity could then be promoted only as much as is necessary in order to be scheduled before the offending activity.

We can also implement several methods to help us explore different search spaces. Priority Search only adjusts priorities to improve execution and rescheduling performance. We could also add new activity precedence constraints (e.g. A must end before B starts) or enforce partitions in the schedule (e.g. all of these activities must be scheduled to end prior to 11 am). These types of constraints could drive the scheduler towards subsets of the schedule search space. Randomized restart can allow our priority search algorithm to better explore the global space rather than searching locally. Another alternative would be to keep a list of promising schedule priority assignments and backtrack to those randomly, allowing us to better explore the search space.

We can also make improvements to our Monte Carlo method and use the resulting simulations for further analysis of the scheduler. In order to build a model of run time variations that is not overly skewed, we use Monte Carlo to repeatedly sample a variety of execution run time results. Standard Monte Carlo simulations tend to focus most runs on the nominal cases, but a more effective methodology samples edge cases but weighs the cases by their likelihood to increase coverage of the variability in the space (in this case variable activity execution times). The Monte Carlo of execution run time variations can provide valuable information for why activities fail to schedule, what input plans are best suited for the current scheduler design, and how the current input could end up executing. We are working on visualizing this information to better inform those working with the scheduler.

Currently, we only test with mandatory activities. In the future, we will extend our approach to include optional activities, which will add further complexity to the algorithm and analysis. Optional activities are lower priority activities what are nice to have scheduled, but not necessary. They are generally only able to be scheduled if mandatory activities end early or consume less resources than expected. We also plan to use an activity's actual scheduled preferred time while testing.

Cycles pose an issue to both Constant Step Reprioritization and Max Step Reprioritization. Better cycle detection would allow us to not only overcome the issues presented, but also provide additional information on how to permute the priority set for the next iteration. For example, cycle detection could allow us to not only detect the cycle in Figure 2, but know that both E and F should be incremented together.

While we have established a few methods to improve the prioritizer and decide on the next permutation of activity priorities, we have utilized the same objective function to determine the success of our algorithm. However, our objective function is simple and coarse; oftentimes, the same score will appear repeatedly in multiple consecutive. As a result, the algorithm often travels swaths of plateaus before sharply improving. This choppiness is suboptimal for Squeaky Wheel Optimization and gradient descent problems in general. Some potential additions to the objective function could be how much energy is leftover in the plan or how close an activity is to their preferred scheduling time. Evaluating a more precise objective function can reduce choppiness and better steer the algorithm towards a more optimal solution.

Conclusion

We have presented a study of methods to assign activity priorities to control a limited, embedded scheduler to optimize rescheduling for a specific problem. We first define a set of static methods that assign activity priorities based on heuristics and schedule dependencies. We then describe how these priorities can be further adjusted based on feedback from simulated execution and rescheduling using Monte Carlo methods to perform Priority Search. We present an empirical evaluation of several static and priority search methods using best available planetary rover operations data. This empirical evaluation shows that Priority Search outperforms static methods including human expert derived priorities. Finally we describe a number of promising areas for future improvements to our algorithms.

Acknowledgments

This work was performed at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

References

Biefeld, E., and Cooper, L. 1991. Bottleneck identification using process chronologies. In *IJCAI*, 218–224.

Chi, W.; Chien, S.; Agrawal, J.; Rabideau, G.; Benowitz, E.; Gaines, D.; Fosse, E.; Kuhn, S.; and Biehl, J. 2018. Embedding a scheduler in execution for a planetary rover. In *ICAPS*.

Chien, S. A.; Sherwood, R.; Tran, D.; Cichy, B.; Rabideau, G.; Castano, R.; Davies, A.; Mandl, D.; Trout, B.; Shulman, S.; et al. 2005. Using autonomy flight software to improve science return on earth observing one. *Journal of Aerospace Computing Information and Communication* 2(4):196–216.

Chien, S.; Doubleday, J.; Thompson, D. R.; Wagstaff, K. L.; Bellardo, J.; Francis, C.; Baumgarten, E.; Williams, A.; Yee, E.; Stanton, E.; et al. 2016. Onboard autonomy on the intelligent payload experiment cubesat mission. *Journal of Aerospace Information Systems*.

Drummond, M.; Bresina, J.; and Swanson, K. 1994. Just-in-case scheduling. In *AAAI*, volume 94, 1098–1104.

Gaines, D.; Anderson, R.; Doran, G.; Huffman, W.; Justice, H.; Mackey, R.; Rabideau, G.; Vasavada, A.; Verma, V.; Estlin, T.; et al. 2016a. Productivity challenges for mars rover operations. In *Proceedings of 4th Workshop on Planning and Robotics (PlanRob)*, 115–125. London, UK.

Gaines, D.; Doran, G.; Justice, H.; Rabideau, G.; Schaffer, S.; Verma, V.; Wagstaff, K.; Vasavada, A.; Huffman, W.; Anderson, R.; et al. 2016b. Productivity challenges for mars rover operations: A case study of mars science laboratory operations. Technical report, Technical Report D-97908, Jet Propulsion Laboratory.

Gregory, N. M.; Dorais, G. A.; Fry, C.; Levinson, R.; and Plaunt, C. 2002. Idea: Planning at the core of autonomous reactive agents. In *Proceedings of the 3rd International Workshop on Planning and Scheduling for Space*. Citeseer.

Jet Propulsion Laboratory. 2017a. Mars 2020 rover mission https://mars.nasa.gov/mars2020/ retrieved 2017-11-13.

Jet Propulsion Laboratory. 2017b. Mars science laboratory mission https://mars.nasa.gov/msl/ 2017-11-13.

Joslin, D. E., and Clements, D. P. 1999. Squeaky wheel optimization. *Journal of Artificial Intelligence Research* 10:353–373.

Kramer, L. A., and Smith, S. F. 2006. Resource contention metrics for oversubscribed scheduling problems. In *ICAPS*, 406–409.

Leon, V. J.; Wu, S. D.; and Storer, R. H. 1994. Robustness measures and robust scheduling for job shops. *IIE transactions* 26(5):32–43.

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

Pell, B.; Gat, E.; Keesing, R.; Muscettola, N.; and Smith, B. 1997. Robust periodic planning and execution for autonomous spacecraft. In *International Joint Conference on Artificial Intelligence*, 1234–1239.

Rabideau, G., and Benowitz, E. 2017. Prototyping an onboard scheduler for the mars 2020 rover. In *International Workshop on Planning and Scheduling for Space*.

Washington, R.; Golden, K.; and Bresina, J. 2000. Plan execution, monitoring, and adaptation for planetary rovers. *Electron. Trans. Artif. Intell*.

Woods, M.; Shaw, A.; Barnes, D.; Price, D.; Long, D.; and Pullan, D. 2009. Autonomous science for an exomars rover–like mission. *Journal of Field Robotics* 26(4):358–390.

The Liquid Handling Robot Scheduling Problem

S. J. Edwards[a], D. Baatar[a], A. T. Ernst [a], and K. Smith-Miles[b]
[a]School of Mathematical Sciences, Monash University, Clayton, Australia;
[b]School of Mathematics and Statistics, The University of Melbourne, Melbourne, Australia
{steven.edwards, davaatseren.baatar, andreas.ernst}@monash.edu, smith-miles@unimelb.edu.au

Abstract

This paper introduces a simplified version of a problem that arises in scheduling an automated laboratory system, referred to as the Liquid Handling Robot Scheduling Problem. The laboratory system aims to process a number of jobs in parallel using a single robot. To process the jobs, the robot must transfer chemicals from a set of vials to the jobs in a pre-specified order using a single pipette with finite capacity. To ensure the proper chemical reactions occur, minimum and maximum time-lags exist between the dispensing of consecutive chemicals on the same sample. Robot travel times must also be considered. The objective is to process all the samples in the least amount of time, i.e., minimise makespan. The problem is modelled as a mathematical programming (MIP) and constraint programming (CP) problem. By assuming only one unit of chemical can be transferred at a time, the problem can be simplified into a special case of the single machine scheduling problem with time lags, for which two heuristics, the job insertion heuristic and the serial schedule generation schedule, are adapted from scheduling literature. The MIP model is able to prove optimality for a number of small instances. The CP model is not able to prove optimality for even small instances, yet manages to find good feasible solutions for many of the large instances. As the size of the problem increases, the exact methods are no longer able to find feasible solutions, and are outperformed by the heuristics.

Introduction

This paper considers a simplified version of a problem that arises in laboratory automation, which will be referred to as the Liquid Handling Robot Scheduling Problem (LHRSP). The problem considers a single robot that is responsible for processing a set of jobs. In reality these jobs represent experiments that are processed through the sequential application of chemicals. When processing a single experiment the robot remains idle for the majority of the time, waiting for the correct chemical reactions to occur. Thus to make better use of the robot the system is capable of processing many experiments in parallel.

The robot has a single pipette with finite volume that it uses to transfer chemicals to the set of jobs J. To transfer a given chemical from a vial to job $j \in J$ the robot must (1) move to the vials, (2) *aspirate* chemical up into the pipette tip, (3) travel to job j, (4) *dispense* the chemical onto the sample. A simple schematic of the problem is given in Figure 1. The robot takes $p^{\uparrow}$ time units to aspirate, and $p^{\downarrow}$ to dispense. The time taken for the robot to move from location j to j' is denoted $p^{\rightarrow}_{j,j'}$, where $j, j' \in J \cup \{0\}$ and 0 is a dummy job that represents the vial locations.

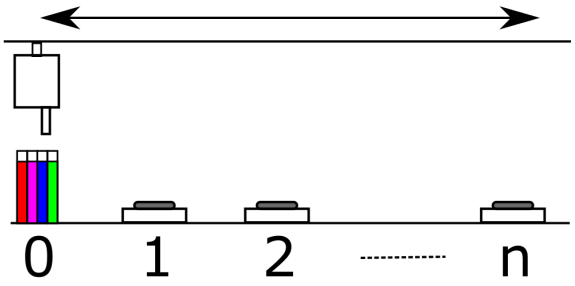


Figure 1: Schematic of the problem set-up. Vials are at location 0, jobs are at locations corresponding to the job's index. The robot can move between the vials and the jobs.

Job $j \in J$ has N_j required operations, each of which corresponds to the dispensing of a certain chemical. To complete a job, chemicals are required in a given order. Each job can be represented by a directed graph, as shown in Figure 2, where arcs represent the precedence relation between consecutive operations and nodes are color-coded to show the chemical required for the operation. Let C represent the set of the chemicals. Let $c_{i,j} \in C$ be the chemical required by operation i from job j. Note that one type of chemical might be required multiple times in the same job.

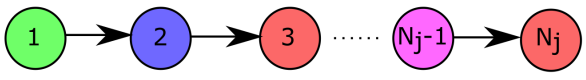


Figure 2: An example of a precedence graph that outlines the order in which the chemicals must be considered for a single job $j \in J$. Colours are used to indicate the chemical that is required by the operation.

To ensure that the correct chemical reactions occur there exist minimum and maximum time-lags between start times

of consecutive operations from the same job. Let $O_j := \{1, 2, ..., N_j\}$ be all the operations of job $j \in J$. The time-lags are defined for all operations of job $j \in J$, except for the very last operation, i.e., $\{O_j \setminus \{N_j\}\}$. Hence once an operation, $i \in O_j \setminus \{N_j\}$ of job $j \in J$, has started there is a minimum amount of time units, $\ell_{i,j}^{min}$, before operation $(i+1)$ is allowed to start. Likewise once operation $i \in O_j \setminus \{N_j\}$ has started there is a maximum amount of time, $\ell_{i,j}^{max}$, within which operation $(i+1)$ must start. These time-lags are expressed as

$$\ell_{i,j}^{max} \geq S_{i+1,j} - S_{i,j} \geq \ell_{i,j}^{min}, \quad (i \in O_j \setminus \{N_j\}, j \in J)$$

where $S_{i,j}$ is the start time of operation $i \in O_j$ from job $j \in J$,

The robot is allowed to aspirate multiple units of a single type of chemical at a time. This means the robot can, for example, (1) aspirate two units of a certain chemical, (2) move to a given job, (3) dispense one unit of the chemical at that job, (4) move to another job requiring the same chemical, (5) dispense the other unit of chemical at that job. For chemical $c \in C$ the robot can aspirate up to L_c units at a time. The time taken to aspirate multiple units of chemical is negligible and the processing time of each aspirate regardless of the quantity is $p^{\uparrow}$. Only one type of chemical can be in the pipette at a time. Given the robot can aspirate multiple units of a chemical at a time, it can effectively reduce the number of times that it must aspirate the chemicals.

In practice there are typically more jobs than the system can accommodate at once. Hence the objective is to process all the jobs in the least amount of time, i.e., minimise makespan.

Related Problems

The LHRSP has a number of closely related problems studied in the literature. Firstly, to prove the complexity of the problem we consider the minimal-makespan single machine scheduling problem with unit processing times and minimum time-lags (1p-SMSPmin), denoted by $PS1|chains(l_{ij}); p_i = 1|C_{max}$ using the project scheduling classification scheme (Brucker et al. 1999). The 1p-SMSPmin is known to be NP-Hard (Yu, Hoogeveen, and Lenstra 2004).

Theorem 1. *The LHRSP is NP-Hard*

Proof. We reduce 1p-SMSPmin to the LHRSP. Let I be an instance of 1p-SMSPmin. For each chain in I construct a job $j \in J$, where each activity in the chain corresponds to an activity $i \in O_j$. Let $\ell_{i,j}^{min} = l_{i,j}$ and $\ell_{i,j}^{max} = M$, where M is some sufficiently large number and $l_{i,j}$ is the minimum time-lags specified in I. Now let $p^{\downarrow} = 1$, $p^{\uparrow} = 0$, $p_{j,j'}^{\rightarrow} = 0$, for all $j, j' \in J$. For completeness let $|C| = 1$, $c_{i,j} = c$ and $L_c = |\bigcup_{j \in J} O_j|$. □

If it is assumed that only one unit of chemical can be transferred at a time, i.e., $c \in C$ is $L_c = 1$, the LHRSP can be simplified into a special case of the single machine scheduling problem with time lags (SMSP-TL), which is denoted $PS1|chains, temp|C_{max}$. The aspirate, travel and dispense times are incorporated into the processing time of the activity. Hence all activities $i \in O_j$ from job j have the same processing time, defined by $p_j := p^{\uparrow} + p_{0,j}^{\rightarrow} + p_{j,0}^{\rightarrow} + p^{\downarrow}$. The time-lags are still valid as all operations in the same job have the same processing time, now p_j as opposed to simply the dispense time $p^{\downarrow}$, and time-lags only exist between operations of the same job.

The SMSP-TL is closely related to the well-studied job shop scheduling problem with time-lags (JSP-TL), denoted by $J_m|temp|C_{max}$ (Brucker, Hilbig, and Hurink 1999). As pointed out in (Caumond, Lacomme, and Tchernev 2008), when considering maximum time-lags building non-trivial feasible solutions is not straight forward, however scheduling all operations of a job after all operations of another job leads to feasible schedules. These types of schedules are referred to as *canonical schedules*. It is possible to generate similar feasible solutions to the LHRSP, albeit very poor ones, by completing all operations of one job, then all operations of the next job, and so on.

To find better initial solutions than the canoncial schedules for the JSP-TL, (Caumond, Lacomme, and Tchernev 2008) proposed a list-based heuristic combined with a method for repairing solutions. Following this, (Artigues, Huguet, and Lopez 2011) proposed a job insertion heuristic (JIH), which exploits the fact that precedences only exist between consecutive operations of the same job. In the experimental study in (Artigues, Huguet, and Lopez 2011), the JIH is shown to produce better solutions than the list-based heuristic from (Caumond, Lacomme, and Tchernev 2008). Moreover the state-of-the-art approach to the JSP-TL, (González et al. 2015), also utilizes the JIH to obtain initial feasible solutions to their scatter-search approach. In this paper we adapt the JIH to the LHRSP.

Both the SMSP-TL and JSP-TL are special cases of the well-studied resource-constrained project scheduling problem with generalised precedence constraints (RCPSP-max), denoted by $P|temp|C_{max}$. Constraint programming (CP) approaches, such as (Schutt et al. 2013) and (Vilim, Laborie, and Shaw 2015), have been particularly effective for this RCPSP-max. Furthermore for the RCPSP-max there are a number of effective schedule generation schemes such as serial schedule generation scheme (SSGS) with unscheduling step, also referred to as the *direct method* proposed by (Franck, Neumann, and Schwindt 2001). This method has been modified for many practical applications, such as the generalised surgery scheduling problem (Riise, Mannino, and Burke 2016). In this paper we adapt the SSGS to the LHRSP.

The LHRSP also closely relates to a number of scheduling problems where material handling is considered. In particular, the problem has a strong resemblance to hoist scheduling problems (HSP)s, which have been motivated by electroplating lines. There are many variants of the HSP considered in the literature, see (Manier and Bloch 2003) for a very detailed classification. In general one or more hoists move different *carriers*, which can be thought of as different jobs, between chemical baths, within which the jobs must remain between a minimum and maximum allowable time. Superfi-

cially, the LHRSP resembles a type of HSP where instead of transferring jobs between different chemical baths, the hoist transfers different chemicals to the jobs.

From the perspective of scheduling, the LHRSP has a number of additional challenges that make it difficult to model as a type of HSP. In HSPs, a hoist carries a single job at a time. Furthermore once a job has been lifted from one chemical bath, the hoist must travel and lower the job into the next chemical bath immediately. This is known as the *no-wait requirement* and is enforced to ensure the jobs are not damaged by oxidation during transfer. A result of this is that the transport tasks have a known, fixed duration, and the scheduling of the time the jobs are in the baths can be accounted for in the scheduling of the transfer tasks. In contrast, the LHRSP can transfer a number of units of chemicals simultaneously. This introduces a number of fundamental complexities, such as not knowing how many times the robot must aspirate prior to scheduling.

Finally, as will be discussed in our mixed integer programming (MIP) model, the problem can be seen as a travelling salesman problem (TSP) with minimum and maximum time-lags with additional side constraints. To the best of our knowledge, no papers have studied this pure TSP with minimum and maximum time-lags between deliveries, however similar problems such as the vehicle routing problem with temporal dependencies have been studied by (Dohn, Rasmussen, and Larsen 2011). Practical applications that consider vehicle routing with timelags include homecare crew routing problem (Rasmussen et al. 2012) and the concrete delivery prolem (Kinable, Wauters, and Vanden Berghe 2014), for which both CP and MIP models have been effective.

Mathematical models

This section introduces a MIP model and a CP model for the LHRSP. A key modelling choice in the LHRSP is modelling how the robot aspirates chemical. As was previously mentioned, it is not clear a priori how many times the robot must aspirate the chemicals. The MIP and CP models differ greatly on how they model aspiration. The MIP model incorporates the time required for aspiration into the robot travel times. Whereas the CP model enumerates the maximum number of aspirations that might be required and models them all explicitly as additional variables. As will be seen, both modelling choices introduce additional side constraints.

Let $O = \bigcup_{j \in J} O_j$ be the set of all operations. To distinguish between dispenses of different chemicals, let all the activities requiring $c \in C$ be defined as $O^c = \{(i,j) \in O | c_{i,j} = c\}$.

Mathematical Programming model

In the mathematical programming approach, the problem is modelled as a routing problem by the use of a directed, weighted multi-graph $G(V, A)$. Each operation $(i,j) \in O$ is assigned a vertex in the graph, $v(i,j)$, as well as a dummy start and dummy end node, denoted by 0 and $\gamma := |O| + 1$ respectively. Hence $V := \{0\} \cup \{v(i,j) | i,j \in O\} \cup \{\gamma\}$. In

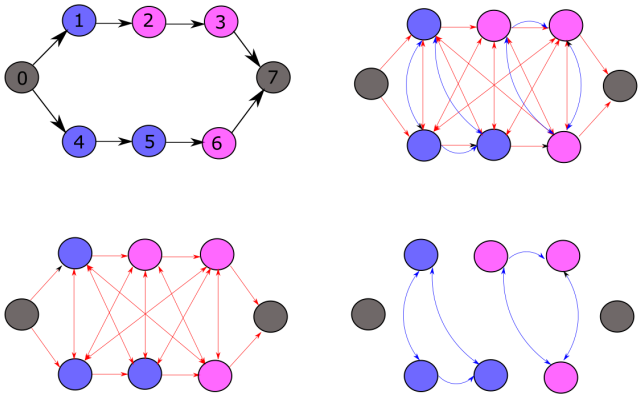


Figure 3: An example of how the multigraph is constructed from imposing two networks. The top-left graph shows the precedence graph of two jobs with three operations each, where the colour of the nodes indicates which of the two chemicals are required. The top-right graph shows the multigraph that is used in the routing problem, which comes from superimposing the slow routes (bottom-left) and fast routes (bottom-right)

a multi-graph, there may be multiple arcs from one node to another, referred to as different *routes*. Hence we use triplet notation to represent arcs, where $(v, v', r) \in A$ represents an arc from node v to v' along route r and has weight $w_{v,v',r}$. The arc set is the union of two disjoint subsets $A = A_1 \cup A_2$, each of which contain at most a single arc from one node to another. Hence there is at most two routes from one node to another in the multi-graph G.

Arc set A_1 can be thought of as the *slow routes*. An arc $(v, v', 1) \in A_1$ has a weight that represents the minimum amount of time required between the start of $v \in V$ and start of $v' \in V$ if the robot aspirate chemicals in between. More explicitly the arc set and equivalent weights can be defined as

- An arc from the dummy start to the node corresponding to the first operation of all jobs, i.e., $(0, v(1,j), 1) \in A_1$, with arc weight $w_{0,v(1,j),1} = p^{\uparrow} + p^{\rightarrow}_{0,j}$, for all $j \in J$. The arc weight comes from the robot aspirating chemical and then travelling to the correct location.
- An arc between operations from different jobs or consecutive operations of the same job, i.e., $(v(i,j), v(i',j'), 1) \in A_1$ for all $(i,j), (i',j') \in O$ given that $j \neq j'$ or $(i',j') = (i+1, j)$ with weight $w_{v(i,j),v(i',j'),1} = p^{\downarrow} + p^{\rightarrow}_{j,0} + p^{\uparrow} + p^{\rightarrow}_{0,j'}$. The arc weight comes from the robot dispensing chemical at job j, moving to the vials, aspirating the next appropriate chemical, and then travelling to job j'.
- An arc from the last operation of all jobs to the dummy end, i.e., $(v(N_j, j), \gamma, 1) \in A_1$ for all $j \in J$ with arc weight $w_{v(N_j,j),\gamma,1} = p^{\downarrow}$. The arc weight is simply the time required to dispense the chemical.

Arc set A_2 can be thought of as the *fast routes*. An arc $(v, v') \in A_2$ has a weight that represents the minimum amount of time required between the start of v and v' if the robot travels directly between the two without aspirating new reagent in between. Since the robot can not carry different chemicals at the same time, arcs can only exist in A_2 if both the predecessor and successor require the same chemical. More explicitly, A_2 and the associated weights can be defined as

- An arc between operations that require the same colour from different jobs or consecutive operations of the same jobs, i.e., $(v(i,j), v(i',j'), 2)$ for all $(i,j),(i',j') \in O^c$ given that $j \neq j'$ or $(i',j') = (i+1,j)$ with weight $p^{\downarrow}+p^{\rightarrow}_{j,j'}$. The arc weight comes from the robot dispensing chemical at job j and then travelling directly from job j to j'.

A small example illustrating how the multigraph is constructed is shown in Figure 3. The example considers an instance with two jobs, each of which contain three operations, and two colours. The precedence graph is also shown, where evidently nodes $\{1,2,3\}$ belong to one job and $\{4,5,6\}$ belong to the other. Furthermore nodes 0 and 7 are the dummy start and end nodes respectively.

In order to tighten the mathematical model, for each vertex $v \in V$ an earliest start time ES_v and latest start time LS_v are defined. The ES_v can be calculated by the longest path between 0 and v in the precedence graph. Whereas LS_v can be set to M minus the length of the longest path between v and γ, where M is a sufficiently large number that we generally obtain from from a heuristic, discussed in the *Heuristics / Upper Bounds* section. Finally for ease of notation let $Vc := \{v(i,j) | (i,j) \in O^c\}$ be the set of nodes associated with operations requiring chemical $c \in C$.

The entire mathematical model can now be formulated as,

$$\text{Min.} \quad S_\gamma \tag{1}$$

$$\sum_{(v',v,r)\in\delta^-(v)} x_{v',v,r} = 1, \qquad (v \in V) \tag{2}$$

$$\sum_{(v,v',r)\in\delta^+(v)} x_{v,v',r} = 1 \qquad (v \in V) \tag{3}$$

$$\sum_{\substack{((v,v',r)\in A_2:\\ v\in P \wedge v'\in P)}} x_{v,v',r} \leq L_c - 1, \qquad (c \in C, P \subseteq V^c : |P| = L_c + 1) \tag{4}$$

$$\ell^{min}_{i,j} \leq S_{v(i+1,j)} - S_{v(i,j)} \leq \ell^{max}_{i,j}, \qquad (i \in O_j \setminus \{N_j\}, j \in J) \tag{5}$$

$$S_v - M_{v,v'}(1 - x_{v,v',r}) \leq S_{v'} - \sum_{\substack{r'\in\{1,2\}:\\ (v,v',r')\in A}} w_{v,v',r'} x_{v,v',r'}, \qquad ((v,v',r) \in A | v \neq \gamma) \tag{6}$$

$$\sum_{(v,v',r)\in A} w_{v,v',r} x_{v,v',r} \leq S_\gamma \tag{7}$$

$$\sum_{v\in V_c} \sum_{\substack{(v,v',r)\in\delta^+(v):\\ r=2}} x_{v,v',r} \geq \bar{\alpha}_c, \qquad (c \in C) \tag{8}$$

$$x_{v,v',r} \in \{0,1\} \qquad ((v,v',r) \in A) \tag{9}$$

$$ES_v \leq S_v \leq LS_v \qquad (v \in V) \tag{10}$$

Binary variables $x_{v,v',r}$ denote whether the robot moved from v to v' along route r. Continuous variable S_v records the time that the robot arrives at node v. In addition, S_γ records the makespan of the schedule.

The objective is to minimise the makespan, or equivalently, the start time of the dummy variable, specified in (1). Constraints (2) and (3) enforce that the in-degree and out-degree of each node is exactly one, respectively, where $\delta^-(v)$ and $\delta^+(v)$ represent the incoming and outgoing arcs from node $v \in V$ respectively.. Constraint (4) is a subpath elimination constraint, which ensures that the robot does not dispense more chemical than it can carry. Given the exponential number of the subpath elimination constraints, these are added lazily into the model at each integer solution. At each integer solution we can determine the sets of nodes between which the robot travels only using the fast routes, $\overline{P} := \{\overline{P}_1, \overline{P}_2, ...\}$. Given the structure of the multigraph, each of these sets of nodes, $\overline{P}_p \in \overline{P}$ are associated with a specific colour, $c(\overline{P}_p) \in C$. If the size of this set of nodes exceeds the possible limit, $|\overline{P}_p| > L_{c(P_p)}$, we add lazy constraints for each subset $P \subseteq \overline{P_p}$ of the sets of nodes according to (4).

Constraints (5) enforce the minimum and maximum time-lags between adjacent operations from the same job. Constraints (6) are linking constraints that relate the arrival time of a node to the arrival time of the node that the robot directly travels from, here $M_{v,v'} := LS_v - ES_{v'}$ is a sufficiently large number. Moreover, these linking constraints remove the need for sub-tour elimination constraints that are required by similar MIP formulations for related routing problems. It should be noted that in constraints (6) if there exists two arcs between the considered nodes then we sum the weights of both arcs to tighten fractional solutions. Constraints (7) and (8) are tightening constraints. Constraint (7) enforce that the makespan is at least larger than the weight of all the arcs completed. Whereas constraints (8) ensure that the number of slow routes exiting nodes requiring chemical $c \in C$ is at least equal to the minimum number of aspirates, α_c, which can be calculated by $\overline{\alpha}_c = \left\lceil \frac{|O^c|}{L_c} \right\rceil$. Finally constraints (9) state that the $x_{v,v',k}$ are binary and constraints (10) are continuous within the range of the earliest and latest start times.

Constraint Programming model

Constraint Programming offers an alternative means to model the LHRSP. In the discussion that follows we use constraints present in IBM's CP Optimizer. Our model utilizes interval variables (Laborie and Rogerie 2008) (Laborie 2009). An interval variable represents an interval of time.

More formally an interval variable α is a variable where domain $dom(\alpha)$ is a subset of $[s, e)|s, e \in Z, s \leq e]$. An interval variable is fixed if its domain is reduced to singleton, i.e., α denotes a fixed interval variable. We require all interval variables to be scheduled however not all interval variables will be accounted for in the objective function.

Each interval variable α has a start time $startOf(\alpha)$, an end time $endOf(\alpha)$, and a duration $dur(\alpha)$. As shorthand notation, an interval variable α is defined as a tuple: $\alpha = [ES, LC, dur]$, specifying the earliest start time, latest completion time and duration respectively. If the duration is not specified then it is not fixed. The constraints used in this model are summarised in Table 1.

The CP model, Algorithm 1, considers both the aspirates and dispenses explicitly as interval variables, lines 1 and 3 respectively. At most $|O|$ aspirates are required and hence all are considered as interval variables by the model. However as the makespan is taken as the completion of the last dispense operation, only the dispense variables are considered in the objective function, as is seen in line 7. The minimum and maximum time-lags between dispense intervals are enforced by appropriate constraints on lines 11 and 12 respectively. To ensure that the robot can only complete one aspirate or dispense interval at a time, while also taking into account travel times between them, a noOverlapSequence is added in line 24.

In order to take into account the logic concerning the different chemicals as well as the finite pipette capacity, two additional types of interval variables and two state functions are introduced. The *carry variables*, line 4, are used to represent the time between aspirates when the robot is moving and dispensing chemical at different jobs. An alternating sequence is enforced on the aspirate and carry variables in lines 18 and 21. Each dispense variable has an associated *cover variables*, line 2, which must start before the start, and end after the end, of the dispense variable, constrained in lines 13 and 14 respectively.

A state function f is a decision variable whose value is a set of non-overlapping intervals, where each interval is associated with a non-negative integer value that represents the start of the function over the interval. The *colour state*, line 5, accounts for what chemical the pipette is currently carrying. Hence the colour state is c when chemical $c \in C$ is being carried and 0 when no chemical is being carried. Whereas the *pipette state*, line 6, accounts for whether any chemical is currently being carried at all. The pipette state is 1 when a chemical is being carried and 0 otherwise.

Each cover variable is both left and right aligned to a pipette state of 1 and to a colour state associated with the chemical required by the enclosed dispense variable, specified by lines 16 and 15 respectively. Additionally, the carry variables are left and right aligned to a pipette state of 1, line 19. These constaints acting in unison enforce that only dispense variables requiring the same reagent can occur between successive aspirations.

Finally to account for the finite capacity of the pipette, a cumulative function is associated with each colour. Each carry variable produces L_c units of each chemical $c \in C$ for the duration of the interval and each cover variable consumes a unit of the required chemical for the duration of its interval. The cumulative resource is then constrained to always be non-negative, line 23.

Algorithm 1 CP Model

Variable definitions:
1: d_vars$_{i,j}$ = {0, inf, $p^{\downarrow}$} $\quad \forall i \in O_j, j \in J$
2: cover_vars$_{i,j}$ = {0, inf, -} $\quad \forall i \in O_j, j \in J$
3: a_vars$_i$ = {0, inf, $p^{\uparrow}$} $\quad \forall i \in \{1, ..., |O|\}$
4: carry_vars$_i$ = {0, inf, -} $\quad \forall i \in \{1, ..., |O|\}$
Expressions:
5: colour_state = $state\_function()$
6: pipette_state = $state\_function()$
Objective:
7: *Minimise* $\max_{(i,j)\in V}(endOf(\text{d\_vars}_{i,j}))$
Constraints:
8: **for** $j \in J$ **do**
9: **for** $i \in O_j$ **do**
10: **if** $i \neq N_j$ **then**
11: startBefStart(d_vars$_{i,j}$, d_vars$_{i+1,j}$, $\ell^{min}_{i,j}$)
12: startBefStart(d_vars$_{i+1,j}$, d_vars$_{i,j}$, -$\ell^{max}_{i,j}$)
13: startBefStart(cover_vars$_{i,j}$, d_vars$_{i,j}$)
14: endBeforeEnd(d_vars$_{i,j}$, cover_vars$_{i,j}$)
15: alwaysEqual(colour_state, cover_vars$_{i,j}$, $c_{i,j}$)
16: alwaysEqual(pipette_state, cover_vars$_{i,j}$, 1)
17: **for** $i \in \{1, ..., |O|\}$ **do**
18: endBefStart(a_vars$_i$, carry_vars$_i$)
19: alwaysEqual(pipette_state, carry_vars$_i$, 1)
20: **if** $i \neq n$ **then**
21: endBefStart(carry_vars$_i$, a_vars$_{i+1}$)
22: **for** $c \in C$ **do**
23: $\sum_{i\in\{1,...|O|\}} pulse(\text{carry\_vars}_i, L_c) - \sum_{i,j\in O^c} pulse(\text{cover\_vars}_{i,j}, 1) \geq 0$
24: $noOverlapSeq($
$\bigcup_{i,j\in O}(\text{d\_vars}_{i,j}, j) \cup \bigcup_{i\in\{1,...,|V|\}}(\text{a\_vars}_i, 0)],$
$\bigcup_{j,j'\in J\cup\{0\}} \overrightarrow{p_{j,j'}})$

Example CP Solution

The CP model is best understood through a small example. Here we will consider the same problem considered in Figure 3. We assume the capacity of the pipette for both colours is 2. A Gantt chart of an optimal solution is shown in Figure 4. The Gantt chart consists of four sections. The top section displays the schedule of the robot, consisting of the aspirate intervals, labelled with the letter "A", and the dispense intervals, labelled (i, j) for all $(i, j) \in O$. The dispense operations are coloured to represent the chemical required. The second top section display the carry intervals, labelled "C". It can be clearly seen that the schedule alternates between aspirate intervals and carry intervals. The second bottom section displays the cover intervals with the appropriate colour, labelled (i, j) for all $(i, j) \in O$. The bottom section displays

Table 1: Description of functions used in the CP Model

Constraint	Description
startBefStart(α, β, z)	A precedence constraint of the form $startOf(\alpha) + z \leq startOf(\beta)$
endBefEnd(α, β, z)	A precedence constraints of the form $endOf(\alpha) + z \leq endOf(\beta)$
alwaysEqual(f, α, v)	Interval variable α must start at the beginning and end at the end of some interval where state function f is maintained in state v
pulse(α, h)	The amount of resource available between $startOf(\alpha)$ and $endOf(\alpha)$ is increased by h units
noOverlapSeq(B, dist)	B is a set of tuples of the form $[\alpha, t]$ where α is an interval variable of type t. Furthermore *dist* is a 2-dimensional matrix that specifies the sequence-dependent set-up between each type of interval variable.

both the colour state and the pipette state. All cover and carry intervals are aligned to the colour and pipette states within which they are scheduled. The cover intervals are only executed in the appropriate colour state. Each dispense interval occurs within the associated cover interval.

Heuristics / Upper Bounds

In this section we outline how a number of heuristics in the literature can be adapted to the LHRSP. Firstly, the canonical schedules provide an upper bound, determined by

$$UB = \sum_{j \in J} (p^{\uparrow} + p^{\rightarrow}_{0,j} + p^{\downarrow} + p^{\rightarrow}_{j,0} + \sum_{O_j \setminus \{N_j\}} \max(\ell^{min}_{i,j}, p^{\uparrow} + p^{\downarrow} + p^{\rightarrow}_{0,j} + p^{\rightarrow}_{j,0})) \quad (11)$$

A better upper bound can be determined by transforming the problem into a special case of the SMSP-TL, as discussed in *Related Problems*, and then modifying the SSGS and JIH heuristics from the literature. There are a number of similarities between the two heuristics. Both heuristics insert jobs one at a time into the schedule. Let $\mathcal{L}$ be the ordered set, in which the jobs are considered. Furthermore both heuristic contain a back-tracking process to account for failures incurred due to the maximal time-lags.

Serial Schedule Generation Scheme (SSGS)

The algorithm for the modified SSGS is shown in Algorithm 2. The algorithm exploits the much simplified structure of the SMSP-TL compared with the RCPSP-max, i.e., each operation has at most one predecessor / successor, and that there is only single resource with a unit capacity. The algorithm considers all operations of one job before the next.

In order to schedule an operation, the algorithm finds the earliest time, after the associated earliest start time, such that the resource is available. If this time is below the associated latest start time, then the operation is scheduled at the time, the resource profile is updated as well as the earliest start and latest start times of the next operation. However if the candidate time is above the associated latest start time, then the direct predecessor's maximum time-lag is violated and an unscheduling step is performed, seen in line 8. The unscheduling step works by updating the earliest start time of the operation's direct predecessor to ensure that the maximum time-lag would not be violated. The algorithm continues until a feasible solution is found.

Algorithm 2 Serial Schedule Generation Scheme (SSGS)

1: Initial resource profile $r(.)$, $ES_{i,j} \leftarrow 0$, $LS_{i,j} \leftarrow UB$
2: **for** each job $j \in \mathcal{L}$ **do**
3: $\quad i \leftarrow 1$
4: $\quad$ **while** $i \leq N_j$ **do**
5: $\quad\quad t^* := \min\{t \geq ES_{i,j} | r(\tau) = 0 \forall \tau \in [t, t + p_j[\}$
6: $\quad\quad$ **if** $t^* > LS_{i,j}$ **then**
7: $\quad\quad\quad u := u + 1$
8: $\quad\quad\quad ES_{i-1,j} := S_{i-1,j} + t^* - LS_{i,j}$
9: $\quad\quad\quad r(\tau) := 0 \forall \tau \in [S_{i-1,j}, S_{i-1,j} + p_j[$
10: $\quad\quad\quad i := i - 1$
11: $\quad\quad$ **else**
12: $\quad\quad\quad S_{i,j} \leftarrow t^*$
13: $\quad\quad\quad r(\tau) \leftarrow 1 \forall \tau \in [t^*, t^* + p_j - 1]$
14: $\quad\quad\quad ES_{i+1,j} \leftarrow S_{i,j} + \max\{p_j, \ell^{min}_{i,j}\}$
15: $\quad\quad\quad LS_{i+1,j} \leftarrow S_{i,j} + \ell^{max}_{i,j}$
16: $\quad\quad\quad i \leftarrow i + 1$

Job Insertion Heuristic

The algorithm of the modified JIH is given in Algorithm 3. Like the SSGS, the JIH considers jobs in a given order. The main point of difference between the two heuristics is that the JIH keeps track of the intervals into which an operation can be inserted as opposed to enumerating the entire resource profile across a given time horizon.

In order to schedule an operation, the algorithm iterates over all the intervals starting from the interval into which it's predecessor was inserted. Operation (i, j) can be inserted into an interval, $I := [\underline{I}, \overline{I}]$, if and only if the following conditions hold,

$$\overline{I} - \underline{I} \geq p_j \quad (12)$$

$$\overline{I} - p_j - ES_{i-1,j} \geq \max(p_j, \ell^{min}_{i-1,j}) \quad (13)$$

$$\underline{I} - LS_{i-1,j} \leq \ell^{max}_{i-1,j} \quad (14)$$

If it is possible for an operation to be inserted into an interval, then the earliest and latest start times of the operation are updated using the following expressions,

$$ES_{i,j} = \max(\underline{I}, ES_{i-1,j} + \max(\ell^{min}_{i-1,j}, p_j)) \quad (15)$$

$$LS_{i,j} = \min(\overline{I}, LS_{i-1,j} + \ell^{max}_{i-1,j}) \quad (16)$$

If conditions (12) and (13) are valid but not (14), then the maximal time-lags have been violated and thus it will not be possible to insert the operation into any of the intervals, shown in line 10. The algorithm then performs back

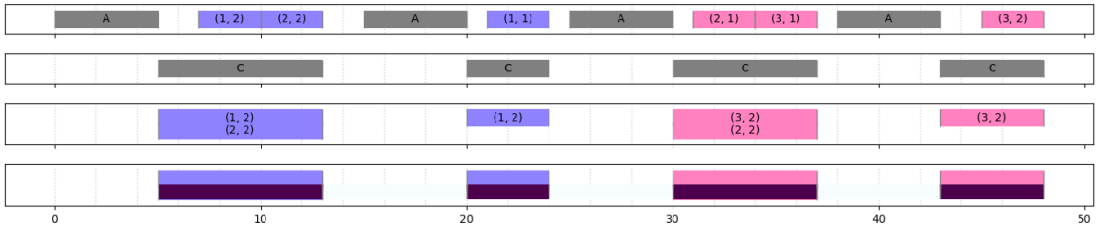


Figure 4: Gantt charts from CP Solver output

tracking and returns to scheduling the previous operations but starting from the interval after it previously was inserted into. Once all the operations of a job are inserted, the start times of the operations are fixed, and the set of intervals are updated. Only intervals for which $\overline{I} - \underline{I} \geq \min_{j \in J} p_j$ are considered in the list of possible intervals.

Algorithm 3 Job Insertion Heuristic (JIH)

for each job $j \in \mathcal{L}$ **do**
  $i \leftarrow 1$; inserted←false; $q \leftarrow 1$
  while *not inserted* **do**
    for *each possible interval starting from* q **do**
      if Conditions (12)-(14) are verified **then**
        compute $ES_{i,j}$ and $LS_{i,j}$ according to expressions (15)-(16)
        inserted ← true; break;
      else if Conditions (12),(13) are verified **then**
        break;
    if inserted = false **then**
      $i \leftarrow i - 1$; $q \leftarrow q_i + 1$
    else
      $q_i \leftarrow q$; $i \leftarrow i + 1$; inserted ← false
  Fix start times for job j and update intervals for job $j + 1$

Computational Study

Data

Unfortunately due to the confidential nature of the problem domain, it is not possible to share any real-world data. To test the different approaches we have generated a data set of 18 instances where parameters are chosen from intervals that replicate the structure of the real world problem.

All jobs in an instance have the same number of operations denoted by m, i.e., $|O_j| = m$ for all $j \in J$. The *size* of an instance is represented by a tuple, $(n, m, |C|)$, which describes the number of jobs, number of operations per job and the number of colours respectively. For all instances the processing times are as follows, $p^{\rightarrow}_{j,j'} = |j' - j|$ for all $j, j' \in J \cup \{0\}$, $p^{\downarrow} = 10$, $p^{\uparrow} = 20$. The carry limit L_c of each chemical $c \in C$ is taken as a random integer between 2 and 5. The minimum time-lags, $\ell^{min}_{i,j}$ are always divisible by 5 and are selected at random from the interval $[p^{\downarrow}, n * (p^{\downarrow} + p^{\uparrow} + 2n)]$, to allow the minimum time-lags to increase as the number of jobs increases. To ensure feasibility, the minimum time-lag is used when constructing the corresponding maximum time-lag, by introducing factor $\epsilon_{i,j} = max(\ell^{min}_{i,j}, p^{\downarrow} + 2n + p^{\uparrow})$, which is the maximum of the minimum time-lag and the minimum amount of time the robot requires to complete consecutive operations for the furthest job if the operations require different chemicals. The maximum time-lags, $\ell^{max}_{i,j}$ are always divisible by 5 and are selected at random between $\ell^{max}_{i,j} \in [\epsilon_{i,j}, n\epsilon_{i,j}]$. This allows the maximum time-lags to increase when more jobs are considered.

A preprocessing step can be performed to determine the minimum amount of time between consecutive operations, (i, j) and $(i + 1, j)$ from the same job, $j \in J \setminus \{N_j\}$. If consecutive operations from the same job require different chemicals then the robot must travel back to the vials to aspirate more chemical between these two operations. On the other hand, if the consecutive operations from the same job require the same chemical, then the robot still must dispense the first unit of chemical before it can dispense the second. More explicitly, for all $(i, j) \in O$, define the preprocessed minimum time-lag as follows,

$$\overline{\ell}^{min}_{i,j} := \begin{cases} \max(\ell^{min}_{i,j}, p^{\downarrow} + p^{\uparrow} + p^{\rightarrow}_{0,j} + p^{\rightarrow}_{j,0}) & \text{if } c_{i,j} \neq c_{i+1,j} \\ \max(\ell^{min}_{i,j}, p^{\downarrow}) & \text{otherwise} \end{cases}$$

Results

Experiments have been tested under the Windows 10 (64-bit) operating system with 8GB RAM and Intel® Core™ i7-3537U, 2.5GHz processor. The MIP model was implemented in Gurobi 7.0.2 and the CP model was implemented in IBM ILOG CPLEX Optimization Studio V12.8.0 CP Optimizer. The MIP and CP approaches are given 4 threads and 10 minutes wall-time. The two heuristics, SSGS and JIH, were implemented in Python 2.7 and each run 100 times per instance where the order in which jobs were considered were shuffled randomly. The best solution obtained by the two heuristics is used as the upper bound in the MIP model. No upper bound is given to the CP model as we are particularly interested in understanding the effectiveness of the heuristics used by CP Optimizer to find feasible solutions.

The results to the computational study are summarised in Table 2. As the lower bounds obtained by the linear relaxation of the MIP model and the initial propagation of the root node of the CP model were equivalent for all instances, these values are given in a single column, *lb-init.*. The columns, *lb*, represent the best lower bound determined by the MIP

Table 2: Computational results

Inst.	Size	lb-init.	ub-canon.	SSGS		JIH		MIP				CP	
				ub	t(s)	ub	t(s)	lb	lazy	ub	t(s)	lb	ub
1	(3-5-2)	292	951	711	0.001	704	0.001	361	7	**361**	3.69	301	**361**
2	(3-5-3)	366	1232	748	0.001	647	0.002	477	-	**477**	3.12	-	**477**
3	(3-5-5)	396	1120	839	0.002	650	0.001	588	-	**588**	8.62	-	**588**
4	(5-5-2)	692	3061	1074	0.004	1024	0.003	-	54	**716**	-	-	**716**
5	(5-5-3)	664	2689	1134	0.004	1079	0.001	-	12	749	-	-	**707**
6	(5-5-5)	624	2455	996	0.003	1055	0.001	-	9	649	-	-	**644**
7	(5-10-2)	1134	5737	2410	0.009	2353	0.008	-	-	-	-	1144	**1341**
8	(5-10-3)	1045	4901	2480	0.013	2340	0.003	-	-	-	-	-	**1267**
9	(5-10-5)	1178	5009	2304	0.013	2147	0.004	-	-	-	-	-	**1475**
10	(10-10-2)	2833	25919	5166	0.058	5253	0.001	-	-	-	-	-	**3077**
11	(10-10-3)	2664	23421	5251	0.051	5161	0.001	-	-	-	-	-	**3076**
12	(10-10-5)	2973	25337	5258	0.055	5089	0.001	-	-	-	-	-	**3346**
13	(15-20-2)	10349	130271	18760	0.789	17493	0.005	-	-	-	-	-	**11404**
14	(15-20-3)	10707	133603	18848	0.868	18206	0.005	-	-	-	-	-	**13836**
15	(15-20-5)	11234	143173	19403	1.041	18879	0.004	-	-	-	-	-	**17081**
16	(20-30-2)	23324	420718	43075	5.239	**42746**	0.111	-	-	-	-	-	-
17	(20-30-3)	25087	425266	43099	5.826	**40757**	0.012	-	-	-	-	-	-
18	(20-30-5)	23048	420748	42571	5.578	**41813**	0.013	-	-	-	-	-	49639

and CP models and, for clarity, are left blank if the algorithm does not improve the lower bounds. The *ub-canon.* represents the upper bound as given by the canonical solution. The columns ub represent the best objective value determined by each of the four approaches. For the SSGS and JIH, the columns, $t(s)$, represent the average wall time of the heuristic across the 100 iterations. For the MIP model, $t(s)$, represents the wall time that the algorithm requires to prove optimality. This column is left empty if the algorithm does not prove optimality. The CP model was not able to prove optimality for any of the instances and hence no $t(s)$ is provided for that algorithm. Finally $lazy$ represents the number of lazy-constraints that were added in the MIP approach.

Discussion

The CP solutions are in general the best compared to the other three approaches. The MIP model is competitive for the smaller instances, and is even able to prove optimality for instances (1-3). However as the size of the instances scales up the MIP model cannot find feasible solutions. The performance of the CP model is highlighted by instance 10, where the solution of 3077 corresponds to a gap of 8.19% with respect to the initial lower bound of 2833. This contrasts to best solution obtained by the heuristics of 5166, which corresponds to a gap of 81.6%. However as the size of the problem increases the CP model fails to find feasible solutions.

For all instances the heuristics are able to provide a significant improvement on the upper bound determined by the canonical schedules. This is particularly true for the larger instances, for example consider instance 18, compared to the canonical schedule value of 420748, the best SSGS solution of 42571 and the best JIH solution of 41813 correspond to a 89.8% and 90.1% reduction respectively. Furthermore both of these heuristics are noticeably fast. This is particularly true for the JIH, of which the largest average run time across all the instances is only 0.111 seconds. For all except two instances (6 and 10) the best solution obtain by the JIH outperformed that of the SSGS. Also in general the JIH is typically faster than the SSGS. This becomes noticeable for the largest set of instances, where the SSGS takes approximately 5 seconds whereas the JIH takes significantly less.

Interestingly, the best solution obtained by both SSGS and JIH outperforms the solution obtained by the CP model for instance 18. For this instance, the CP model required 428.75 seconds to just find the initial feasible solution of $62,719$, which it was then able to improve to $49,639$ with the remaining time. This suggests that perhaps the CP model would benefit from having the heuristic solutions seeded as an initial feasible solution.

Conclusion and future research

The LHRSP demonstrates the benefits that can be made by modelling the full problem, as opposed to making simplifying assumptions. By giving the CP and MIP models freedom to transfer multiple units of chemical at a time, better solutions are found for small instances. On the other hand, as the size of the problem grows, these approaches fail to find feasible solutions. In contrast, the heuristic methods are able to produce fast, feasible solutions to the simplified problem for all instances. Future research includes incorporating additional real-world complexities, such as additional types of resources, the manual mixing of chemicals by the pipette to produce new chemicals, job placement, and pipette cleaning. Finally modelling the problem as a temporal planning problem and evaluating the effectiveness of current temporal planners could provide an effective alternative to the proposed MIP and CP models.

Acknowledgements

This research was supported by the Australian Research Council under grant LP140101063

References

Artigues, C.; Huguet, M.-J.; and Lopez, P. 2011. Generalized disjunctive constraint propagation for solving the job shop problem with time lags. *Engineering Applications of Artificial Intelligence* 24(2):220–231.

Brucker, P.; Drexl, A.; Möhring, R.; Neumann, K.; and Pesch, E. 1999. Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research* 112(1):3–41.

Brucker, P.; Hilbig, T.; and Hurink, J. 1999. A branch and bound algorithm for a single-machine scheduling problem with positive and negative time-lags. *Discrete Applied Mathematics* 94(1-3):77–99.

Caumond, A.; Lacomme, P.; and Tchernev, N. 2008. A memetic algorithm for the job-shop with time-lags. *Computers and Operations Research* 35(7):2331–2356.

Dohn, A.; Rasmussen, M. S.; and Larsen, J. 2011. The vehicle routing problem with time windows and temporal dependencies. *Networks* 58(4):273–289.

Franck, B.; Neumann, K.; and Schwindt, C. 2001. Truncated branch-and-bound, schedule-construction, and schedule-improvement procedures for resource-constrained project scheduling. *OR-Spektrum* 23(3):297–324.

González, M. A.; Oddi, A.; Rasconi, R.; and Varela, R. 2015. Scatter search with path relinking for the job shop with time lags and setup times. *Computers & Operations Research* 60:37–54.

Kinable, J.; Wauters, T.; and Vanden Berghe, G. 2014. The concrete delivery problem. *Computers and Operations Research* 48:53–68.

Laborie, P., and Rogerie, J. 2008. Reasoning with Conditional Time-Intervals. *Proceedings of the Twenty-First International Florida Artificial Intelligence Research Society Conference, May 15-17, 2008, Coconut Grove, Florida, USA* 555–560.

Laborie, P. 2009. IBM ILOG CP Optimizer for detailed scheduling illustrated on three problems. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 5547 LNCS:148–162.

Manier, M. A., and Bloch, C. 2003. A classification for hoist scheduling problems. *International Journal of Flexible Manufacturing Systems* 15(1):37–55.

Rasmussen, M. S.; Justesen, T.; Dohn, A.; and Larsen, J. 2012. The Home Care Crew Scheduling Problem: Preference-based visit clustering and temporal dependencies. *European Journal of Operational Research* 219(3):598–610.

Riise, A.; Mannino, C.; and Burke, E. K. 2016. Modelling and solving generalised operational surgery scheduling problems. *Computers and Operations Research* 66:1–11.

Schutt, A.; Feydy, T.; Stuckey, P. J.; and Wallace, M. G. 2013. Solving RCPSP/max by lazy clause generation. *Journal of Scheduling* 16(3):273–289.

Vilim, P.; Laborie, P.; and Shaw, P. 2015. Failure-Directed Search for Constraint-Based Scheduling. *Integration of AI and OR Techniques in Constraint Programming: 12th International Conference, CPAIOR 2015, Barcelona, Spain, May 18-22, 2015, Proceedings* 437–453.

Yu, W.; Hoogeveen, H.; and Lenstra, J. K. 2004. Minimizing makespan in a two machine flow shop with delays and unit time operations is NP-hard. *Journal of Scheduling* 7(5):333–348.

What can Automated Planning do for Intelligent Tutoring Systems?

Sachin Grover* and **Tathagata Chakraborti*** and **Subbarao Kambhampati**
Arizona State University, Tempe, AZ 85281 USA
{ sgrover6, tchakra2, rao } @ asu.edu

Abstract

In this paper, we build on the latest in automated planning techniques to develop a generalized framework for course-independent design of Intelligent Tutoring Systems (ITSs). This is meant to provide targeted and personalized assistance to students, in order to meet the demands of the increasing class size, as well as help instructors who can use higher level specifications to design courses without having to worry about building the course-specific tutoring assistance. Thus the aim of this paper is to demonstrate what automated planning can bring to the table for the design of course-independent ITS features. We will illustrate these capabilities in **Dragoon**, an ITS deployed at Arizona State University.

1 Introduction

While the last decade has seen massive advances in technologies aimed at creation and dissemination of knowledge across a variety of platforms, concerns remain as to how effectively this knowledge is absorbed at the user (student) end. This is especially true for both massive open online courses (MOOCs) and also for (rapidly growing sizes of) physical classrooms where targeted attention towards individual students is often hard to provide. The state-of-the-art in student and instructor support technology has traditionally struggled to catch up with the demands of the rapidly evolving landscape of education in the 21^{st} century. In this paper, we address this by proposing a framework for the design of generic course-independent student and instructor support capabilities using techniques in the field of human-aware planning, and demonstrate those features in **Dragoon**, a celebrated intelligent tutoring system.

1.1 Learning 2.0

The world of learning is indeed changing fast - information can now be provided across a variety of platforms to large groups of people who can access *on demand* knowledge and participate in the learning process as a *community*. This is the Learning 2.0 paradigm (Seely Brown and Adler 2008), and requires a rethink of the affordances (McLoughlin and Lee 2007) expected from current learning tools.

Learning on Demand Learning on demand refers to the increasing popularity of individual student-centric and topic-driven learning achieved on the web – i.e. students pick a particular topic they want to learn about and actively consume content just based on that, instead of participating in an entire class or following through an entire curriculum. For example, consider that you want to learn about regression – you could log on to Coursera, complete the relevant tutorials and assignments on regression, and leave the course. This requires a rethink of traditional curriculum generation and course recommendation approaches that would traditionally compute end to end curricula for an entire class. It follows that such new approaches must be able to leverage detailed student models to provide effective support.

Social Learning One of the many advantages of social platforms for learning is peer feedback and community participation – i.e. social learning (Burke 2011). This involves two critical aspects – *knowledge advancement as a community* (Scardamalia and Bereiter 2006) and *information processing* (Webb 2013) on the part of the individual student as a member of that community. In a sense, this can even be seen as a proxy towards providing individual classroom attention from the instructor. However, forming study partners remains an arduous task, especially in large classrooms such as in online learning communities where students usually do not know most of their classmates (or their skill sets). It is also fraught with the usual pitfalls associated with group work including individual students hogging all the group activity or slackers not contributing to the group activity at all (Mesch 1991). Without principled drivers for building in-class communities that can promote learning, effective collaborations are hard to achieve. As such, forming useful teams for collaborative study can become a problem by itself rather than a facilitator for learning to the extent that students can end up spending too much effort in forming and maintaining teams or just prefer to study by themselves, thus leaving the potential benefits of a social learning environment largely untapped. Recent work has shown that peer recommendations can have positive impact (Labarthe et al. 2016) on student engagement but has remained ambiguous (Bouchet et al. 2017) as to the best way to go about it.

1.2 A Brief History of ITS and AI

ITSs are aimed to provide personalized support to students and bring in expert (human) tutors in the loop wherever necessary, thus reducing the burden on the instructor as well as improving the learning experience of the student. In fact, it has been shown that when designed correctly, an ITS can

*Authors marked with * contributed equally.

be as effective as a human teacher (VanLehn 2011). A thorough description of the different components of ITSs can be found in (Vanlehn 2006). Existing applications of such systems range from solving numerical problems like Andes (Gertner and VanLehn 2000) which can help in teaching basic laws of physics (Schulze et al. 2000), Dragoon (VanLehn et al. 2017), Q&A type problems as in Autotutor (Graesser et al. 2005) or for an SQL tutor (Mitrovic 2003). ITSs, of course, go beyond individual information processing stage and find uses in knowledge building as a community (Magnisalis, Demetriadis, and Karakostas 2011) as well, thereby embracing the principles of the Learning 2.0 paradigm.

Student Assessment Models One of the most important capabilities an ITS needs to have is to be able to estimate the (mental) model or capabilities of the student. This has been explored in the context of the (1) *item response theory (IRT)* (Hambleton, Swaminathan, and Rogers 1991) which treats learning and testing as separate processes and the (2) *Bayesian knowledge tracing (BKT) theory* (Corbett and Anderson 1994) which considers a more dynamic model of the student state. The latter becomes more relevant in the context of ITSs that can provide more dynamic feedback and hints as discussed next. Indeed this is an issue where AI techniques have been deployed before for dynamic modeling of the evolution of the student model in terms of knowledge components, concentration / focus levels, etc. (Murray, VanLehn, and Mostow 2004). This includes different techniques such as *decision theoretic approaches* (i.e. Markov Decision Processes or MDPs) (Murray, VanLehn, and Mostow 2004; Murray and VanLehn 2006), and *reinforcement learning* (Chi, VanLehn, and Litman 2010; Mandel et al. 2014; Mandel 2017). This paper assumes for the most part[1] that these techniques are available and builds on top of that assumption, i.e. being able to estimate the student model is necessary for ITS techniques and we want to demonstrate, from the perspective of automated planning how this can be exploited to provide a better learning experience to a student.

Feedbacks and Hints Once the ITS has estimated a model of the student, it can provide targeted feedback to improve the learning process. Existing work in this area (Barnes and Stamper 2010; Stamper et al. 2013; Rivers and Koedinger 2013; 2017) has largely focused on ITSs operating as *recommender systems*. This paper is largely situated in this space but aimed at providing much more sophisticated feedback in both the inner and outer loops (Vanlehn 2006) of an ITS which requires longer-term sequential reasoning.

1.3 What can planning bring to the table?

Automated planning, as a field, has been around ever since the inception of AI, and is considered a necessary ability of any autonomous system – the ability to reason about and decide on a course of action (CoA) or *plan* given the current state of the world. Many of the challenges faced in the design of an ITS bears parallels to the planning agenda – making a curriculum, solving a given problem, or in general dealing with the combinatorics of orchestrating a class can be potentially seen through the lens of planning, i.e. computing a sequence of steps given a set of constraints. This was the starting point of our investigation in this direction.

However, when operating with humans in the loop, traditional planning techniques are not sufficient (Kambhampati and Talamadupula 2015). A "human-aware" planner must be able to take into account the (mental) model (Chakraborti et al. 2017a) of the user. Recent work (Sengupta et al. 2017) has looked at how planning techniques can evolve in the context of *decision support* to guide the planning process of a *human* decision-maker. This includes support for plan validation, critiquing, recommendation, explanations, and so on. Much of the discussion here derives inspiration from recent advances in the planning community along these directions.

Contributions Thus, to answer the question what automated planning can do for the ITS scene, we build on the following two features of planning techniques –

- *Domain independence* – Planning techniques have been particularly geared towards domain-independent solutions – i.e. algorithms that can work across a variety of domains provided in higher-level specification. This is especially useful in the contexts of ITSs which have traditionally been restricted to class or course specific solutions that do not generalize; and
- *Model-based reasoning* – Personalized support for students require higher level and sequential reasoning about the course and *student models*, planning techniques remain ideally suited for this.

In this paper, we expound on the above two themes to –

- Provide targeted feedback when students are stuck on problems by leveraging the student model; (Section 3.2)
- Compute *on demand* curriculum based on class materials requested by the student; (Section 3.3)
  - We will show how this technique can be used to teach concepts to a student to attain different levels of expertise as desired by the student; and
  - We will show how student models may be composed to form joint plans of study.
- Generate class curriculum in the spirit of social learning by including fellow classmates in a student's curriculum while also guaranteeing desired properties of the curriculum – e.g. that students not only learn but also apply all the concepts at least once. (Section 3.4)

We do not, of course, set out to model the full scope of challenges[2] in building and end-to-end ITS. However, we recognize that much of the existing work on deploying ITS

[1] In fact, the "model reconciliation" technique discussed later can handle uncertain models (Sreedharan, Chakraborti, and Kambhampati 2018) and can even be modified to function as an estimator for the student model but this is outside the scope of the paper.

[2] For example, the current discussion only focuses on the learning and interaction phase and does not include post-hoc reflection / evaluations as explored in (Katz, O'Donnell, and Kay 2000; Katz, Allbritton, and Connelly 2003; Connelly and Katz 2009)

systems, if not in conceptualizing them, has focused on specific learning platforms or courses without any coherent approach or general principles of design and implementation of the roles usually attributed to ITSs. The aim of this paper is thus to introduce techniques from the planning community that can formalize some of these concepts and provide a generalized framework for building such systems from the ground up. This has useful implications for both the planning as well as the educational technologies communities – i.e. the former can provide solutions to existing problems in ITSs (as we demonstrate in this paper) while feedback form the learning community can provide useful feedback towards the refinement of said techniques, including defining new areas of research of mutual interest. The biggest advantage of such an approach, as mentioned above, is that the techniques are domain-independent, i.e. they are defined at the procedural level and can be grounded with the description of a particular course as specified by the instructor. Of course, the problem of knowledge representation is (for a specific course and assignments in it) remain a challenge, but the ITS features themselves generalize given the proposed planning framework.

2 Background

In the following, we will introduce concepts from the planning literature that will be used in the rest of the paper.

A Classical Planning Problem (CPP) (Russell and Norvig 2003) is the tuple $\mathcal{M} = \langle \mathcal{D}, \mathcal{I}, \mathcal{G} \rangle$ with domain $\mathcal{D} = \langle F, A \rangle$ - where F is a set of fluents that define a state $s \subseteq F$, and A is a set of actions - and initial and goal states $\mathcal{I}, \mathcal{G} \subseteq F$. Action $a \in A$ is a tuple $\langle c_a, pre(a), eff^{\pm}(a) \rangle$ where c_a is the cost, and $pre(a), eff^{\pm}(a) \subseteq F$ are the preconditions and add/delete effects, i.e. $\delta_{\mathcal{M}}(s,a) \models \perp$ *if* $s \not\models pre(a)$; *else* $\delta_{\mathcal{M}}(s,a) \models s \cup eff^{+}(a) \setminus eff^{-}(a)$ where $\delta_{\mathcal{M}}(\cdot)$ is the transition function. The cumulative transition function is $\delta_{\mathcal{M}}(s, \langle a_1, a_2, \ldots, a_n \rangle) = \delta_{\mathcal{M}}(\delta_{\mathcal{M}}(s, a_1), \langle a_2, \ldots, a_n \rangle)$.

A CPP is represented using the Planning Domain Definition Language or PDDL (McDermott et al. 1998).

A Plan Generator Module (PGM) (Helmert 2006) computes a solution to a CPP $\mathcal{M}$ as sequence of actions or a (satisficing) *plan* $\pi = \langle a_1, a_2, \ldots, a_n \rangle$ such that $\delta_{\mathcal{M}}(\mathcal{I}, \pi) \models \mathcal{G}$. The cost of π is $C(\pi, \mathcal{M}) = \sum_{a \in \pi} c_a$ if $\delta_{\mathcal{M}}(\mathcal{I}, \pi) \models \mathcal{G}$; ∞ otherwise. The cheapest plan $\pi^* = \arg\min_{\pi} C(\pi, \mathcal{M})$ is the optimal plan with cost $C^*_{\mathcal{M}}$.

A Plan Validation Module (PVM) (Howey, Long, and Fox 2004) outputs, given plan π and planning problem $\mathcal{M}$, `True` iff $\delta_{\mathcal{M}}(\mathcal{I}, \pi) \models \mathcal{G}$; `False` otherwise.

A Plan Recognition Module (PRM) (Ramírez and Geffner 2010) outputs, given a *partial plan* $\hat{\pi}$ and a planning problem $\mathcal{M}$, a plan π that maximizes the probability that $\hat{\pi}$ is a sub-plan of π –

$$\pi \leftarrow \arg\min_{\pi} \mathcal{P}([\hat{\pi}]_{k=0}^{k \leq |\pi|})$$

Note that the above approach does not directly compute this. Instead, we use the compilation approach from (Ramírez and Geffner 2009) to compute *the optimal plan that satisfies a set of observations given a goal* as the output of the PRM.

A Landmark Generation Module (LGM) (Hoffmann, Porteous, and Sebastia 2004) outputs, given a planning problem $\mathcal{M}$, a set of state (or action) landmarks $\mathcal{L}$ containing states (or actions) that must be passed through (or executed) in any satisficing solution of $\mathcal{M}$. Thus –

- An action landmark $a \in A$ requires that $a \in \pi$ $\forall \pi : \delta_{\mathcal{M}}(\mathcal{I}, \pi) \models \mathcal{G}$.
- A state landmark $s \subseteq F$ is such that $\forall \pi : \delta_{\mathcal{M}}(\mathcal{I}, \pi) \models \mathcal{G}$, $\exists [\hat{\pi}]_{k=0}^{k \leq |\pi|} : \delta_{\mathcal{M}}(\mathcal{I}, \hat{\pi}) \models s$. (Zhu and Givan 2003)

A Human-Aware Planning Problem (HAP) is given by the tuple $\Psi = \langle \mathcal{M}, \mathcal{M}^H \rangle$ where $\mathcal{M}^H = \langle D^H, \mathcal{I}^H, \mathcal{G}^H \rangle$ is the human's understanding of the planning problem $\mathcal{M}$ (Chakraborti et al. 2017a).

An Explicable Planning Module (EPM) computes a plan π such that it is a satisficing solution to $\mathcal{M}$ and is as close as possible to the expected plan in the human's model (Zhang et al. 2017; 2016; Kulkarni et al. 2016) –

$$C(\pi, \mathcal{M}) \approx C^*_{\mathcal{M}^H}$$

A Plan Explanation Module (PEM) outputs, given a HAP $\Psi = \langle \mathcal{M}, \mathcal{M}^H \rangle$ and the optimal solution π^* to $\mathcal{M}$, the *shortest* explanation (Chakraborti et al. 2017b) in the form of a model update to the human mental model $\mathcal{M}^H$ so that the same plan is now also optimal in the human's updated mental model $\widehat{\mathcal{M}^H}$ of the problem –

$$C(\pi^*, \widehat{\mathcal{M}^H}) = C^*_{\widehat{\mathcal{M}^H}}$$

The PEM can, in fact, trade off (Chakraborti, Sreedharan, and Kambhampati 2018) the relative cost of explicability (i.e. deviation from optimality in the planner's model) to the cost (i.e. length) of explanations during the plan generation process itself by computing a plan π and an explanation or model update $\mathcal{E}$ such that π is a solution to $\mathcal{M}$ and is the optimal solution to $\widehat{\mathcal{M}}$ modulated by a hyperparameter α –

$$\pi \leftarrow \arg\min_{\pi} \; |\mathcal{E}| \; + \; \alpha \times | \, C(\pi, \mathcal{M}) - C^*_{\mathcal{M}} \, |$$

With higher α, PEM computes plans that require more explanation, while with lower α, it generates more explicable plans. We refer to this variant as PEM(α).

Internally, PEM performs what is referred to as a *model space search* to come up with these explanations. This is done using *unit edit functions* λ that progressively try out one or more updates to the model $\mathcal{M}^H$ from the set of possible updates in $\mathcal{M}\Delta\mathcal{M}^H$ until the optimality conditions as described above are satisfied. This is known as the process of *model reconciliation* (Chakraborti et al. 2017b; Chakraborti, Sreedharan, and Kambhampati 2018).

3 ITS as Planning

We will now cast the design of a generic ITS in terms of the planning modules discussed in the previous section.

3.1 Class Configuration

A class configuration is defined as the tuple –

$$\mathcal{C} = \langle \{KC_i\}, \{T_i\}, \{A_i\}, \{S_i\} \rangle$$

- *Knowledge Components or Concepts:* $\{KC\}$ is a set of knowledge components or concepts KC_i. In ITS literature, the process of knowledge acquisition by a student has been decomposed into smaller components referred to as KCs (Koedinger, Corbett, and Perfetti 2010). KCs can be anything from a production rule (Mayer 1981), to a facet, misconception, fact or even a skill (Bloom, of College, and Examiners 1964). The aim of the social learning process is to make a student acquire different KCs based on their and their classmates already existing ones.
- *Tutorial:* The class also constitutes of a set $\{T_i\}$ of tutorials $T_i \subseteq \{KC_i\}$ that consist of a set of KCs on which they provide information on. These directly modify the student's knowledge state by providing information on specific topics or on how certain problems or (parts of) assignments may be solved. These form an integral part of a curriculum for the class.
- *Activities / Assignments:* The class also has a set $\{A_i\}$ of activities or assignments $A_i = \langle \mathcal{M}, \kappa \rangle$ where $\mathcal{M}$ is the model of the assignment and $\kappa \subseteq \{KC_i\}$ consists of a set of KCs that are required to solve it. These engage the student in actions that derive from knowledge introduced in the class (learning by doing). These form the core content of the class. Technically, these can also be used as sensing actions for the ITS in determining the knowledge state of the student. Thus, an assignment may be used both as a way of estimating the student model as well as a technique for imparting knowledge to the student.
- Finally, the class has a set $\{S_i\}$ of students S_i. The student knowledge state or *model* is defined as $S_i = \langle \{A_i^S\}, \kappa_1, \kappa_2 \rangle$ where A_i^S is the student's understanding (similar to the definition of a HAP) of the assignment model A_i and $\kappa_1, \kappa_2 \subseteq \{KC_i\}$ consists of a set of KCs that they have *learned* and *applied* respectively.

Given a class configuration $\mathcal{C}$, a curriculum is given by a sequence $c(\mathcal{C}) = \langle c_1, c_2, \ldots, c_n \rangle; c_i \in \{T_i\} \cup \{A_i\} \cup \{S_i\}$ of tutorials, assignments and partnerships with other students.

3.2 Tips and Hints

A solution to an assignment in a general sense can be seen as a sequence of steps, a.k.a. a *plan*. Thus, we posit that a large variety of assignments can in fact be modeled in terms of the planning problem. The model $A_i(\mathcal{M})$ of an assignment A_i (as mentioned before) is thus the model of a planning problem CPP. As explored in (Sengupta et al. 2017) in the context of decision support using automated planners, this opens up the slew of planning techniques (described in Section 2) that can be readily adopted to provide targeted (problem specific but domain independent) feedback to the students.

Solution Validation For a partial attempt (represented as a partial plan $\hat{\pi}$) on an assignment A_i, the Plan Validation Module (PVM) indicates conditions that were unsatisfied, which can be used to provide targeted feedback. For example, the PVM can be used by the instructor to auto-grade solutions proposed by a student, since this is a domain independent way of checking if the plan is a valid solution of the given assignment (represented as a CPP $A_i(\mathcal{M})$). This is also useful for the student as well who can receive immediate feedback on whether they are successful (and why, if not) without having to wait for the instructor. This is one of the features that most ITSs already possess. However, they are usually system level implementations that do not generalize across assignments.

Solution Completion For a partial attempt (represented as a partial plan $\hat{\pi}$) on an assignment A_i, the Plan Recognition Module (PRM) produces a completion that can be *sampled* from to provide hints that guide the student towards the full solution. The PRM thus allows the ITS to anticipate what actions the student needs to take given what they have already done in order to achieve their goal. Notice that the partial plan is generated by the student (from the model A_i^S) even though the completion is done using A_i. This can thus help the student in cases of cognitive overload, but not if they lack the knowledge to solve the problem, i.e. $A_i^S \neq A_i$. We will discuss ways to deal with the latter case in Sections 3.3.

The PRM module can be also used to provide *proactive support* by recognizing that the students is going astray and providing pop-ups to guide them towards the right solution. Proactive support and has been shown (Zhang et al. 2015; Sengupta et al. 2017; Chakraborti et al. 2017c) to be desirable of an artificial agent in collaborative settings. Interestingly, one could also imagine using the PRM to detect gaming of the tutoring system (Muldner et al. 2010) by defining it as a possible goal that a student might be trying to achieve, and based on the observations identify whether a student is working diligently or trying to game the system.

Problem Summarization Finally, the Landmark Generation Module (LGM) takes in the Classical Planning Problem (CPP) representation $A_i(\mathcal{M})$ for a specific assignment A_i and produces a set of steps (action landmarks) or situations (state landmarks) that the student *must* go through in order to solve the assignment. This can be very useful in providing a concise summary of "TODOs" required of the student to arrive at the solution, or by considering the domain variables that the student has already set to true, measure the progress of a student and thereby help the instructor in classroom orchestration (Dillenbourg et al. 2011).

We shall illustrate each of these use cases in Section 5.1.

3.3 On-demand Curriculum Generation

A typical feature of online learning, as we discussed in Section 1, is that students increasingly select a subset of class materials to follow and leave once they are done (e.g. MOOCs are known to have notoriously low completion rates (Amy Ahearn 2017)). As a result, students end up following individual and different curricula asynchronously. From the students' perspective an obvious problem with this is that they might not have the required knowledge to complete the materials they want. In the following, we thus address the problem of *on-demand curriculum generation*. In this paradigm, the student selects a particular assignment A_i to complete and the ITS performs argumentation with the assignment model $A_i(\mathcal{M})$ and the students model of the

assignment $A_i^S(\mathcal{M})$ to identify deficiencies in the student model that need to be addressed using relevant tutorials.

In order to achieve this, the ITS spawns an instance of the Plan Explanation Module (PEM) with the HAP $\Psi = \langle A_i(\mathcal{M}), A_i^S(\mathcal{M})\rangle$ – here the instructor model is the ground truth and the student model needs to be reconciled. The model edit functions λ are the tutorials in the class. The output of the PEM is thus the *optimal set of tutorials* (this forms the recommended curriculum) that guarantees that the same solution (plan) is optimal in both the student model as well as the instructor model (even though they are not equal). This is especially useful since the instructor model is going to contain information pertaining to the entire class, while the student does not need to know all these details in order to solve a specific assignment. The PEM is thus able to leverage the student and instructor models of an assignment to provide the exact set of tutorials that the student requires. We will provide illustrations of this process in Section 5.2. Notice that, the ITS can either use its estimate of A_i^S or engage in active information gathering by asking the student questions to determine parts of the student model it is uncertain about (Sreedharan, Chakraborti, and Kambhampati 2018), in order to meet the specific needs of the student.

Teaching as an α trade-off Notice that the formulation of the assignments as planning problems allow us to spawn CCPs with the student models (indicating how the student can solve the problem) or the instructor model (indicating how the instructor will solve the same problem) or anywhere in between (as computed by PEM(α)). The student solution (equivalent to an explicable plan) is likely to be suboptimal, or in most cases, not feasible in the ground truth or instructor model. An instantiation of PEM(α) with the HAP $\Psi = \langle A_i(\mathcal{M}), A_i^S(\mathcal{M})\rangle$ thus allows us to modulate the level of expertise with which a student wants to solve an assignment. For low values of α, the ITS will recommend the smallest possible curriculum that will just enable the student to solve the assignment (albeit suboptimally) while for progressively higher values of α it will start recommending more and more advanced curriculum to the point it matches the output of PEM, i.e. the optimal complete curriculum. From the perspective of the instructor as well, the α hyperparameter can be gradually increase from a low value to generate study materials for individual students as the course progresses. Thus the teaching process itself can be viewed through the lens of the model reconciliation process as one of modulation of the value of α in the PEM(α). We shall demonstrate this in Section 5.2.

Remark To the best of our knowledge, algorithms for the on-demand curriculum generation process driven by a specific class activity, and the argumentation process over the curriculum with the desired expertise level of student, have not been explored before in the ITS literature. This technique can be useful from the perspective of both the instructor and the student – e.g. the former can stagger the course content to meet the student's expertise level, while the latter can chose to learn at different levels of expertise (thus possibly reducing the high dropout rates that plague the on-demand learning communities).

Composition of Student Models Finally, we note that we can extend the model edit functions in the PEM from just the tutorials in the class to the other student models as well. Thus the model updates during the model reconciliation process can be affected by either the KCs provided by tutorial or a composition of one or more student models. The output of PEM will now provide an optimal recommendation of tutorials and potential study partners based on the skill sets (i.e. models) of the individual students.

3.4 The Jigsaw Problem

The Jigsaw Problem is the process of creating smaller groups in a class for cooperative learning (Aronson 1997). It has shown to have positive effect on students learning the course material together, and then engaging in discussions. This leads to a more active and deeper learning in class (Aronson 2011). Aronson, points out ten fixed steps to achieve this where the groups are created based on the ethnicity, race, gender and ability. However, it is intractable for a teacher to reason about all the student models and create study groups. Casting the class-level curriculum generation problem as a planning problem allows us to generate curricula for the entire class while enabling the instructor to specify desired properties of the curricula that needs to be maintained. These properties may be –

- Maximum size of study groups;
- Specific assignments of students;
- No repetition (or conversely, continuation) of study partners; and so on . . .
- In this paper, we specifically focus on the following property – *every student not only learns but applies all concepts in the class at least once*. This is especially important in the social learning paradigm, to ensure that students have mastered all concepts and not depended on other students to finish a shared curriculum.

In order to achieve this, we define a planning problem with the start state compiled from the class configuration $\mathcal{C}$ and a goal state that model a class configuration where $\forall S_i : S_i(\kappa_2) = \{KC_i\}$ – i.e. every student has applied all the concepts in class. The operators are generated from the set of tutorials and assignments – the tutorial operator has its associated KCs as effects of being learned; while assignment operators has KCs as preconditions (that need to be learned) and effects (of those KCs having been applied).

This formulation[3] thus not only ensures that all the students have mastered all the concepts in the class materials but also that the length of the curriculum is reduced (from $|\{S_i\}|$ times the length of the curriculum for individual students) due to the collaborations across students who can bring in complementary skill sets and transfer knowledge. We provide an illustration of this in Section 5.3.

[3]Note that this problem may be solved by horizon-limited planning, which is known to be NP-complete, the horizon being equal to $|\{S_i\}|$ times the length of the curriculum for individual students, which is the worst case curriculum length when no groups could be found. Thus, the jigsaw problem does not need the full expressiveness of CPP which is known to be PSPACE-complete.

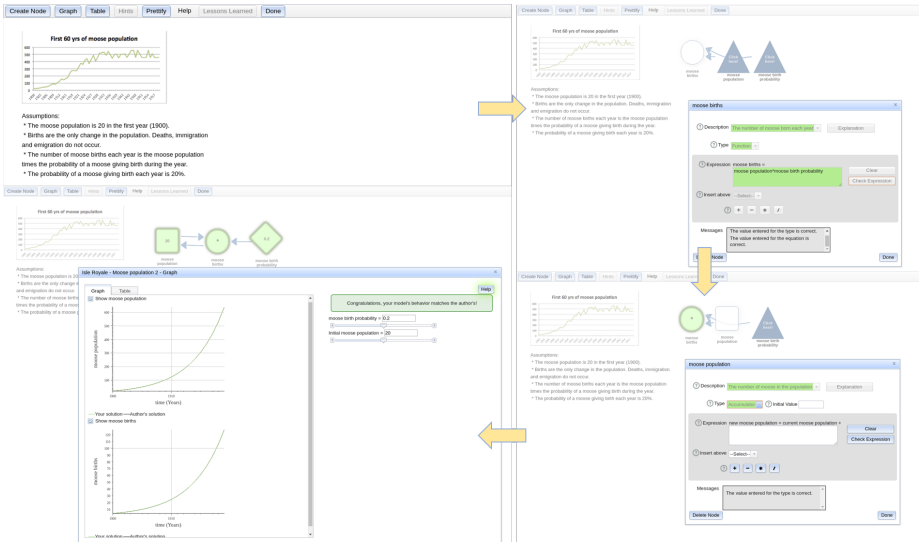


Figure 1: Illustration of the different stages of a "plan" being executed by a student in **Dragoon** – (1) the empty interface at the start of the problem (initial state); (2) the first node being completed; (2) the second node being created; and finally (3) the problem being completed with the feedback on the graph.

4 Introducing Dragoon

We will illustrate the above capabilities in **Dragoon** an ITS developed at Arizona State University to teach *dynamic system modeling* (VanLehn 2013) in the physical classroom setting – over the course of almost half a decade of deployment, the system has served 13 courses with approximate class sizes of 30, with more than a 1000 sessions per class. It is an ideal testbed for studying the nuances of tutoring systems currently deployed in classes in the space of mathematics, algebra and any other generic step-based tutoring systems. Figure 1 provides a snapshot of the interface.

In dynamic system modeling, a *system* is a part of the environment and *dynamic system* is the part of the environment that changes with time. Usually, first (or higher) order differential equations (differentiated with respect to time) represent dynamic systems mathematically. For simplicity of solving differential equations, time is discretized to calculate the values of different quantities. A *Model* refers to a representation of the system in a formal language.

Dragoon's formal language is based on Stella's stock and flow network (Doerr 1996). It consists of three different types of quantities – (1) **accumulator** (quantity that changes); (2) **function** (quantity that may or may not change); and (3) **parameter** (quantity that remains constant). These quantities are called *nodes*. To create a node a student needs to define its *properties* – i.e. description, type, value, units and equation. They are connected to each other by equations called *relations*. Students are taught template structure for interaction between nodes, which show particular rate of change in values called *schemas* – e.g. linear schema represents linear change in values while exponential schemas represent exponential changes. Students practice on **Dragoon** through tutorial and assignment workbooks. A detailed description of **Dragoon** is available at (Wetzel et al. 2017; VanLehn et al. 2016; 2017).

4.1 The Isle Royale Workbook

We use the Isle Royale Workbook (`https://goo.gl/ECrNnt`) to illustrate the proposed techniques. It teaches students population dynamics of moose and wolf population and learn interactions in a predator prey environment. There are six problems in the workbook (time step is a year) –

- **Isle-1** – Linear growth model of moose population, that is constant growth of two moose.
- **Isle-2** – Exponential growth model of moose population. The problem defines a constant growth rate which is multiplied by the population in the previous time-step to calculate the net growth.
- **Isle-3** – Exponential growth and death model of moose population. This problem adds the a constant death rate and the change in moose population is defined as the difference number of moose born and died.
- **Isle-4** – Exponential growth and death model of moose population with a fixed carrying capacity of the environment which effects the moose death rate.
- **Isle-5** – Exponential growth and death model of Wolf Population. This model is similar to Isle 3 problem.
- **Isle-6** – Exponential growth and death model of moose and wolf population with constant effect of wolf (predator) population on death rate of moose (prey) and constant effect of moose population on birth rate of wolf.

Epidemic schema is sometimes confused with exponential schema. Thus, we use one extra problem modeling flu epidemic in college which spreads through meetings between students. The number of students in the meeting and the chance that a student is affected is assumed to be constant.

The Zener Diode Problem Most problems in **Dragoon** are solved with a single or unique *set* of steps. The only

thing that changes is the sequence in which nodes are created. However, there are a few problems which can be solved in multiple ways, where a student can change the equations in the nodes to solve the problem in lesser number of nodes. One such problem is to model a Zener diode using **Dragoon** – if a student has a more advanced understanding of circuit theory, then they can easily solve the problem in fewer steps (i.e. using fewer nodes). We will thus use this problem to demonstrate the usefulness of PEM(α).

5 ITS as Planning in Action

We will now illustrate how the techniques introduced in Section 3 manifests themselves on **Dragoon**. The first step is to construct the instructor model $\mathcal{M}^I$ – examples can be accessed at – https://goo.gl/cyVthK.

We used nested object types to represent different objects in Dragoon, i.e. node, schema (KCs) and properties. Accumulator, parameter and function were of type node. Linear, exponential, extended_exponential, carrying_capacity and epidemic were types of schema. Description, value, type, equation and units are type of properties. These object types were used to define the state variables which characterize the properties that were part of a node, nodes that were part of schema, and schemas that were part of the problem. The operators in the domain represent the actions that are available student in the **Dragoon** environment. For example, a student fills each property to complete a node and it can be done in a fixed order. So the operator definitions were also related to initializing a node, filling every property of the node, completing a node and completing a schema. Students need an understanding of the schema to fill the type and equation of the node. Thus actions for those steps have a precondition of has_schema to create the node. Finally, the initial state consists of all the nodes and schemas that are part of the assignment as well as the knowledge state of the student, that is whether they understand the schemas required to solve the problem. The goal state required that the student complete all the schemas that are present in a given problem.

5.1 Tips and Hints (c.f. Section 3.2)

Plan Validation Figure 2, shows the 20-step solution for **Isle-2**, and Figure 1 shows some of these actions in the **Dragoon** environment. Figure 2 presents the incomplete attempt of the student being flagged as unsuccessful by the PVM, and shows the error generated after executing the incomplete plan in the **Dragoon** interface.

Plan Recognition Figure 3 shows the correct identification by the PRM among two possible solutions of the **Isle-3** assignment using the "exponential_growth" schema or the "exponential_decay" schema from partial observations of the actions of the student in **Dragoon**.

Landmarks Figure 4 shows the 35 state landmarks produced by the LGM for the **Isle-3** assignment.

5.2 On-demand Curriculum Generation (c.f. Section 3.3)

We use the same domain that we used in tips and hints. We are testing the case where a student wants to solve the **Isle-4** problem. Figure 5 shows the output of PEM when a student expresses a desire to complete the **Isle-4** assignment and requests a curriculum for it. The explanation presents the model differences in the initial state that prevents the student from completing the assignment at this time and suggests tutorials to introduce these concepts. The explanation is of size 3, and references the missing knowledge concepts that are needed for solving the problem in the 40 steps.

Figure 6 shows how PEM(α) can be used to modulate the expertise levels of the recommended curriculum. The complete curriculum is of size 3 after which the problem can be solved in 17 steps. But, with lower value of α, the problem can be solved with a longer 20 step plan. As explained earlier, even though the student needs two knowledge concepts to solve the problem (zener_voltage_regulator and kvl schema), but to solve the optimal plan a student needs to be an expert and improve the equations in one of the nodes and create a better model for Zener Diode problem.

5.3 Jigsaw Problem (c.f. Section 3.4)

Here, we took an instance of a **Dragoon** class with 7 concepts and 9 assignments. A single student curriculum comes out as 12 steps long, with 7 tutorials and 5 assignments. However, with the introduction of groups of two students, this reduces to a combined curriculum of 23 steps where every student applies every concept at least once. For every new student, plan size increases by 11 steps, showing that one of the assignment can be done in the group. This is shown in Figure 7, which plots the curriculum length with increasing class sizes. In this particular class configuration, only one of the assignments could be done in a group.

Now we study the effect of varying class configurations by the making assignments that randomly teach up to 4 concepts. The number of concepts were fixed to 10 and there were 20 assignments that would teach these concepts. Figure 8 shows the curriculum length for 50 different randomly generated four student class configurations. We observe a decrease of 3 to 7 steps in every class.

Conclusion and Work in Progress

In this paper, we demonstrated how an ITS framework can be built using the state-of-the-art in human-aware planning techniques for the design of course independent support features. The last section illustrated these properties in a real tutoring system **Dragoon**. Currently, we are working on integrating these features into **Dragoon** in order to perform ablation studies of the system with one or more of the support components deployed in a real class. We hope to report on those results in future iterations of the paper.

Acknowledgements. This research is supported in part by the AFOSR grant FA9550-18-1-0067, the ONR grants N00014-16-1-2892, N00014-13-1-0176, N00014-13-1-0519, N00014-15-1-2027, N00014-18-1-2442 and the NASA grant NNX17AD06G. The first author is also supported by the IBM Ph.D. Fellowship from 2016-18. The authors also thank Professors Kurt Van Lehn and Erin Walker (Arizona State University) for their valuable inputs.

```
(create_node moose_birth_probability)
(create_node moose_births)
(create_node moose_population)
(fill_description da moose_population)
(fill_type_single_schema ta da moose_population s1 exponential_growth)
(fill_equation_single_schema ea ta moose_population s1 exponential_growth)
(fill_units ua ta moose_population)
(fill_value va ta moose_population)
(complete_accumulator da ta va ua ea moose_population)
(fill_description df moose_births)
(fill_type_single_schema tf df moose_births s1 exponential_growth)
(fill_equation_single_schema ef tf moose_births s1 exponential_growth)
(fill_units uf tf moose_births)
(complete_function df tf uf ef moose_births)
(fill_description dp moose_birth_probability)
(fill_type_single_schema tp dp moose_birth_probability s1 exponential_growth)
(fill_units up tp moose_birth_probability)
(fill_value vp tp moose_birth_probability)
(complete_parameter dp tp vp up moose_birth_probability)
(complete_exponential_schema moose_birth_probability moose_births moose_population exponential_growth s1)
; cost = 20 (unit cost)
```

```
(create_node moose_birth_probability)
(create_node moose_births)
(create_node moose_population)
(fill_description da moose_population)
(fill_type_single_schema ta da moose_population s1 exponential_growth)
(fill_units ua ta moose_population)
(fill_value va ta moose_population)
(complete_accumulator da ta va ua ea moose_population)
(fill_description df moose_births)
(fill_type_single_schema tf df moose_births s1 exponential_growth)
(fill_equation_single_schema ef tf moose_births s1 exponential_growth)
(fill_units uf tf moose_births)
(complete_function df tf uf ef moose_births)
(fill_description dp moose_birth_probability)
(fill_type_single_schema tp dp moose_birth_probability s1 exponential_growth)
(fill_units up tp moose_birth_probability)
(complete_parameter dp tp vp up moose_birth_probability)
(complete_exponential_schema moose_birth_probability moose_births moose_population exponential_growth s1)
; cost = 18 (unit cost)

Plan failed to execute

Plan Repair Advice:

(complete_accumulator da ta va ua ea moose_population) has an unsatisfied precondition at time 8
(Set (is_filled ea moose_population) to true)

Successful plans:
Value: 20
 ../fast-downward/isle2_complete_plan 20


Failed plans:
 ../fast-downward/isle2_incomplete_plan
```

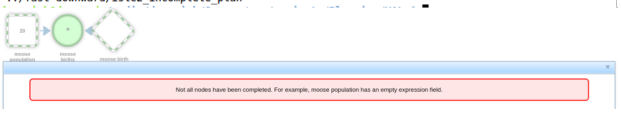
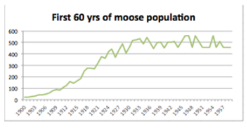
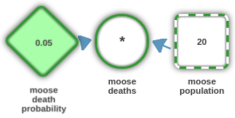


Figure 2: Response of PVM to the correct and incorrect or incomplete attempts in the **Isle-3** problem.

```
Experiment=../test-pr2/test-dragoon/test-dragoon.tar.bz2
Num_Hyp=2
Hyp_Atoms=(applied_exponential_schema moose_birth_probability moose_births moose_population)
Hyp_Test_Failed=True
Hyp_Cost_O=10000000.000000
Hyp_Cost_Not_O=10000000.000000
Hyp_Prob_O=0.500000
Hyp_Prob_Not_O=0.500000
Hyp_Plan_Time_O=0.030000
Hyp_Plan_Time_Not_O=0.040000
Hyp_Trans_Time=0.050000
Hyp_Plan_Time=0.070000
Hyp_Test_Time=0.120000
Hyp_Is_True=False
Hyp_Atoms=(applied_exponential_schema moose_death_probability moose_deaths moose_population)
Hyp_Test_Failed=True
Hyp_Cost_O=10000000.000000
Hyp_Cost_Not_O=10000000.000000
Hyp_Prob_O=0.500000
Hyp_Prob_Not_O=0.500000
Hyp_Plan_Time_O=0.020000
Hyp_Plan_Time_Not_O=0.040000
Hyp_Trans_Time=0.030000
Hyp_Plan_Time=0.060000
Hyp_Test_Time=0.090000
Hyp_Is_True=True
```

Create Node | Graph | Table | Hints | Prettify | Help | Lessons Learned | Done





Assumptions:
* The moose population is 20 in the first year (1900).
* Births and deaths are the only change in the population.
Immigration and emigration do not occur.
* The probability of a moose giving birth each year is 20%.
* The probability of a moose dying each year is 5%.

Figure 3: The output of the PRM in the **Isle-3** problem which can be solved in two separate ways. Here the student seemed to have decided to work on the exponential_decay schema.

```
34 Atom applied_exponential_schema(moose_birth_probability, moose_births, moose_population)
33 Atom applied_exponential_schema(moose_death_probability, moose_deaths, moose_population)
8 Atom applied_schema(exponential_decay)
17 Atom applied_schema(exponential_growth)
1 Atom is_complete(moose_birth_probability)
18 Atom is_filled(up1, moose_birth_probability)
4 Atom is_filled(vp1, moose_birth_probability)
19 Atom is_filled(tp1, moose_birth_probability)
21 Atom is_filled(dp1, moose_birth_probability)
20 Atom init(moose_birth_probability)
6 Atom is_complete(moose_births)
5 Atom is_filled(ef1, moose_births)
3 Atom is_filled(uf1, moose_births)
7 Atom is_filled(tf1, moose_births)
24 Atom is_filled(df1, moose_births)
0 Atom init(moose_births)
31 Atom is_complete(moose_death_probability)
29 Atom is_filled(up2, moose_death_probability)
28 Atom is_filled(vp2, moose_death_probability)
13 Atom is_filled(tp2, moose_death_probability)
10 Atom is_filled(dp2, moose_death_probability)
9 Atom init(moose_death_probability)
30 Atom is_complete(moose_deaths)
26 Atom is_filled(ef2, moose_deaths)
11 Atom is_filled(uf2, moose_deaths)
32 Atom is_filled(tf2, moose_deaths)
27 Atom is_filled(df2, moose_deaths)
12 Atom init(moose_deaths)
14 Atom is_complete(moose_population)
2 Atom is_filled(ea, moose_population)
23 Atom is_filled(ua, moose_population)
22 Atom is_filled(va, moose_population)
25 Atom is_filled(ta, moose_population)
16 Atom is_filled(da, moose_population)
15 Atom init(moose_population)
```

Figure 4: The 35 state landmarks generated by the LGM for the **Isle-3** problem.

```
Explanation >> has-initial-state-has_schema s1 carrying_capacity
Explanation >> has-initial-state-has_schema s1 exponential_decay
Explanation >> has-initial-state-has_schema s1 exponential_growth
Explantion Size: 3
Total Time  24.517663002
```

Figure 5: On-demand curriculum generated by the PEM. This is the smallest change to the student model required to solve the **Isle-4** problem.

```
current explanation 2
['create_node current_thru_r'
'create_node v_across_load'
'create_node v_across_r1'
'fill_description df4 v_across_r1'
'fill_type_double_schema tf4 df4 v_across_r1 s1 kvl zener_voltage_regulator'
'fill_equation_double_schema ef4 tf4 v_across_r1 s1 kvl zener_voltage_regulator'
'fill_units uf4 tf4 v_across_r1'
'complete_function df4 tf4 uf4 ef4 v_across_r1'
'complete_kvl_schema_temp v_across_r1 kvl s1'
'fill_description df5 v_across_load'
'fill_type_single_schema tf5 df5 v_across_load s1 zener_voltage_regulator'
'fill_equation_single_schema ef5 tf5 v_across_load s1 zener_voltage_regulator'
'fill_units uf5 tf5 v_across_load'
'complete_function df5 tf5 uf5 ef5 v_across_load'
'fill_description df6 current_thru_r'
'fill_type_single_schema tf6 df6 current_thru_r s1 zener_voltage_regulator'
'fill_equation_single_schema ef6 tf6 current_thru_r s1 zener_voltage_regulator'
'fill_units uf6 tf6 current_thru_r'
'complete_function df6 tf6 uf6 ef6 current_thru_r'
'complete_zener_voltage_regulator_schema v_across_r1 v_across_load current_thru_r zener_voltage_regulator s1']

current explanation 3
['create_node current_thru_r'
'create_node v_across_load'
'create_node v_across_r1'
'fill_description df4 v_across_r1'
'fill_type_double_schema tf4 df4 v_across_r1 s1 kvl zener_voltage_regulator'
'fill_equation_double_schema ef4 tf4 v_across_r1 s1 kvl zener_voltage_regulator'
'fill_units uf4 tf4 v_across_r1'
'complete_function df4 tf4 uf4 ef4 v_across_r1'
'complete_kvl_schema_temp v_across_r1 kvl s1'
'fill_description df5 v_across_load'
'fill_type_single_schema tf5 df5 v_across_load s1 zener_voltage_regulator'
'fill_description df6 current_thru_r'
'fill_type_single_schema tf6 df6 current_thru_r s1 zener_voltage_regulator'
'fill_equation_single_schema_expert ef6 tf6 s1 zener_voltage_regulator'
'fill_units uf6 tf6 current_thru_r'
'complete_function df6 tf6 uf6 ef6 current_thru_r'
'complete_zener_voltage_regulator_schema v_across_r1 v_across_load current_thru_r zener_voltage_regulator s1']
```

Figure 6: Different plans and associated model updates generated by the PEM(α) based on the α-hyperparameter. For a high value of α the curriculum is of size 3 after which the problem can be solved in 17 steps. With a lower value of α, the problem can be solved with a longer 20 step plan.

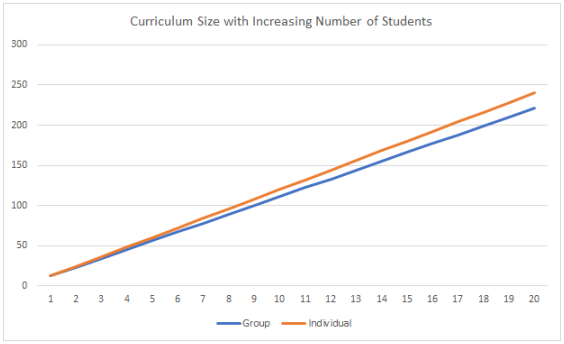


Figure 7: Group versus individual curriculum lengths with increasing class size.

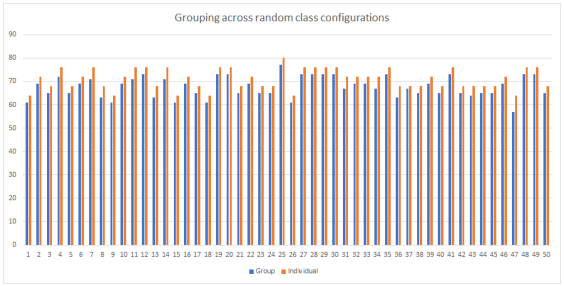


Figure 8: Group versus individual curriculum lengths in different class configurations.

References

Amy Ahearn. 2017. The Flip Side of Abysmal MOOC Completion Rates? Discovering the Most Tenacious Learners. `https://goo.gl/DR7nxa`. EdSurge.

Aronson, E. 1997. *The jigsaw classroom: Building cooperation in the classroom*. Scott Foresman & Company.

Aronson, E. 2011. *Cooperation in the classroom: The jigsaw method*. Printer & Martin Limited.

Barnes, T., and Stamper, J. 2010. Automatic hint generation for logic proof tutoring using historical data. *Journal of Educational Technology & Society* 13(1):3.

Bloom, B. S.; of College, C.; and Examiners, U. 1964. *Taxonomy of educational objectives*, volume 2. Longmans, Green New York.

Bouchet, F.; Labarthe, H.; Yacef, K.; and Bachelet, R. 2017. Comparing peer recommendation strategies in a mooc. In *Adjunct Publication of the 25th Conference on User Modeling, Adaptation and Personalization*, 129–134. ACM.

Burke, A. 2011. Group work: How to use groups effectively. *Journal of Effective Teaching* 11(2):87–95.

Chakraborti, T.; Kambhampati, S.; Scheutz, M.; and Zhang, Y. 2017a. AI Challenges in Human-Robot Cognitive Teaming. *arXiv preprint arXiv:1707.04775*.

Chakraborti, T.; Sreedharan, S.; Zhang, Y.; and Kambhampati, S. 2017b. Plan Explanations as Model Reconciliation: Moving Beyond Explanation as Soliloquy. In *IJCAI*.

Chakraborti, T.; Talamadupula, K.; Dholakia, M.; Srivastava, B.; Kephart, J. O.; and Bellamy, R. K. 2017c. Mr.Jones – Towards a Proactive Smart Room Orchestrator. In *AAAI Fall Symposium on Human-Agent Groups*.

Chakraborti, T.; Sreedharan, S.; and Kambhampati, S. 2018. Balancing Explicability and Explanation in Human-Aware Planning. In *AAMAS*.

Chi, M.; VanLehn, K.; and Litman, D. 2010. Do micro-level tutorial decisions matter: Applying reinforcement learning to induce pedagogical tutorial tactics. In *ITS*, 224–234.

Connelly, J., and Katz, S. 2009. Toward more robust learning of physics via reflective dialogue extensions. In *EdMedia: World Conference on Educational Media and Technology*, 1946–1951. Association for the Advancement of Computing in Education (AACE).

Corbett, A. T., and Anderson, J. R. 1994. Knowledge tracing: Modeling the acquisition of procedural knowledge. *User modeling and user-adapted interaction* 4(4):253–278.

Dillenbourg, P.; Zufferey, G.; Alavi, H.; Jermann, P.; Do-Lenh, S.; Bonnard, Q.; Cuendet, S.; and Kaplan, F. 2011. Classroom orchestration: The third circle of usability. *CSCL2011 Proceedings* 1:510–517.

Doerr, H. M. 1996. Stella ten years later: A review of the literature. *International Journal of Computers for Mathematical Learning* 1(2):201–224.

Gertner, A. S., and VanLehn, K. 2000. Andes: A coached problem solving environment for physics. In *International conference on intelligent tutoring systems*, 133–142.

Graesser, A. C.; Chipman, P.; Haynes, B. C.; and Olney, A. 2005. Autotutor: An intelligent tutoring system with mixed-initiative dialogue. *IEEE Transactions on Education*.

Hambleton, R. K.; Swaminathan, H.; and Rogers, H. J. 1991. *Fundamentals of item response theory*, volume 2. Sage.

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

Hoffmann, J.; Porteous, J.; and Sebastia, L. 2004. Ordered landmarks in planning. *JAIR*.

Howey, R.; Long, D.; and Fox, M. 2004. Val: Automatic plan validation, continuous effects and mixed initiative planning using pddl. In *Tools with Artificial Intelligence (ICTAI). 16th IEEE International Conference on*, 294–301. IEEE.

Kambhampati, S., and Talamadupula, K. 2015. Human-in-the-loop planning and decision support. *AAAI Tutorial*.

Katz, S.; Allbritton, D.; and Connelly, J. 2003. Going beyond the problem given: How human tutors use post-solution discussions to support transfer. *International Journal of Artificial Intelligence in Education* 13(1):79–116.

Katz, S.; O'Donnell, G.; and Kay, H. 2000. An approach to analyzing the role and structure of reflective dialogue. *International Journal of Artificial Intelligence in Education*.

Koedinger, K. R.; Corbett, A. T.; and Perfetti, C. 2010. The knowledge-learning-instruction (kli) framework: Toward bridging the science-practice chasm to enhance robust student learning. *Cognitive Science*.

Kulkarni, A.; Chakraborti, T.; Zha, Y.; Vadlamudi, S. G.; Zhang, Y.; and Kambhampati, S. 2016. Explicable Robot

Planning as Minimizing Distance from Expected Behavior. *CoRR* abs/1611.05497.

Labarthe, H.; Bouchet, F.; Bachelet, R.; and Yacef, K. 2016. Does a peer recommender foster students' engagement in moocs? In *9th International Conference on Educational Data Mining*, 418–423.

Magnisalis, I.; Demetriadis, S.; and Karakostas, A. 2011. Adaptive and intelligent systems for collaborative learning support: A review of the field. *IEEE transactions on Learning Technologies* 4(1):5–20.

Mandel, T.; Liu, Y.-E.; Levine, S.; Brunskill, E.; and Popovic, Z. 2014. Offline policy evaluation across representations with applications to educational games. In *AAMAS*.

Mandel, T. 2017. Refraction: Teaching Fractions through Gameplay. `https://goo.gl/BrPpmV`.

Mayer, R. E. 1981. Frequency norms and structural analysis of algebra story problems into families, categories, and templates. *Instructional Science* 10(2):135–175.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL-the planning domain definition language.

McLoughlin, C., and Lee, M. J. 2007. Social software and participatory learning: Pedagogical choices with technology affordances in the web 2.0 era. In *Proceedings of ICT: Providing choices for learners and learning*, 664–675.

Mesch, D. J. 1991. The jigsaw technique: A way to establish individual accountability in group work. *Journal of Management Education* 15(3):355–358.

Mitrovic, A. 2003. An intelligent SQL tutor on the web. *International Journal of Artificial Intelligence in Education*.

Muldner, K.; Burleson, W.; Van de Sande, B.; and VanLehn, K. 2010. An analysis of gaming behaviors in an intelligent tutoring system. In *International Conference on Intelligent Tutoring Systems*, 184–193. Springer.

Murray, R., and VanLehn, K. 2006. A comparison of decision-theoretic, fixed-policy and random tutorial action selection. In *Intelligent Tutoring Systems*, 114–123.

Murray, R. C.; VanLehn, K.; and Mostow, J. 2004. Looking ahead to select tutorial actions: A decision-theoretic approach. *International Journal of Artificial Intelligence in Education* 14(3, 4):235–278.

Ramírez, M., and Geffner, H. 2009. Plan recognition as planning. In *IJCAI*.

Ramírez, M., and Geffner, H. 2010. Probabilistic plan recognition using off-the-shelf classical planners. In *AAAI*.

Rivers, K., and Koedinger, K. R. 2013. Automatic generation of programming feedback: A data-driven approach. In *The First Workshop on AI-supported Education for Computer Science (AIEDCS 2013)*, volume 50.

Rivers, K., and Koedinger, K. R. 2017. Data-driven hint generation in vast solution spaces: a self-improving python programming tutor. *International Journal of Artificial Intelligence in Education* 27(1):37–64.

Russell, S., and Norvig, P. 2003. *Artificial intelligence: a modern approach*. Prentice Hall.

Scardamalia, M., and Bereiter, C. 2006. Knowledge building: Theory, pedagogy, and technology. *The Cambridge Handbook of the Learning Sciences* 97–118.

Schulze, K. G.; Shelby, R. N.; Treacy, D. J.; Wintersgill, M.; VanLehn, K.; and Gertner, A. 2000. Andes: An active learning, intelligent tutoring system for newtonian physics. *THEMES in Education* 1(2):115–136.

Seely Brown, J., and Adler, R. 2008. Open education, the long tail, and learning 2.0. *Educause review* 43(1):16–20.

Sengupta, S.; Chakraborti, T.; Sreedharan, S.; and Kambhampati, S. 2017. RADAR – A Proactive Decision Support System for Human-in-the-Loop Planning. In *AAAI Fall Symposium on Human-Agent Groups*.

Sreedharan, S.; Chakraborti, T.; and Kambhampati, S. 2018. Handling model uncertainty and multiplicity in explanations via model reconciliation. In *ICAPS*.

Stamper, J.; Eagle, M.; Barnes, T.; and Croy, M. 2013. Experimental evaluation of automatic hint generation for a logic tutor. *International Journal of Artificial Intelligence in Education* 22(1-2):3–17.

VanLehn, K.; Chung, G.; Grover, S.; Madni, A.; and Wetzel, J. 2016. Learning science by constructing models: Can dragoon increase learning without increasing the time required? *International Journal of Artificial Intelligence in Education*.

VanLehn, K.; Wetzel, J.; Grover, S.; and van de Sande, B. 2017. Learning how to construct models of dynamic systems: An initial evaluation of the dragoon intelligent tutoring system. *IEEE Transactions on Learning Technologies*.

Vanlehn, K. 2006. The behavior of tutoring systems. *International Journal of Artificial Intelligence in Education*.

VanLehn, K. 2011. The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems. *Educational Psychologist* 46(4):197–221.

VanLehn, K. 2013. Model construction as a learning activity: A design space and review. *Interactive Learning Environments* 21(4):371–413.

Webb, N. M. 2013. Information processing approaches to collaborative learning. *Routledge Handbooks Online*.

Wetzel, J.; VanLehn, K.; Butler, D.; Chaudhari, P.; Desai, A.; Feng, J.; Grover, S.; Joiner, R.; Kong-Sivert, M.; Patade, V.; et al. 2017. The design and development of the dragoon intelligent tutoring system for model construction: Lessons learned. *Interactive Learning Environments* 25(3):361–381.

Zhang, Y.; Narayanan, V.; Chakraborty, T.; and Kambhampati, S. 2015. A human factors analysis of proactive assistance in human-robot teaming. In *IROS*.

Zhang, Y.; Sreedharan; Kulkarni, A.; Chakraborti, T.; Zhuo, H. H.; and Kambhampati, S. 2016. Plan Explicability for Robot Task Planning. In *RSS Workshop on Planning for Human-Robot Interaction*.

Zhang, Y.; Sreedharan; Kulkarni, A.; Chakraborti, T.; Zhuo, H. H.; and Kambhampati, S. 2017. Plan Explicability and Predictability for Robot Task Planning. In *ICRA*.

Zhu, L., and Givan, R. 2003. Landmark extraction via planning graph propagation. *ICAPS Doctoral Consortium*.

Automated Mission Task Scheduling in Marine Voyage Planning

T. R. Hammond

Defence Research and Development Canada Atlantic Research Centre
Tim.Hammond@drdc-rddc.gc.ca

Abstract

This paper is about how to support, with software, the planning of marine voyages that also have to accomplish a series of mission tasks (MTs). Tactical naval missions provide the prime example of such voyages, but civilian applications are by no means excluded. Specifically, the paper considers how to schedule MTs automatically, both in time and in space, along a proposed route for the ship, so as to obey all the pertinent MT constraints. These constraints can restrict both where and when each MT is done. It proposes a greedy heuristic, Prioritizing by Maximum Procrastination (PMP), for selecting an order in which to do the mission tasks. PMP maximizes the packing density of MTs. It proposes another heuristic, *scheduleTasks*, to assign a time and place to each MT, given a particular order for them. Used together, these could facilitate the investigation of alternate routes for a voyage and enhance plan agility in response to unforeseen events and setbacks.

Introduction

When planning naval missions at the tactical level, officers of the Royal Canadian Navy (RCN) typically consider three different courses of action (COAs) for accomplishing the mission goals. Each COA characterizes a different approach to achieving these aims (Bryant 2000, DND 2008, Bélanger 2006), so each could involve a different route for the RCN ships involved. This paper suggests a way that software could help RCN planners prepare multiple tactical COAs quickly.

A tactical naval mission usually involves a voyage to be undertaken by the planners' own ship, so a tactical COA should include a proposed track for that vessel. In addition, a tactical mission will usually call on the ship to accomplish various key tasks.

Naval ships engage in a broad range of activities whilst at sea. For instance, they launch their organic assets, fire their weapons and counter those of adversaries. Many of these activities require the ship to sail at a particular speed or with a particular orientation to the wind or waves (e.g., helicopter launch). Many require the ship to be in a particular location or at a particular range from other vessels (e.g., refueling at sea). Many are compromised by extreme weather conditions, even to the point of becoming impractical (e.g., small boat launch). An activity subject to these types of constraints is here called a mission task (MT).

At the core of a tactical naval COA, one can expect to see a track annotated with markers indicating the locations and times for all the key MTs (Hammond 2017). This paper is about facilitating the construction of that core part, on the assumption that the route (p) to be taken is already selected by the planners. The focus is on scheduling the MTs automatically, considering their constraints and dependencies. Given a maximum speed for the ship (v_{max}), a list (T) of MTs, a route (p, from arc lengths p_{start} to p_{end}) for the ship to take, a timespan for the mission (m_{start} to m_{end}) and a field of meteorological and oceanographic forecasts (W) spread over the mission area and timespan, the aim is to schedule all the tasks and transits between them in such a way as not to violate the MT constraints or exceed v_{max}.

While scheduling techniques are increasingly applied to marine voyage planning problems (see review in Christiansen et al. 2013) the current focus on MTs is novel. There are civilian applications, as operations like fishing, dredging, cable laying, or towing could all be called MTs. Thus, the problem considered is not solely a naval one.

Mission Tasks and Their Dependencies

From the scheduling perspective, the most important thing about a particular mission task (t) is its duration (t_d), measured in hours. In addition, MTs allow for an effective progress speed (t_s), in knots. The product of t_s and t_d would determine the distance covered along p during the task. It is assumed here that MTs must be accomplished sequentially; they cannot be done concurrently, often because they require things of the ship itself. Moreover, MTs cannot be interrupted for some period and resumed later. Af-

ter speed and duration, the most important MT characteristics are constraints. These come in several different types.

Precedence Constraints

In projects of all sorts, there are typically constraints to the order in which tasks can be completed. The most common and useful form these constraints take is defined by a finish-to-start relationship between two tasks (Hammond 2017). This relationship implies that one task must be completed before the other can start. When MT b must start after the end of MT a, this is indicated here as $a < b$.

Fixed Start Times

Tactical missions often include a few MTs that are already scheduled by higher command. Other MTs must not be scheduled in such a way as to conflict with these orders.

Spatial Constraints

A spatial MT constraint is always associated with a particular area, or set of areas, within which the MT must be accomplished. For example, to fire the ship's guns at a target on land, the ship must be within artillery range of it. While such constraint areas are most naturally defined as two-dimensional regions on the surface of the ocean, for scheduling purposes it is helpful to project them onto the route p. Doing so allows these constraints to be represented as a list of arc length intervals over which p is within the relevant areas. The algorithms presented below expect spatial constraints to be represented with such intervals, which are here called 'arc length constraints'.

Meteorological and Oceanographic Constraints

The fact that environmental conditions change with time poses a challenge to voyage planning. On the one hand, one cannot determine which forecasted conditions (in W) will be encountered along p until one has a schedule for the trip. On the other hand, one cannot properly schedule that trip without knowing what conditions will be encountered. One way to address this conundrum is to start by assuming that the projected conditions at the start of the mission, namely at time m_{start}, will remain unchanged.

Assuming fixed conditions allows any environmental constraints that limit where a particular MT (a) can be done to be converted into a set of arc length intervals on p, during which a can be accomplished. These arc length intervals can then be used, along with other relevant constraints, to schedule the tasks and the traversal of p.

Thereupon, the assumption of constant conditions can be relaxed, since the track just computed can be used to revise the conditions (from W) that will be encountered along p, yielding a new set of constraint intervals. These new intervals are then used to revise the schedule, and so on. Iterating back and forth in this manner, until the changes in the track schedule are acceptably small, will yield a schedule that accounts for changing environmental conditions. By employing such an iterative process, environmental constraints can be treated as arc length constraints.

Approach to the Problem

This paper assumes that RCN planners want to conserve fuel, in order to stay in the key mission space longer. Provided that the mission is not full of slack time (in which case it should be shortened or have more MTs added), a good way to conserve fuel is to minimize the maximum steaming speed between MTs (v_{top}), as fuel consumption per unit distance is generally proportional to the second power of sailing speed (Christiansen et al. 2013). Note that, unlike v_{top}, the average steaming speed (v_{req}) is fixed:

$v_{req} = (p_{end} - p_{start} - \sum_T t_s t_d)/(m_{end} - m_{start} - \sum_T t_d)$

Thus, the scheduling objective is to keep v_{top} as close to v_{req} as possible, while still accomplishing all the MTs.

The scheduling problem is divided into two parts: deciding on an order in which to do the MTs and then scheduling them in that order, both in time and in position along the route p. Both parts employ a *shift* operation to handle arc length constraints, so this will be introduced first.

The *shift* Operation

This section describes the *shift* operation with reference to Fig. 1, which has two panels separated by a horizontal line. These panels depict the distance covered along p during a particular task ($t_s \times t_d$) with a red segment and arc length constraints to that task with blue ones. The side labelled Start shows the situation before the *shift*, while the side labelled Finish gives the result. In each panel, the *extent* of the desired *shift* is indicated with a black segment, with vertical endpoints. This *extent* defines the minimum distance that the red interval will be shifted.

The idea of the *shift* operation is that the red segment is confined to lie above the blue ones. In the top panel, after sliding the red segment to the right by the specified *extent* amount, the shifted segment still lies above the blue ones, so that is the result. In the bottom panel, however, the initial shift would not place the red segment above the blue intervals. Thus, the red segment keeps sliding further to the right, until a blue interval long enough to accommodate it is found. If no suitable spot is available, *shift* returns *null*. Note that a *shift* to the left can be specified by a negative value for the *extent*, and this works as if in mirror image.

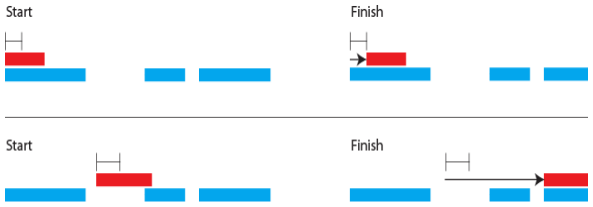


Fig. 1. This figure depicts the results of the shift *operation in two cases (top and bottom). In each, the location of the red segment before the* shift *is indicated at left and after at right.*

Choosing an MT Sequence

This section suggests an approach to choosing an order in which to perform the mission tasks. A bad order can not only induce unnecessary haste in transit, it can jeopardize the crew's ability to accomplish some of the mission tasks. This risk is reduced by maximizing the MT packing density, as doing so leaves as much route space and mission time available to remaining tasks as possible.

The approach follows a technique for dealing with precedence constraints suggested by Lawler (Lawler 1973). That technique builds the schedule back to front (from last MT to first), considering just the set of MTs that no others depend on (MT b depends on MT a, if $a < b$). Looking just over that set, the procedure chooses the MT that can be 'slid back furthest', both towards the end of p and towards the end time of the mission, and schedules that task (t_{max}) to be done last. Then it clips off p (by setting an arc length limit (p_{end})) at the (slid) start location of t_{max} and advances the mission end time (m_{end}) to the start time of t_{max}. Thereupon it simply repeats the process, considering at each step only the MTs that no other unscheduled MTs depend on. Repetitions continue until the MT to be done first is clear by elimination.

It remains to clarify just how 'slid back furthest' is measured, as there are a few details to this. Sliding mission tasks back like this seems a bit like procrastination. From this perspective, the strategy selects to do last the MT that permits the most procrastination, hence the name: Prioritizing by Maximum Procrastination (PMP).

The next few paragraphs add some precision to the concept of sliding MTs back, both in time and along p. In these paragraphs MTs are divided into two categories: those with starting time constraints and those without. Naturally, an MT (t) with constrained start time cannot be slid in time at all, but it can still be slid towards the end of p. Just how far depends on whether or not it is also spatially constrained.

If t does have arc length constraints, then the *shift* operator is used to determine how far this MT can be slid towards the end of p. The slight twist in this process is that the red interval (of Fig. 1), with length given by the product of t's duration and speed, is slid in from the right (using a negative *shift extent*). Also, the arc length constraints are limited (by intersection) to the interval from p_{start} to p_{end}. If the *shift* result is *null*, it is not possible to schedule the MTs in question. Note that the route will be clipped at the left endpoint of the *shift* result.

If t has a fixed start time but no arc length constraints, then how far it can be slid towards the end of p is determined by how far the ship can get when travelling from the start of p at v_{max} until the start time of t (naturally, it is also limited by the end of p). This calculation must account for the time spent doing the other as-yet unscheduled MTs and for the distance covered during them.

MTs without a fixed start time come in two types: with spatial constraints and without them. For the former type, the *shift* operator is used, once again, to find how far they can be slid towards the end of p. This works by sliding each task interval in from the right, just as above (also limit the arc length constraints to the interval from p_{start} to p_{end}). Denote the right limit of the *shift* result for MT t by pt_{end}. To find how long t can be delayed, compute the time it takes to get from pt_{end} to p_{end} at v_{max}. Subtract that time from the current mission end time m_{end}, yielding the end time t_{end} for t. Then, of course, the start time t_{start} is given by $t_{end} - t_d$. For the latter type (with no spatial constraints), note that these can be slid all the way to the end of p and all the way to the end time too.

Once it has been determined how far each MT can be slid, in both space and time, each is assigned a score. For MT t, this score depends on its right spatial endpoint pt_{end} and on its right temporal endpoint t_{end} after the slide. It is given by the product of $(pt_{end} - p_{start})/(p_{end} - p_{start})$ and $(t_{end} - m_{start})/(m_{end} - m_{start})$. This score gets larger, the closer the task gets to the end of p, but it is always below 1.

Normally, PMP will choose as t_{max} the MT with highest score, but there are two exceptions. The first exception clarifies that tied scores are broken by looking at the duration and length of each tied MT t. For MT t, the tiebreaking score is given by $1-(1-t_d/(m_{end} - m_{start}))(1-(t_d \times t_s)/(p_{end} - p_{start}))$, where again higher scores are treated as better. Managing MTs and their scores is facilitated by placing them in a priority queue data structure (*ScoreQ*), in which higher scores go first and the tiebreaking rule is considered. The second exception is a bit more complex.

A given MT (t), can only be chosen as t_{max}, if all the other as yet unscheduled MTs can be 'slid clear' of its start time and start position. An unallocated MT cannot be 'slid clear' of t, if either its earliest possible finish time is not before the latest start time (t_{start}) of t, or its first potential finish arc length position along p still is not less than the slid start arc length position (pt_{start}) of t. Thus 'slid back' and 'slid clear' have analogous meanings. Whenever an unscheduled MT is found that cannot be slid clear of t, PMP will move on to next MT in *ScoreQ*. If necessary, PMP will continue moving down the *ScoreQ* until an MT is found to meet the 'slid clear' condition. If none can be found, PMP will declare the mission to be impossible.

It can be shown that the runtime for PMP is $O(n^2 \log(n))$.

PMP with a Backtrack Stack

With minor modification, PMP can recover from order decisions that are subsequently found to fail, by recording key state variables on a backtrack stack object after every choice of t_{max}. The key objects to record are copies of the

ScoreQ, of which MTs are already scheduled, of which remain, and of p_{start}, p_{end}, m_{end}, and m_{start}. Recording a copy of the terminal MTs, which are the unallocated ones with no dependents, is also helpful. Whenever a given ordering choice is found to fail, popping these objects off the back track stack allows the algorithm to revise a previous decision and then resume scheduling from there, following the PMP heuristic. If necessary, such back tracking could run through all the possible MT orders, in order to find a feasible schedule. The worst case run time is O(n!), but scheduling failure is avoided. This method is denoted here by PMPwBTS.

Scheduling Tasks in a Specific Order

This section describes a recursive procedure (*scheduleTasks*) for tackling the simplified scheduling problem, where the order in which the MTs are to be completed has already been decided. When successful, the procedure returns a schedule for the mission in which the ship will not exceed v_{max} (the maximum ship speed, in knots) on any of the legs of the journey. Note that MT constraints can force the ship to go faster than v_{req} (the average steaming speed). The key idea is that repeated calls to *scheduleTasks*, with values of v_{max} set progressively closer to v_{req} by one knot (so long as there is continued success), provide a simple mechanism for determining how low the maximum speed v_{top} can get, to the nearest knot.

The arguments of *scheduleTasks* are as follows: v_{max}, an ordered sub list (*T2*) of *T* (as would be given by the output of PMP), a sub route of *p* (defined as an arc length interval from p_a to p_b (in nautical miles)), a timespan (m_a to m_b (in hours after m_{start})) contained within the original mission duration, and an integer named *adjustLevels* (which limits schedule adjustments).

Now *scheduleTasks* calls itself, so it must provide an escape from endless recursion. This exit is provided by empty task list situation. If the MT list *T2* is empty, *scheduleTasks,* checks whether the transit of p_a to p_b can be accomplished in the timespan from m_a to m_b without exceeding v_{max}. If so, it returns a schedule for the transit; if not, it returns an "insufficient time" error message.

When things go wrong, the *scheduleTasks* procedure indicates errors of two types: "insufficient time" and "insufficient space". The former type indicates that there is not enough time to accomplish all the desired activities, while the latter indicates that the arc length constraint to some task is not met.

When the MT list *T2* is not empty, the *scheduleTasks* procedure will try to schedule the very first mission task (*t*) in the list *T2*. It will do so using the *scheduleFirstTask* subroutine, described in more detail below. Note that it may prove impossible to schedule *t*, in which case *scheduleTasks* will terminate, passing on the error message from its subroutine. If *scheduleFirstTask* is successful, it will propose a schedule with the following parameters: let t_{start} to t_{end} be the proposed time interval for MT *t*, and let pt_{start} to pt_{end} be the corresponding arc length interval proposed for *t* on the route *p*. The key idea is that *scheduleTasks* will now call itself recursively twice, once for the transit before *t* and once for events after.

This paragraph provides more detail about the two recursive calls in *scheduleTasks*. The first of these is made with the following arguments: v_{max}, an empty MT list, an arc length interval from p_a to pt_{start}, a time interval from m_a to t_{start}, followed by the integer *adjustLevels*, which is not altered. The second recursive call has the following arguments: v_{max}, the remaining tasks after *t* (*T2* - *t*), the remaining route (pt_{end} to p_b), the remaining time interval (t_{end} to m_b). The integer *adjustLevels* is also passed on unaltered. If both calls return without error, then the routine will return the amalgamated schedules for before, during and after *t*. If there are errors, the routine does not give up, provided that *adjustLevels* > 0.

If either recursive call returns with an error message, then *scheduleTasks* will look at the value of *adjustLevels*. If this value is 0, the procedure gives up, returning an error message. If *adjustLevels* > 0, however, it will attempt to reschedule *t*, according to the particular error messages returned. This is accomplished with a subroutine called *adjustTaskSchedule*, which will reset t_{start}, t_{end}, pt_{start} and pt_{end}, as described in more detail below. If *t* can be rescheduled, the two recursive calls to *scheduleTasks* are repeated, as above, only with the newly revised values for t_{start}, t_{end}, pt_{start} and pt_{end} and with the *adjustLevels* value reduced by one. Thus, nested schedule adjustments are limited to the number recursion levels specified by *adjustLevels*. Attempted schedule adjustments continue until either no further adjustments are possible (a situation indicated by the return value of *adjustTaskSchedule*) or the two recursive calls both return without error. In the former case, *scheduleTasks* will return an amalgamated error message, in the latter an amalgamated schedule.

The *scheduleFirstTask* and *adjustTaskSchedule* subroutines will shortly be described in detail. Their description will complete the specification of *scheduleTasks*. Both subroutines rely on the *shift* operation to deal with spatial constraints.

Scheduling the First Mission Task

This section describes the *scheduleFirstTask* subroutine, which takes the following arguments: v_{max}, *T2*, p_a, p_b, m_a and m_b, all defined as above. Like *scheduleTasks*, it can return "insufficient time" and "insufficient space" errors. In addition, the subroutine sets four flags associated with *t;* these are called *canDelay*, *canAdvance*, *canPull,* and *can-*

Push. These flags constrain any future adjustments to the schedule for *t* that might be needed. The first, *canDelay*, indicates that, if need be, *t* could be delayed in time. The second indicates that *t* could occur earlier. The remaining two indicate whether *t* could be pulled forward or pushed back (respectively) along the route *p*.

To start off, *scheduleFirstTask* checks whether there is enough time to accomplish all the MTs in *T2* as well as make the transit from p_a to p_b. This is done by computing the travel speed required (v_{req}), after allowing for all the time spent on the MTs and for any progress along *p* made during them. If $v_{req} > v_{max}$, *scheduleFirstTask* will return an "insufficient time" error message. Otherwise, it will extract the very first task (*t*) from *T2*. The primary objective is to set the schedule variables for *t*, namely t_{start}, t_{end}, pt_{start} and pt_{end}, all defined as above.

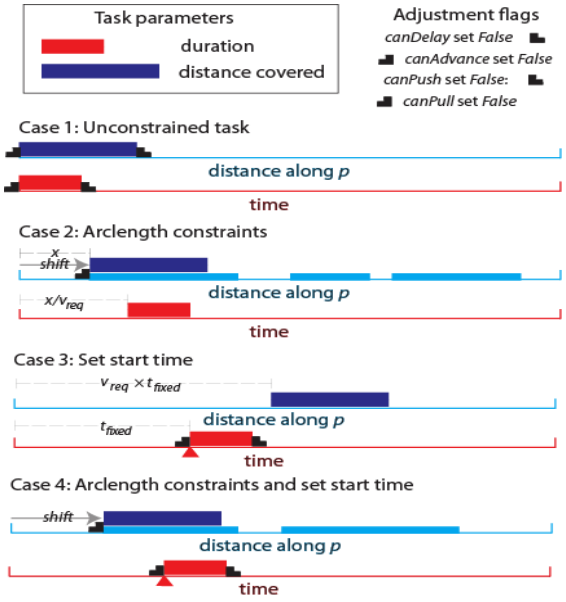


Fig. 2. This figure shows how the first MT is scheduled, in both time and space, in four cases given by its constraints. It also indicates how the adjustment flags are set in each case.

The schedule proposed for *t* and the settings for the *canDelay*, *canAdvance*, *canPull*, and *canPush* flags depend on the particular constraints of *t*, as illustrated in Fig. 2.

Rescheduling a Mission Task

Rescheduling MTs is the role of the *adjustTaskSchedule* subroutine, which also indicates, in its Boolean return value, whether or not adjustments were possible. Naturally, this subroutine needs to know which task *t* it is adjusting. It needs the latest schedule for *t*, which is defined by t_{start}, t_{end}, pt_{start} and pt_{end}. It needs the interval endpoints p_a, p_b, m_a and m_b. In addition, it needs to know the error messages from the two latest recursive calls to *scheduleTasks*. The idea is to use these messages and the flags (*canDelay*, *canAdvance*, *canPull*, and *canPush*) to guide the rescheduling effort.

The procedure also has some configuration parameters. Just how much the timing of *t* would be advanced or delayed in time is governed by a setting called *tby*. Just how much the position of *t* along *p* would be pushed or pulled is governed by another setting called *by*. Both settings are positive real numbers. The smaller these settings get, the greater the chance of finding a suitable scheduling result, at the cost of increased run time. Parameters *by* and *tby* can be regarded as the resolution of the resulting schedule. For example, if *tby* is 0.5, schedule adjustments smaller than 30 minutes will not be considered.

Temporal adjustments to the schedule for *t* are made by adding *tby* to t_{start}, in the case of a delay, and by subtracting *tby* from t_{start}, in the case of an advance. Then t_{end} is given by $t_{start} + t_d$. It is important to check that these new times are still in the interval from m_a to m_b. If not, *adjustTaskSchedule* will abort, indicating that no adjustments are possible. If the new schedule for *t* is still within m_a to m_b, then the routine returns with it, reporting success.

Spatial adjustments to the task schedule are accomplished using the *shift* operator. The starting interval (red in Fig. 1) in these *shift* calls is always pt_{start} to pt_{end}. If *t* has spatial constraint intervals, then the intersection of these with the arc length interval from p_a to p_b is used to constrain the shift (see the blue intervals in Fig. 1). If *t* has no spatial constraint, then the *shift* constraint is the entire interval from p_a to p_b. The *extent* of the *shift* is *+by*, when pushing the task back along *p*, and *–by*, when pulling the task forward. Note that *shift* results can be *null*, in which case *adjustTaskSchedule* will abort, indicating no adjustments are possible. If there is a valid *shift* interval result, denote its left endpoint by p_{first}. Set pt_{start} to p_{first} and pt_{end} to $p_{first} + t_d \times t_s$ and return with this new schedule, indicating success.

Whenever a schedule adjustment is made, whether spatial or temporal, the flag for the opposite adjustment is always set to *false*. For instance, if the task is delayed in time, then *canAdvance* is set to *false*. Conversely, if the task is advanced in time, then *canDelay* is set to *false*. This is to prevent *adjustTaskSchedule* from reversing its previous changes.

It remains to specify in what situations *t* will pushed, pulled, advanced or delayed, as well as when no further adjustments are possible. These situations are illustrated in Fig. 3. That figure shows MT duration in red and distance covered during the MT in dark blue.

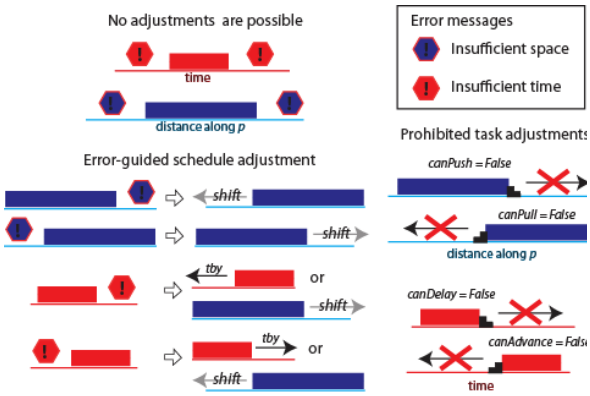


Fig. 3. At bottom left (BL), this figure shows how the possible error messages before and after a particular MT guide schedule adjustments. The legend at top right (TR) provides the error symbols. The requested adjustments (after the hollow arrow) are made, unless this is prevented by conditions at TL or BR.

Runtime

This section discusses the runtime of *scheduleTasks* as a function of n, the number of MTs. Much depends on the need to call *adjustTaskSchedule*. In the ideal case, when no adjustment is needed at all, *scheduleTasks* will run in $O(n^2)$. More generally, the finer the spatial and temporal resolution of the schedule (see *by* and *tby*), the more calls to *adjustTaskSchedule* will tend to be made. As a result, the worst case run time is $O(n^2(i_s+i_t)^{adjustLevels})$, where i_s is inversely proportional to *by* and i_t to *tby*. Note that *scheduleTasks* avoids run times that are exponential in n by limiting nested schedule adjustments. These may not occur on more recursion levels than given by *adjustLevels*.

Test Case Results

This section gives a few test case results, to suggest what can be expected from applying the heuristics. In all examples, the maximum ship speed v_{max} is 30 knots. In the first three test cases, the voyage lasts 24 hours and the route p has length 300 NM. Also, *adjustLevels* is set to 5.

Scenario 1

Suppose there are just two MTs. The first, A1, has duration 7.5 hours and allows progress on p at 10 knots. It has the following arc length constraints in NM: [[10.0, 80.0], [90.0, 180.0], [195.0, 285.0]]. The second, A2, lasts one hour, allows progress on p at 5 knots, but has to start exactly 10.51 hours into the mission. If the tasks are done in the order [A1, A2], *scheduleTasks* can produce a schedule, provided the configuration parameters are set suitably.

If the temporal resolution (*tby*) is not below about 0.02, *scheduleTasks* will suggest the mission above is impossible. Setting *tby* to 0.01 and *by* to 3, however, reveals that this suggestion is exaggerated. It also reveals the source of trouble: the ship is forced to go very fast, as indicated in Table 1.

Table 1. Scenario 1 schedule with MT order [A1, A2].

Mission Schedule			
Task	***Time window (h)***	***Arc length interval (NM)***	***Speed (KTS)***
Sail leg 0	[0.0, 3.0]	[0.0, 90.0]	29.99
A1	[3.0, 10.5]	[90.0, 165.0]	10.0
Sail leg 1	[10.5, 10.51]	[165.0, 165.09]	10.4
A2	[10.51, 11.51]	[165.09, 170.09]	5.0
Sail leg 2	[11.51, 24.0]	[170.09, 300.0]	10.4

Here, PMP would suggest doing A1 after A2, because A1 can be slid well past the fixed end time of A2. Running *scheduleTasks* with this suggested MT order results in the less hectic cruise indicated in Table 2. These tables illustrate how a poor choice of MT order can induce unnecessary haste in transit, and thus waste fuel.

Table 2. Scenario 1 schedule using the suggested MT order.

Mission Schedule			
Task	***Time window (h)***	***Arc length interval (NM)***	***Speed KTS***
Sail leg 0	[0.0, 10.51]	[0.0, 149.17]	14.19
A2	[10.51, 11.51]	[149.17, 154.17]	5.0
Sail leg 1	[11.51, 14.39]	[154.17, 195.0]	14.19
A1	[14.39, 21.89]	[195.0, 270.0]	10.0
Sail leg 2	[21.89, 24.0]	[270.0, 300.0]	14.19

Scenario 2

This mission has three MTs: B1, B2, and B3. All three last 5 hours. B3 allows progress on p at 5 knots, but the other two allow 10 knots. All have the same arc length constraints: [[30.0, 60.0], [80.0, 180.0]]. *scheduleTasks* claims that doing these MTs in the order [B1, B2, B3] is impossible. In this case, no amount of fiddling with settings will change the result because it's true. Here, PMP would suggest reverse order [B3, B2, B1], after invoking its score tiebreaker rules. Using that order, *tby* = 0.5 and *by* = 1.0, *scheduleTasks* yields the schedule in Table 3.

Table 3. Scenario 2 schedule using suggested MT order.

Mission Schedule			
Task	***Time window (h)***	***Arc length interval (NM)***	***Speed (KTS)***
Sail leg 0	[0.0, 1.54]	[0.0, 30.0]	19.44
B3	[1.54, 6.54]	[30.0, 55.0]	5.0
Sail leg 1	[6.54, 7.83]	[55.0, 80.0]	19.44
B2	[7.83, 12.83]	[80.0, 130.0]	10.0
B1	[12.83, 17.83]	[130.0, 180.0]	10.0
Sail leg 2	[17.83, 24.0]	[180.0, 300.0]	19.44

Scenario 3

This scenario illustrates the value of PMP's 'slid clear' provision. Let MT C1 have duration 5 hours, permit progress at 5 knots, and have arc length constraint [[240.0, 270.0]]. Let MT C2 have duration 10 hours, and permit progress at 5 knots, with no other constraints. C2, being unconstrained, can be slid back further along p than C1. Thus, considering the score alone would suggest the order [C1, C2], an order than cannot be made to work. The problem is that C1 cannot be 'slid clear' of the latest start position of C2. Thus, PMP would suggest the order [C2, C1], which is readily seen to work.

Scenario 4

For this scenario and the two simulation experiments that follow, the voyage was lengthened to 72 hours and the route to 900 NM.

PMP can fail, even though a valid schedule exists. Consider three MTs called F1, F2 and F3. Let F1 last 2 hours at speed 5 knots, start 31.8 hours in and be confined to the interval [150, 179.4]. Let F2 last 2.5 hours at a speed of 1 knot and be confined to [[230, 240], [270, 504]]. Let F3 last 3.5 hours at speed 3 knots and start 35 hours in. PMP will pick F3 to go last, which fails. In fact, the only way to accomplish this mission is in the order [F1, F3, F2].

Simulation Experiment 1

In the first experiment, several tactical missions were simulated, each with ten MTs. These MTs had randomly generated parameters and constraints. Each had a random duration in the interval from [0.5, 4.5] hours, and a random speed in the interval from [1.0, 6.0] knots. Each had a 50% chance of having a spatial constraint formed by the intersection of a random interval with [[30, 120], [150, 240], [270, 600], [660, 890]]. That random interval had minimum in [0, 300] and maximum in [300, 900]. Each had a 5% chance of having a fixed start time, selected at random in the [12, 60] hour interval. Each also had a 90% chance of having a precedence constraint, selected randomly from among the previously-generated MTs in the same mission.

Tactical missions were simulated until 100 missions were found to have a feasible schedule. Such a schedule was identified by trying *scheduleTasks* (with *adjustLevels =5*, *tby* = 0.25 and *by* = 0.5) on all the possible MT orders, until one was found to work. The MT order that produced the lowest value of v_{top} was denoted by *ExhaustiveSearch*.

PMP, PMPwBTS and a brute force approach called First2Work (which tries different task orders until one is found to produce a schedule) were all compared to the results of *ExhaustiveSearch*. The first three of these methods used *scheduleTasks*, with *adjustLevels =5*, *tby* = 0.5 and *by* = 1.0. Note that the last two values were double those used by *ExhaustiveSearch*.

On these simulated data, PMP had a success rate in finding a schedule of 97%, while the other methods were always successful. The average run time was 0.2 ms for PMP, 129.8 ms for PMPwBTS, and 953.3 ms for First2Work. The average top speed (v_{top}) for PMP (when successful) was 17.6 KTS and PMPwBTS produced identical results in those situations. Comparable results for First2Work were 18.3 KTS and 17.2 KTS for *ExhaustiveSearch*. Overall, PMPwBTS kept the top speed down to 17.9 KTS, whereas First2Work had 18.5 KTS, as compared to 17.5 KTS from *ExhaustiveSearch*. The smaller values of *by* and *tby* used in *ExhaustiveSearch*, as compared to PMPwBTS or First2Work, never prevented *scheduleTasks* from finding a valid schedule.

Simulation Experiment 2

The second experiment increased the number of simulated MTs in each mission to 40. In addition, the duration of each MT was fixed at 0.5 h and the speed during it was fixed at 10 KTS. The probability of having a fixed start time was also reduced to 2.5%. Otherwise the MT constraints were simulated as in the previous experiment.

Simulations were produced until there were 50 for which a valid schedule could be found by PMP and *scheduleTasks*, the latter with *adjustLevels =10*, *tby* = 0.5 and *by* = 1.0. The experiment looked at using different values of for *adjustLevels*, in the interval from 1 to 5, to see if lowering these values would change the schedule results.

In 4 cases out of 50, lowering *adjustLevels* to 1 prevented a schedule from being found. In a further six, that level changed the resulting schedule, leading to a higher v_{top}. In two cases, a value of 2 for *adjustLevels* similarly changed the schedule, increasing v_{top}. In no case did a value above 2 change the schedule result, compared to that from *adjustLevels =10*.

Conclusion

This paper described two heuristics for addressing a scheduling problem that arises in marine voyage planning. The situation to which they are applicable involves MTs. These can be regarded as tasks that interrupt the trip. The original inspiration for the problem was tactical naval planning, but other applications are possible.

Prioritizing by Maximum Procrastination (PMP) suggests an order in which the MTs should be done, given their temporal, precedence and arc length constraints. It handles the precedence constraints by building the schedule back to front. It works by repeatedly choosing to do last the as-yet-unscheduled MT that permits the most procrastination. It runs in $O(n^2\log(n))$, where n is the number of MTs. As a modified greedy algorithm, it is not guaranteed to produce a workable order, even when one exists (see Scenario 4). The results of simulation experiment 1, however, suggest that PMP is fairly robust (at least with 10 MTs), since it successfully produced a schedule in 97% of simulated missions. PMPwBTS can fill in where basic PMP fails, but at the cost of $O(n!)$ worst case run time (nothing close to this worst case arose in the experiment).

Using PMPwBTS yielded improved run time and lower values for maximum speed (v_{top}) than First2Work (which used a brute-force search over all possible MT orders, stopping at the first to yield success). PMP's advantage indicated that procrastination has more benefits than is commonly realized. Still, broad reliance on these benefits should await comparison of PMP with tools based on mixed integer or constraint programming (for example Laborie and Messaoudi 2017).

scheduleTasks determines a time and place to do each MT, given an order in which to do them. Its output looks like the tables above. As illustrated in Scenario 1, it is possible for *scheduleTasks* to fail to find a workable schedule, though one exists, because *by* or *tby* are not set low enough. Though such situations could arise in theory, in practice, the results of Simulation Experiment 1 suggest that *by* could be increased from 0.5 to 1.0 and *tby* from 0.25 to 0.5 with no effect on results. Results may thus be robust to small changes in these parameters.

The value of the *adjustLevels* parameter could also have an effect on the results of *scheduleTasks*. This parameter limits the number of recursion levels at which schedule adjustments are permitted. Simulation experiment 2 showed that schedule results are insensitive to values above 2, even with 40 MTs. Thus, using a value of 3 should be adequate in practice. This is important because the worst case run time of *scheduleTasks* is $O(n^2(i_s+i_t)^{adjustLevels})$.

Both heuristics assumed that meteorological constraints could be translated into constraints on arc length. This translation relies on an iterative procedure that has not been demonstrated in practice. Indeed, neither of the heuristics has been tested on real missions, so this work is at a preliminary stage. Still, on the example problems, the results suggest a potential for automating tasks that are currently done manually.

The two suggested heuristics open the door to optimization of the route p for various goals, total fuel consumption being but one possibility. One could optimize the route, while simultaneously ensuring all the MTs can be accomplished. Existing route suggestion algorithms for ships do not consider MTs (Hammond 2017), so the ideas suggested here raise new possibilities.

In the voyage planning context, the greatest value of automated scheduling may be revealed when plans do not go as expected. Setbacks, delays and unforeseen events are almost inevitable, especially in a military conflict. It will be essential to revise the plan, and how fast this can be done can be important in responding effectively. It is likely that the new plan will still need to achieve most, if not all, of the same MTs. To the extent that MTs and their dependencies remain relevant, automated scheduling has the potential to facilitate and speed up plan revision.

Acknowledgement

This work was funded by the Predictive Situational Awareness task of DRDC – Atlantic Research Centre's Integration of Command Decision Support project (01db).

References

Bélanger, M. 2006. The Estimate process: Observations, Technical Report, DRDC Valcartier TM 2003-357.

Bryant, D. 2000. Functional analysis of the Canadian Naval Task Group Operational Planning Process, Contract Report, DRDC-RDDC-2000-104.

Christiansen, M.; Fagerholt, K.; Nygreen, B. and Ronen, D. 2013. Ship Routing and Scheduling in the New Millennium. *European Journal of Operations Research*. 228(2013) 467-483.

Department of National Defence, 2008. The Canadian Forces Operations Planning Process (OPP): Change 2, In *Canadian Forces Joint Publication 5.0*, B-GL-300-003/FP-000.

Hammond, T. 2017. Current Practice in Mission Planning in the Canadian Navy and Opportunities for Automation, Technical Report, DRDC-RDDC-2017-R022.

Laborie, P. and Messaoudi, B. 2017. New Results for the GEO-CAPE Observation Scheduling Problem. *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling (ICAPS 2017)*.

Lawler, E. 1973. Optimal sequencing of a single machine subject to precedence constraints, *Management Science*, 19(5): 544-546.

Tabu-Based Large Neighbourhood Search for Time-Dependent Multi-Orbit Agile Satellite Scheduling

Lei He[1,2], Mathijs de Weerdt[2], Neil Yorke-Smith[2], Xiaolu Liu[1], Yingwu Chen[1]
[1]College of System Engineering, National University of Defense Technology, 410073 Changsha, China
[2]Delft University of Technology, P.O. Box 5031, 2600 GA Delft, The Netherlands

Abstract

Agile Earth observation satellite (AEOS) scheduling is complex, due to long visible time windows and time-dependent transitions between observations. We introduce a generic approach suited for scheduling problems characterised by time-dependency and/or sequence-dependency. Our approach is a novel hybridization of adaptive large neighbourhood search (ALNS) and tabu search. We further introduce partial sequence dominance and insertion position ordering operators to the ALNS. Extensive computational results on a real-world multi-orbit AEOS observation scheduling benchmark show that the hybrid ALNS robustly outperforms an improved mixed integer programming model and two recent state-of-the-art metaheuristic methods. The proposed method increases solution quality by more than 10% and reduces calculation time by more than 70% on average.

Introduction

Agile Earth observation satellites (AEOSs) are a new generation of orbital imaging platforms, possessing three degrees of freedom (roll, pitch, yaw), which enables them to observe targets on the Earth's surface before/after an upright pass and next to/along the path (Maillard 2015). This agility greatly enhances the observing abilities of AEOSs.

Scheduling the operation of AEOSs is complex due to long visible time windows (VTWs), and time-dependent transitions. During the VTW the target is visible for the satellite. AEOS VTWs are much longer than the necessary imaging time; target imaging can start anywhere within its VTW. During the transition time, the satellite adjusts its observing angle between two adjacent observations. This transition time is not only sequence-dependent, but also time-dependent because it depends on the observing angles, which differ for different observation start times. In addition, the observation start time also influences the image quality. The best image quality can be acquired when the satellite is at the nadir, i.e., the middle of the VTW. AEOS scheduling is an NP-hard combinatorial optimization problem (Lemaître et al. 2002).

Research on the offline AEOS scheduling problem can be divided into the Maximum Shot Orbit Sequencing Problem (MSOP) and the Maximum Shot Sequencing Problem (MSP) (Lemaître et al. 2002). MSOP aims to select the images with maximum total priority from a single orbit of one satellite and determine the observing sequence and the observing times without violating constraints. MSP is more complex and realistic. In addition to selecting the image-taking tasks, we must also decide which VTWs are chosen out of *several* consecutive orbits. These two decisions are dynamically coupled rather than procedurally separated.

Due to the complexity of the MSP, there exist few exact algorithms. Bianchessi et al. (2007) propose a column generation method to solve a linear programming (LP) relaxation of the problem. Wang et al. (2011) propose a mixed integer programming (MIP) model, where the continuous observation angle is discretized as only three angles. As a result of this approximation, the solution space is reduced and the transition time can be pre-computed. Both methods can only be used in small-size instances. Besides exact algorithms, a variety of metaheuristics and heuristics have been applied to MSP, including tabu search (Lin et al. 2005; Bianchessi et al. 2007), simulated annealing (Dilkina and Havens 2005; Li, Xu, and Wang 2007), genetic algorithms (Wolfe and Sorensen 2000; Li, Xu, and Wang 2007), hybrid differential evolutionary algorithms (Li et al. 2017) and priority-based constructive algorithms (Wolfe and Sorensen 2000; Wang et al. 2011; Xu et al. 2016). However, all the above works neglect the transition time, fix it as a constant value, or simplify it as a sequence-dependent time.

The only works on MSP with time-dependent transition time to date are by Geng et al. (2016) and Liu et al. (2017). Geng et al. (2016) propose a genetic algorithm and only briefly treat the time-dependency. They also ignore the constraints of image quality, onboard memory and onboard energy. Liu et al. (2017) first define a mathematical model that is non-linear due to the time-dependent constraints. Then, in order to use LP, the authors linearize and simplify the MSP into two separate subproblems: all feasible VTW combinations are enumerated, then, for each combination, LP is used to obtain the optimal schedule. The transition time is fixed as a constant, a major approximation, but even so the approach could not solve instances involving more than 12 tasks. Therefore, the authors propose a metaheuristic based on adaptive large neighbourhood search (ALNS) combined with a fast task insertion heuristic. The angles in VTWs are pre-computed and cached. The ALNS method performs well for small-size instances, while when the problem instance

grows in size, the solution quality deteriorates and the computation time grows.

In this paper, we investigate the time-dependent multi-orbit AEOS observation scheduling problem. The major contributions of this paper are summarized as follows:

1. For the first time, a complete MIP model is defined. Compared with the two-stage mixed integer linear programming (MILP) model in Liu et al. (2017), our model avoids enumerating all VTW combinations. The time-dependent onboard energy constraints are also considered. The new MIP model scales better and is more realistic.
2. An improved ALNS is hybridized with tabu search. Our novel hybrid approach provides results with higher quality and robustness and consumes less time compared with state of the art. The tabu mechanism helps the ALNS to avoid searching recently visited solutions.
3. A partial sequence dominance heuristic is proposed, which can help to collect and use the in-process information that is neglected in standard ALNS. It greatly improves the performance of ALNS, especially when the problem instance grows in size.
4. A position ordering heuristic is included in the task insertion algorithm. This strategy explores more insertion positions following an ascending order of possible transition times to save time and energy resources and increase the possibility of successful insertion.

Problem Description

The time-dependent multi-orbit AEOS observation scheduling problem aims to select a number of tasks from several consecutive orbits and determine the observing sequence and the observing times without violating technical constraints. We define a task t_i as one image or target to observe from the users' task list T. Each task corresponds to a small area on the Earth's surface that can be observed in one pass.

We account for but simplify the onboard memory and energy constraints. We assume that the memory and energy used during each orbit cannot exceed an upper bound to simulate these orbit-renewable resources.

We firstly introduce notation and provides an angle-fitting method to represent the time-dependent transition time in the MIP model. Then we define the MIP model itself.

Time-dependent transition time

According to Liu et al. (2017), the image quality q_i of task t_i must be higher than its required minimum image quality c_i. The image quality is a function of u_{ij}, the observation start time in the j^{th} VTW w_{ij} of t_i, and can be calculated according to the following equation:

$$q_i = 10 - 9\frac{|\frac{u_{ij}+v_{ij}}{2} - w^*_{ij}|}{\frac{l_{ij}}{2} - \frac{d_i}{2}} = 10 - 9\frac{|2u_{ij} + d_i - 2w^*_{ij}|}{l_{ij} - d_i} \tag{1}$$

where v_{ij} is the observation end time of t_i in w_{ij}, d_i is the observation duration, $v_{ij} = u_{ij} + d_i$, l_{ij} is the length of w_{ij}, and w^*_{ij} is the nadir time of w_{ij}.

In Liu et al. (2017), the quality of an image was treated as a constraint, requiring the image quality to be higher than a user-specified minimum value. In our paper, we use Eq. (1) to prune parts of the VTWs to reduce the solution space and increase the accuracy of angle fitting, which enables us to build our MIP model.

According to (1) and $q_i \geq c_i$, the feasible interval of observation start time is within the original VTW, represented by b^*_{ij} and e^*_{ij}:

$$b^*_{ij} = \max\left(\frac{(l_{ij} - d_i)(c_i - 10)}{18} + w^*_{ij} - \frac{d_i}{2}, b_{ij}\right) \tag{2}$$

$$e^*_{ij} = \min\left(\frac{(l_{ij} - d_i)(10 - c_i)}{18} + w^*_{ij} - \frac{d_i}{2}, e_{ij} - d_i\right) \tag{3}$$

The exact VTWs and the observing angle sequences for tasks are computed before solving the actual scheduling problem. This pre-processing phase takes the satellite's position, the task's position as well as the Earth's rotation and produces VTWs and time-dependent functions of the roll, pitch and yaw angles for every VTW. Although these functions are non-linear, the change of angles can be approximated quite well with a linear function between b^*_{ij} and e^*_{ij} (e.g., on average over 200 tasks, the duration of a VTW is over 300s, while the transition time error is less than 0.5s).

Mixed integer programming model

Objective function The importance of one task t_i is evaluated by its priority $g_i \in [1, 10]$. The objective we consider is to maximize the total priority of all the scheduled tasks:

$$\text{Maximize} \quad \sum\nolimits_{i=1}^{|T|} \sum\nolimits_{j=1}^{|W_i|} x_{ij} g_i \tag{4}$$

where W_i is the VTW set of t_i and x_{ij} is a binary decision variable equal to 1 if and only if w_{ij} is chosen to observe t_i. Another key decision variable is u_{ij}, determining the observation start time for t_i in w_{ij}.

Constraints Constraints (5) are the uniqueness constraints, meaning that each task is observed once at most.

$$\sum\nolimits_{j=1}^{|W_i|} x_{ij} \leq 1 \quad \forall t_i \in T \tag{5}$$

Constraints (6) are the manoeuvring constraints: they ensure that there is sufficient time between the end time v_{ij} and the start time u_{kl} for the transition time $\tau_{w_{ij}w_{kl}}$ between tasks t_i and t_k. These constraints are only enforced for VTWs that are closer to each other than the maximum possible transition time τ_{max}, and if w_{ij} and w_{kl} are selected to observe t_i and t_k respectively, and t_i is the immediate predecessor of t_k, which is encoded by $\rho_{w_{ij}w_{kl}} = 1$ and Constraints (7)–(9).

$$\begin{gathered} v_{ij} + \tau_{w_{ij}w_{kl}} \leq u_{kl} \quad \text{if} \quad \rho_{w_{ij}w_{kl}} = 1 \\ \forall t_i, t_k \in T, i \neq k, w_{ij} \in W_i, w_{kl} \in W_k, \\ b^*_{kl} - e^*_{ij} \leq \tau_{max}, b^*_{ij} - e^*_{kl} \leq \tau_{max} \end{gathered} \tag{6}$$

$$\sum_{k=1}^{|T|} \sum_{l=1}^{|W_k|} \rho_{w_{ij}w_{kl}} + \rho_{w_{ij}w^e} = x_{ij} \quad \forall w_{ij} \in W_i, t_i \in T \tag{7}$$

$$\sum_{k=1}^{|T|}\sum_{l=1}^{|W_k|}\rho_{w_{kl}w_{ij}}+\rho_{w^s w_{ij}}=x_{ij}\quad\forall w_{ij}\in W_i,t_i\in T \quad (8)$$

$$\sum_{i=1}^{|T|}\sum_{j=1}^{|W_i|}\rho_{w_{ij}w^e}=\sum_{i=1}^{|T|}\sum_{j=1}^{|W_i|}\rho_{w^s w_{ij}}=1 \quad (9)$$

In Constraints (7) and (8), w^s and w^e are two dummy nodes representing the first and the last VTW on the satellite. These constraints express that if a VTW is selected, there is a unique selected VTW preceding it, and a unique selected VTW following it.

Constraints (10) ensure that the observation for each task lasts for the required duration.

$$v_{ij}=u_{ij}+d_i\quad\forall w_{ij}\in W_i,t_i\in T \quad (10)$$

Constraints (11)–(15) are used to calculate the transition time between two observations. In Constraints (11), $\theta_{w_{ij}w_{kl}}$ is the total transition angle and a_1–a_4 are four different transition angular velocities for different transition angles. In (12), $\gamma_{ij}^t,\pi_{ij}^t,\psi_{ij}^t$ are observation roll, pitch and yaw angles at t, which are calculated by (13)–(15). The parameters a_{ij}^{γ}, a_{ij}^{π}, a_{ij}^{ψ}, b_{ij}^{γ}, b_{ij}^{π} and b_{ij}^{ψ} are the parameters of the functions of angles and time of the chosen VTW w_{ij}, which are computed in the pre-processing phase of section 'Time-dependent transition time'.

$$\tau_{w_{ij}w_{kl}}=\begin{cases}10+\theta_{w_{ij}w_{kl}}/a_1 & \theta_{w_{ij}w_{kl}}\le 15\\ 15+\theta_{w_{ij}w_{kl}}/a_2 & 15<\theta_{w_{ij}w_{kl}}\le 40\\ 20+\theta_{w_{ij}w_{kl}}/a_3 & 40<\theta_{w_{ij}w_{kl}}\le 90\\ 25+\theta_{w_{ij}w_{kl}}/a_4 & \theta_{w_{ij}w_{kl}}>90\end{cases}$$
$$\forall t_i,t_k\in T,i\neq k,w_{ij}\in W_i,w_{kl}\in W_k \quad (11)$$

$$\theta_{w_{ij}w_{kl}}=\left|\gamma_{ij}^{u_{ij}}-\gamma_{kl}^{u_{kl}}\right|+\left|\pi_{ij}^{u_{ij}}-\pi_{kl}^{u_{kl}}\right|+\left|\psi_{ij}^{u_{ij}}-\psi_{kl}^{u_{kl}}\right|$$
$$\forall t_i,t_k\in T,i\neq k,w_{ij}\in W_i,w_{kl}\in W_k \quad (12)$$

$$\gamma_{ij}^t=a_{ij}^{\gamma}t+b_{ij}^{\gamma}\quad\forall w_{ij}\in W_i,t_i\in T \quad (13)$$

$$\pi_{ij}^t=a_{ij}^{\pi}t+b_{ij}^{\pi}\quad\forall w_{ij}\in W_i,t_i\in T \quad (14)$$

$$\psi_{ij}^t=a_{ij}^{\psi}t+b_{ij}^{\psi}\quad\forall w_{ij}\in W_i,t_i\in T \quad (15)$$

Constraints (16) and (17) are the memory and energy constraints, respectively, where α^M and α^E are two estimated values which measure the percentage of total memory M and energy E available on an orbit, r_{ij}^m is a binary parameter showing whether w_{ij} is on the m^{th} orbit o_m of the orbit set O, and m^o,p^o,p^s,p^a are the consumed memory for observation per second, the consumed energy for observation per second, the consumed energy per observation and the consumed energy for angle transition per degree respectively. The energy constraints are also time-dependent because the energy for satellite transition depends on the total angles the satellite rotates. In Constraints (18), $\theta^*_{w_{ij}w_{kl}}$ is an auxiliary variable to calculate the rotation energy. The value of $\theta^*_{w_{ij}w_{kl}}$ is a piecewise linear function influenced by the value of $\rho_{w_{ij}w_{kl}}$.

$$\sum_{i=1}^{|T|}\sum_{j=1}^{|W_i|}r_{ij}^m d_i m^o\le\alpha^M M\quad\forall o_m\in O \quad (16)$$

$$\sum_{i=1}^{|T|}\sum_{j=1}^{|W_i|}r_{ij}^m(x_{ij}d_i p^o+x_{ij}p^s+\sum_{k=1}^{|T|}\sum_{l=1}^{|W_k|}\theta^*_{w_{ij}w_{kl}}p^a)\le\alpha^E E$$
$$\forall o_m\in O \quad (17)$$

$$\theta^*_{w_{ij}w_{kl}}=\begin{cases}\theta_{w_{ij}w_{kl}} & \text{if } \rho_{w_{ij}w_{kl}}=1\\ 0 & \text{otherwise}\end{cases} \quad (18)$$
$$\forall t_i,t_k\in T,i\neq k,w_{ij}\in W_i,w_{kl}\in W_k$$

Constraints (19)–(21) restrict the domains of the variables. Note that in Constraints (20), the start and end time of the VTW have been cut according to the quality constraints in (2) and (3), so there are no additional quality constraints.

$$x_{ij}\in\{0,1\}\quad\forall w_{ij}\in W_i,t_i\in T \quad (19)$$

$$b_{ij}^*\le u_{ij}\le e_{ij}^*\quad\forall w_{ij}\in W_{ij},t_i\in T \quad (20)$$

$$\rho_{w_{ij}w_{kl}}\in\{0,1\}\quad\forall t_i,t_l\in T,i\neq k,w_{ij}\in W_i,w_{kl}\in W_k \quad (21)$$

To the best of our knowledge, this is the first MIP model proposed for the complete time-dependent multi-orbit AEOS observation scheduling problem. The angle fitting strategy enables the modelling of the time-dependency. Compared with the two-stage MILP model in Liu et al. (2017), we avoid enumerating all the VTW combinations and we consider energy constraints, which are also time-dependent.

This problem is NP-hard, and we observe that the runtime of the MIP solver does not scale well with larger problem size. Therefore, in the next section, we propose a metaheuristic approach.

Hybrid ALNS Algorithm

ALNS (Pisinger and Ropke 2007; Liu et al. 2017) provides a flexible framework in which several different operators can be defined according to the problem characteristics. ALNS can be adopted to provide solutions for instances with different characteristics. However, we observe two main drawbacks of ALNS. First, the search efficiency of ALNS can founder due to re-visiting recent solutions. Second, ALNS accepts a new solution depending on the quality of the whole solution sequence. However, during the search process, solutions with some good parts are rejected due to the low quality of the whole sequence – thus neglecting potentially valuable in-process information.

Our ALNS approach is based on the work of Liu et al. (2017). In the following subsections, we first introduce the standard ALNS framework. Then we introduce three new main features of our ALNS approach: tabu search hybridization (TS), partial sequence dominance (PSD), and insertion positions ordering (IPO). The resulting algorithm, called ALNS/TPI, is shown as Algorithm 1. In the fifth subsection we introduce a new definition of conflict degree (CD) which increases the calculation speed.

ALNS framework

ALNS is less sensitive to the initial solution than general local search (Demir, Bektaş, and Laporte 2012), therefore we use a simple greedy search to construct an initial solution. We sort the tasks by an ascending order of start times of

Algorithm 1 Overview of ALNS/TPI

```
Generate an initial solution S by greedy search;
repeat
    Choose destroy, repair operators D_i, R_i based on weights;
    S' ← R_i(D_i(S));
    Update tabu attributes of new inserted tasks;
    Produce compound solution S_c from S and S';
    if f(S_c) > f(S') then
        S' ← S_c;
    if SA accepts S' then
        S ← S';
    if f(S) > f(S*) then
        S* ← S;
    if S* has not improved for many iterations then
        S ← S*;
    Update the weights of operators;
until Terminal condition is met;
return S*.
```

their VTWs and we attempt to insert each task under all the constraints stated above.

ALNS updates solutions through destroying and repairing. In the destroying process, some tasks are removed from the current solution by removal operators. The unscheduled and removed tasks are then inserted into the destroyed solution in the repairing process by insertion operators. There are six removal and three insertion operators: removal by random, min priority (tasks with lower priority are removed first), max opportunity (tasks with more VTWs are removed first), max conflict (tasks with higher conflict degree are removed first), cluster 1 (tasks in the orbits with fewest tasks are removed first) and cluster 2 (tasks in the orbits with the lowest priority are removed first); insertion by max priority, min opportunity and min conflict.

At each iteration, a pair of removal and insertion operators is selected according to their weights. The weights are updated adaptively according to the performance of operators in the previous iterations. A simulated annealing (SA) criterion is used to control the acceptance of new solutions.

Tabu search

Žulj, Kramer, and Schneider (2018) propose a method hybridizing ALNS with TS, and apply it to the order-batching problem. Their method combines the diversification capabilities of ALNS and the intensification capabilities of TS. It uses ALNS to search for better solutions and, if a certain number of ALNS iterations have passed, TS is used. Thus ALNS and TS are alternated in a two-stage manner. But since ALNS and TS are used in separate stages, this hybridization does not change the short-term cycling nature of ALNS.

In contrast, we propose a tight integration of ALNS with TS. We declare a removal tabu attribute for each task. Whenever one task is inserted into the current solution, the removal of this task is forbidden for $\sqrt{|T|/2}$ iterations. We use this strategy to prevent the algorithm re-visiting recent evaluated solutions. We compare the two ALNS–TS hybridizations in the experiments below.

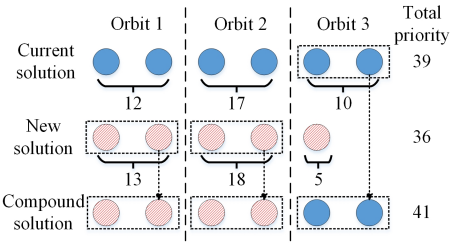


Figure 1: An example of partial sequence dominance

Partial sequence dominance

Due to the time-dependency and sequence-dependency, the quality of a solution is influenced significantly by its partial sequences. Inspired by genetic algorithms, we propose the PSD heuristic. When a new solution is produced, we compare a small part of it with the corresponding part of the current solution. In this paper we use the orbit as the small part. Figure 1 shows one example. In standard ALNS, the new solution is given up. However, Orbit 1 and Orbit 2 of the new solution are better than the current solution. So according to PSD, we keep Orbit 1 and Orbit 2 of the new solution and Orbit 3 of the current solution, and we get the compound solution, which is better than the current solution.

This paper studies a multi-orbit scheduling problem, which means one task might have multiple VTWs on different orbits. Therefore the compound solution might violate the uniqueness constraints (5). When a compound solution is produced, the feasibility is checked and the tasks that violate Constraints (5) are removed. If the repaired compound solution is better than the new solution, it is accepted. We note that for a MSOP problem, this check can be omitted.

Insertion positions ordering

In Liu et al. (2017), two strategies to select the observation start time are used: the earliest start time insertion and the middle start time insertion. According to their experiments, the middle start time insertion strategy works better. However, both of them waste too many insertion opportunities.

We propose an insertion position ordering (IPO) strategy to insert tasks. For every candidate task, we calculate all possible insertion positions. Due to the time-dependency and sequence-dependency, the difference of transition times in different insertion positions can be large. To increase the possibility of successful insertion without increasing the computation time too much, we calculate the possible transition time for each position and insert the task into the positions following an ascending order of possible transition times. The rationale is that time is a valuable resource, especially when we consider energy constraints, and it is better to use time for observation instead of transitioning.

Note because we cannot know the observation start time until we insert the task into the solution sequence, we cannot know the exact transition time. Therefore, we use the angles at the middle of the VTWs to compute an approximate transition time. This value is used to rank the possible positions.

Conflict degree

In Liu et al. (2017), the heuristic conflict degree (CD) is defined as 'the time that one VTW is overlapped with other scheduled tasks'. Since the solution is changed in every ALNS iteration, CD must be updated in each iteration. In order to reduce the computation time, we instead define CD as 'the time that one VTW is overlapped with any other VTWs'. This calculation can be done in the pre-processing phase. We show the quality of solutions is barely affected and the computation time is significantly decreased.

Empirical Results

The aim of the experiments is to assess the effectiveness of the proposed ALNS/TPI hybrid algorithm. Experiments were conducted using Intel Core i5 3.20GHz CPU running Windows 7 with 8GB memory. A time limit of 3600s is used. IBM ILOG CPLEX version 12.8 is used for MIP solving. The results for metaheuristics are the average of 20 runs.

The generation of the problem instances follows the configuration from Liu et al. (2017) except for the required minimum quality. In previous work, the required minimum quality c_i is set at 0 and this makes the quality constraints useless. Therefore in this paper we set it as a uniform random integer in $[5, 10]$ (hence, solution quality for the same-size instance in our experiments is lower than the one in Liu et al. (2017)).

The tasks are generated according to a uniform random distribution over two geographical regions: China and the whole world. For the Chinese area distribution mode, fifteen instances are designed and the number of tasks contained in these instances changes from 50 to 400, with an increment step of 25. For the worldwide distribution mode, twelve instances are designed and the number of tasks contained in these instances changes from 50 to 600, with an increment step of 50. Other parameters of the problem instances are: $M = 2400$, $E = 2400$, $m^o = 1$, $p^o = 1$, $p^s = 2$, $p^a = 1$, $\alpha^M = 0.6$, $\alpha^E = 0.8$, $a_1 = 1.5$, $a_2 = 2$, $a_3 = 2.5$, $a_4 = 3$.

Comparison of different algorithms First, we compare the proposed ALNS/TPI with our improved MIP model and the metaheuristics in Liu et al. (2017) ('old ALNS'), and the coarse ALNS–TS hybrid of Žulj, Kramer, and Schneider (2018) ('ALNS/TS'). The parameters of the ALNS algorithms are as in Liu et al. (2017): the total iteration time is 5000 and the simulated annealing parameter is 0.9975. In ALNS/TS, TS is run for 200 iterations after every 1000 iterations of ALNS. In each TS iteration, 10 neighbourhoods by our removal and insertion operators are examined to find the best local move. The whole process is run four times, for 12000 neighbourhood moves in total. Recently visited solutions are inserted in a tabu list for $\sqrt{|T|/2}$ iterations. Here, we implemented two versions of the modified (Žulj, Kramer, and Schneider 2018) algorithm. The first one is ALNS/TS for MSP without any further improvements. The second, called ALNS/TS/PI, has all the improvement features except the tight TS hybridization.

We compare the solution quality and the CPU time. The solution quality is the percentage of the total priority of scheduled tasks (i.e., the objective value) divided by the total priority of all the tasks in T. Figure 2 shows the comparison of the five different algorithms: our ALNS/TPI, old ALNS, ALNS/TS, ALNS/TS/PI, and our improved MIP. In Figure 2 left (for Chinese area) and middle (for worldwide), black solid lines show the solution quality (left axis) and the blue dash lines show the CPU time (right axis, log scale), showing that the CPU time of the ALNS/TPI increases slowly with the increasing number of tasks. The solution quality is significantly higher than that of the old ALNS and ALNS/TS. ALNS/TS/PI uses more time to produce solutions with worse quality, which proves that our integrated hybridization of ALNS and TS is better than the two-stage hybridization in Žulj, Kramer, and Schneider (2018) for this MSP problem. Furthermore, the implementation of our hybridization is easier than ALNS/TS because we only need to add tabu attributes of tasks to ALNS, while in ALNS/TS, a new TS search process is included. MIP can find optimal solutions for small-size instances but performs badly when the instance size gets large. For the three small instances with optimal solutions by MIP, ALNS/TPI also finds the same optimal solution. Among all the methods, old ALNS performs worst, consuming a long time to produce solutions with the lowest quality. Finally, Figure 2 right shows the anytime quality of different algorithms for instance 600_W with 600 tasks distributed worldwide. The MIP method fails to give a solution within the time limit for this large instance.

Second, in order to compare the improved MIP model with the two-stage MILP model in Liu et al. (2017), we fix the transition time of our model as 20s and remove the energy constraints ('MIP(20s)').

Figure 3 (top) shows the number of instances solved within 600s by different methods. The proposed ALNS/TPI, as well as ALNS/TS and ALNS/TS/PI can solve all the problem instances. The old ALNS, however, fails to solve three worldwide instances. MIP(20s) and the improved MIP can only solve small-size instances. If we set the time limit as 3600s (in Figure 3 (bottom)), the old ALNS and MIP(20s) can solve all the instances. The improved MIP can solve eight more instances. Unfortunately, the two-stage MILP model in Liu et al. (2017) fails to solve all the problem instances because of memory overflow. It cannot enumerate all the combinations of VTWs even for our smallest instance: the model can only solve problem with at most 12 tasks.

Comparison of different features of ALNS/TPI Last, in order to understand which features of ALNS/TPI contribute to its superior performance, we perform a factor analysis of features. The results are shown in Tables 1 and 2 (we also include the results of ALNS/TS, ALNS/TS/PI and old ALNS). We compare ALNS without PSD (ALNS/TI), ALNS without TS (ALNS/PI), ALNS without IPO (ALNS/TP) and ALNS with frequent CD update (ALNS/TPI/CD). All the results are compared with ALNS/TPI, so for other algorithms, we calculate the percentage of increase in quality (IQ, higher is better) and increase in time (IT, lower is better).

All the features contribute more to the solution quality for area distribution. Among all these features, IPO contributes most to the solution quality. However, IPO also increases

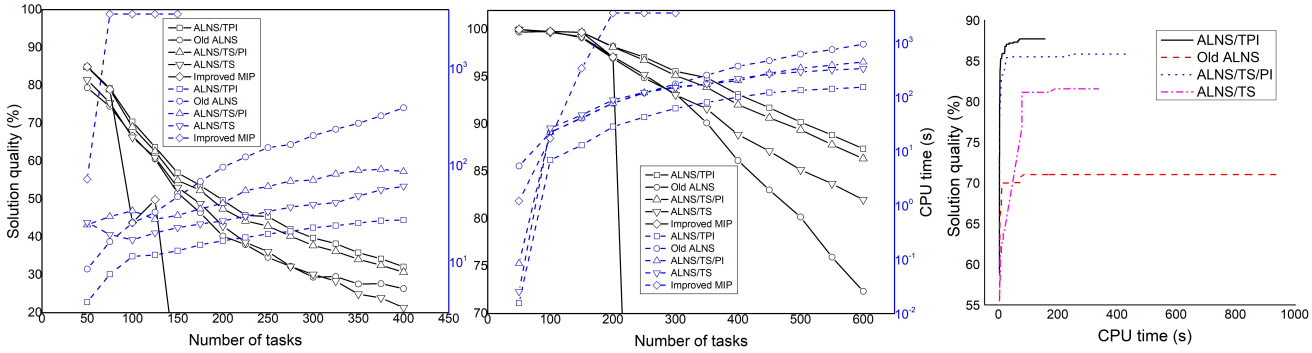


Figure 2: Comparison of algorithms on area distribution (left) and worldwide (middle), anytime quality of different algorithms (right)

Table 1: Results of different ALNSs for area distribution

Instance	ALNS/TPI		ALNS/TI		ALNS/PI		ALNS/TP		ALNS/TPI/CD		ALNS/TS		ALNS/TS/PI		Old ALNS	
	Quality/%	Time/s	IQ/%	IT/%	IQ/%	IT/%	IQ/%	IT/%	IQ/%	IT/%	IQ/%	IT/%	IQ/%	IT/%	IQ/%	IT/%
50_A	**84.94**	3.86	**0.00**	-7.12	-2.55	-0.43	-3.78	**-33.23**	-0.35	12.38	-4.26	84.91	-0.35	84.02	-7.01	54.34
75_A	**79.24**	7.47	-0.04	-6.19	-1.94	-1.95	-3.18	**-44.29**	-0.28	18.43	-5.12	61.14	-0.41	74.65	-6.38	53.94
100_A	**70.44**	11.42	-0.60	0.48	-3.24	-11.83	-5.14	**-73.69**	-0.56	27.69	-6.25	32.60	-2.04	66.18	-5.55	54.75
125_A	63.75	11.87	-0.07	8.30	-2.68	-11.66	-4.87	**-56.93**	**0.14**	27.07	-4.61	40.50	-1.92	57.04	-5.23	63.63
150_A	**56.95**	13.04	-1.42	5.87	-2.97	-6.76	-6.60	**-55.78**	-0.18	32.89	-7.34	42.23	-3.64	56.94	-10.29	72.23
175_A	53.57	15.05	-1.34	3.73	-2.62	-15.34	-4.62	**-51.04**	**0.24**	39.20	-9.81	39.04	-2.44	57.37	-15.55	77.68
200_A	49.70	16.59	-2.82	3.12	-4.41	-16.37	-6.46	**-44.88**	**0.26**	41.49	-15.94	37.34	-4.76	58.92	-23.49	82.54
225_A	45.73	17.93	-3.30	2.05	-3.45	-15.52	-5.57	**-47.45**	**0.62**	40.86	-18.44	39.51	-3.32	67.09	-20.38	85.19
250_A	**45.39**	19.49	-5.00	2.06	-4.76	-17.63	-9.13	**-40.99**	-0.23	45.70	-25.70	41.16	-5.89	67.58	-31.16	87.13
275_A	42.07	20.67	-3.36	3.78	-3.07	-11.25	-6.98	**-35.90**	**0.36**	52.38	-30.48	43.85	-4.57	69.73	-30.51	87.30
300_A	39.75	22.57	-2.69	0.71	-3.43	-14.01	-8.30	**-37.89**	**0.86**	54.34	-31.99	42.35	-5.32	67.76	-35.44	88.80
325_A	38.26	23.76	-2.62	1.98	-3.38	-11.75	-9.10	**-32.95**	**1.42**	61.09	-34.99	41.96	-5.67	70.65	-29.40	89.85
350_A	35.85	25.26	-2.32	0.73	-2.89	-9.79	-7.64	**-28.85**	**1.48**	65.26	-43.97	47.32	-5.25	71.27	-30.05	90.62
375_A	34.26	26.73	-3.45	-1.92	-2.67	-11.99	-6.95	**-33.75**	**1.70**	68.27	-43.32	51.84	-5.23	70.46	-23.84	91.68
400_A	32.15	27.13	-2.08	-1.20	-1.78	-3.14	-6.30	**-28.61**	**1.82**	68.84	-50.69	55.16	-4.82	68.40	-21.95	93.04
Avg.	51.47	17.52	-2.07	1.09	-3.06	-10.63	-6.31	**-43.08**	**0.49**	43.73	-22.19	46.73	-3.71	67.20	-19.75	78.18

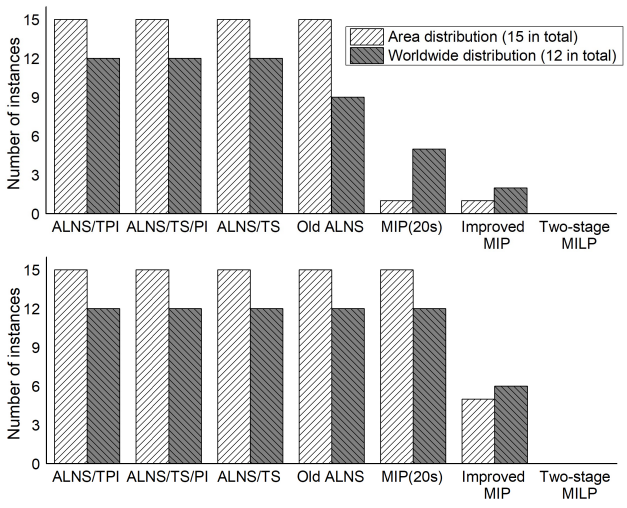


Figure 3: Number of instances solved within 600s (top) and 3600 (bottom)

the CPU time more compared with PSD and TS. TS works better than PSD, especially for the area distribution. This is because for area distribution, the distribution of tasks is dense and the CD of tasks is high. It is then more difficult for the algorithm to find a good solution. TS, which prevents the algorithm from searching recent solutions again, works better in this case. PSD works much better when the problem instance gets larger, which means that PSD performs well when the solution sequences get long. When the solution sequences get long, evaluating a solution only by its total quality gives up too much in-process information of partial sequences. ALNS/TPI/CD works better than ALNS/TPI for the area distribution because CD is an important heuristic influencing the quality for the dense distribution. However, the CPU time is nearly 3 times that of ALNS/TPI. From the results of ALNS/TPI/CD for the worldwide distribution, we can find that the frequent CD updates do not contribute to the solution quality all the time. The previous definition of CD focuses too much on the conflict with scheduled tasks while neglecting the potential conflict with unscheduled tasks.

Table 2: Results of different ALNSs for worldwide distribution

Instance	ALNS/TPI		ALNS/TI		ALNS/PI		ALNS/TP		ALNS/TPI/CD		ALNS/TS		ALNS/TS/PI		Old ALNS	
	Quality/%	Time/s	IQ/%	IT/%	IQ/%	IT/%	IQ/%	IT/%	IQ/%	IT/%	IQ/%	IT/%	IQ/%	IT/%	IQ/%	IT/%
50_W	**100.00**	**0.02**	**0.00**	72.00	**0.00**	79.90	-2.17	99.41	**0.00**	44.74	**0.00**	38.12	**0.00**	81.31	-0.28	99.71
100_W	**99.78**	**6.91**	**0.00**	7.29	**0.00**	11.47	-1.49	10.81	**0.00**	19.30	-0.10	74.21	**0.00**	69.36	**0.00**	69.00
150_W	**99.70**	**12.92**	**0.00**	10.18	-0.20	14.16	-2.26	17.19	**0.00**	51.45	-0.47	72.48	**0.00**	68.92	-0.57	68.41
200_W	98.19	28.46	0.00	**-5.55**	-0.19	-0.14	-2.30	-2.69	**0.09**	45.27	-1.10	67.99	-0.04	63.56	-1.27	63.05
250_W	**97.08**	42.88	**0.00**	**-5.13**	-0.60	2.31	-2.38	-3.99	-0.05	45.47	-1.93	64.99	-0.35	64.56	-2.30	64.21
300_W	**95.58**	61.99	**0.00**	-3.49	-0.56	-8.02	-3.29	**-8.56**	-0.15	43.46	-2.67	58.59	-0.46	60.33	-2.56	64.50
350_W	**94.91**	79.99	**0.00**	-2.78	-1.11	**-22.33**	-4.36	-14.59	-0.19	48.61	-3.58	54.47	-1.09	54.06	-5.30	68.60
400_W	**93.14**	100.12	-0.06	-1.24	-0.99	**-24.80**	-4.90	-21.52	-0.20	51.80	-4.77	54.65	-1.17	50.43	-8.11	73.21
450_W	**91.73**	121.16	-0.23	-1.78	-0.98	-14.79	-5.48	**-25.84**	-0.24	56.07	-5.26	54.66	-1.17	56.20	-10.47	74.27
500_W	**90.22**	134.52	-0.54	1.55	-0.69	-5.23	-5.32	**-28.04**	-0.06	61.52	-5.93	54.19	-0.92	60.93	-12.50	78.54
550_W	**88.84**	142.44	-0.64	5.92	-0.84	-2.87	-4.79	**-20.21**	-0.11	66.28	-6.13	55.35	-1.15	63.52	-16.96	81.12
600_W	**87.41**	153.27	-0.85	6.66	-0.60	0.37	-5.38	**-20.74**	-0.17	69.63	-6.59	54.60	-1.22	65.77	-20.82	83.96
Avg.	**94.71**	73.72	-0.19	6.97	-0.56	2.50	-3.68	**-1.56**	-0.09	50.30	-3.21	58.69	-0.63	63.25	-6.76	74.05

Application in Real World

In this section, we discuss the difference between our simulations and the potential application of our work in the real world.

First, although the instances in our test instances are randomly generated, they are very similar to real-world ones. We do not have access to (often classified) instances of task locations. However, in our instances, except for task locations, all parameters are real: satellite's, orbits', Earth's parameters. The calculation of VTWs and transition time is based on the real geographical locations. There is little difference between our and classified instances, since in reality tasks are raised by the users and can be anywhere on the Earth surface. Further, tasks can be dense in a small area: we use the Chinese area distribution to simulate this. The number of tasks can be different on different days: we use different numbers of tasks to simulate this.

The satellite used in the simulation is called AS-01, which is the first AEOS of China. The scheduler of AS-01 was developed by the group of Liu et al. (2017), which uses several simple heuristic operators of the old ALNS to construct the observation schedule. The satellite has now been in orbit for more than two years and the scheduler has worked well until now. However, since the current scheduler is simple and greedy, the solutions generated by it are generally of lower quality than the ones generated by the old ALNS. But since our hybrid ALNS is much more efficient than the old ALNS, we believe it could improve on current operational procedures.

Another difference between our model and real-life satellites is the constraints of specific satellites. For example, for some satellites, the observation time in an orbit is bounded not only by memory and energy, but also by the maximum continuous working time of sensors and the maximum working temperature. If this information is known, it can be added as additional constraints to the proposed models.

Conclusions

We studied time-dependent multi-orbit agile Earth observation satellite scheduling, which is a complex real-world scheduling problem. We developed the first realistic mixed integer programming (MIP) model, and a novel hybridization of adaptive large neighbourhood search (ALNS) and tabu search (TS). As expected, MIP can find optimal solutions only for small-size instances. Extensive empirical results demonstrated that, compared with two state-of-the-art metaheuristic approaches, our proposed ALNS hybrid produces solutions with higher quality in less time. Factor analysis finds the novel insertion position ordering contributes most to the performance, but also consumes the most time. The partial sequence dominance heuristic performs better when the problem instance grows in size. The TS heuristic performs better when the conflict degree is high.

Our work proves that ALNS and TS hybridization is an efficient method for this satellite scheduling problem. Our next step is to evaluate the heuristics in this work on other similar problems. We believe these strategies can significantly improve other algorithms for problems characterized by time- and/or sequence-dependency.

Acknowledgements

We gratefully thank the SPARK 2018 reviewers for their valuable comments.

References

Bianchessi, N.; Cordeau, J.-F.; Desrosiers, J.; Laporte, G.; and Raymond, V. 2007. A heuristic for the multi-satellite, multi-orbit and multi-user management of Earth observation satellites. *European Journal of Operational Research* 177(2):750–762.

Demir, E.; Bektaş, T.; and Laporte, G. 2012. An adaptive large neighborhood search heuristic for the pollution-routing problem. *European Journal of Operational Research* 223(2):346–359.

Dilkina, B., and Havens, B. 2005. Agile satellite scheduling via permutation search with constraint propagation. Technical report, Actenum Corporation, `www.cs.sfu.ca/CourseCentral/827/havens/papers/topic`.

Geng, X.; Li, J.; Yang, W.; and Gong, H. 2016. Agile satellite scheduling based on hybrid coding genetic algorithm. In *12th World Congress on Intelligent Control and Automation (WCICA)*, 2727–2731.

Lemaître, M.; Verfaillie, G.; Jouhaud, F.; Lachiver, J.-M.; and Bataille, N. 2002. Selecting and scheduling observations of agile satellites. *Aerospace Science and Technology* 6(5):367–381.

Li, G.; Chen, C.; Yao, F.; He, R.; and Chen, Y. 2017. Hybrid differential evolution optimisation for earth observation satellite scheduling with time-dependent earliness-tardiness penalties. *Mathematical Problems in Engineering* 2017:Article ID 2490620.

Li, Y.; Xu, M.; and Wang, R. 2007. Scheduling observations of agile satellites with combined genetic algorithm. In *International Conference on Natural Computation*, 29–33.

Lin, W.-C.; Liao, D.-Y.; Liu, C.-Y.; and Lee, Y.-Y. 2005. Daily imaging scheduling of an earth observation satellite. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans* 35(2):213–223.

Liu, X.; Laporte, G.; Chen, Y.; and He, R. 2017. An adaptive large neighborhood search metaheuristic for agile satellite scheduling with time-dependent transition time. *Computers & Operations Research* 86:41–53.

Maillard, A. 2015. Flexible scheduling for an agile earth-observing satelllite. In *24th International Joint Conference on Artificial Intelligence (IJCAI)*, 4379–4380.

Pisinger, D., and Ropke, S. 2007. A general heuristic for vehicle routing problems. *Computers & Operations Research* 34(8):2403–2435.

Wang, P.; Reinelt, G.; Gao, P.; and Tan, Y. 2011. A model, a heuristic and a decision support system to solve the Earth observing satellites fleet scheduling problem. *Computers & Industrial Engineering* 61(2):322–335.

Wolfe, W. J., and Sorensen, S. E. 2000. Three scheduling algorithms applied to the earth observing systems domain. *Management Science* 46(1):148–166.

Xu, R.; Chen, H.; Liang, X.; and Wang, H. 2016. Priority-based constructive algorithms for scheduling agile earth observation satellites with total priority maximization. *Expert Systems with Applications* 51:195–206.

Žulj, I.; Kramer, S.; and Schneider, M. 2018. A hybrid of adaptive large neighborhood search and tabu search for the order-batching problem. *European Journal of Operational Research* 264(2):653–664.

Discrete Uncertainty Representation for CSP-based Planning and Scheduling and Application to Control Command Systems

Philippe Morignot
ASPertise
pmorignot@aspertise.net

Christophe Guettier
Safran Electronics & Defense
christophe.guettier@safrangroup.com

Abstract

Intelligent and Autonomous Unmanned Ground Vehicles (AUGV), both for civil security and warfare domains, are subject of growing interest from the Intelligent Transportation Systems community and from the Planning & Scheduling (P&S) one. This paper presents an approach to cope with discrete uncertainty representations of P&S problems. The model enables statement of coordination constraints within multiple agents. This representation is based on a discrete confidence interval, denoting bounds around an exact (certain) value provided at planning time. Search algorithms are also proposed, solving P&S problems of realistic size. An implementation, inside the CSP-based P&S system known as ORTAC, demonstrates that the computation time, due to this additional uncertainty representation, is not significantly degraded.

Introduction

Since their introduction in the early 90s, intelligent and autonomous vehicles aim at progressively replacing manually-driven vehicles by computer-driven automated ones, in order to prevent traffic accidents and injuries/fatalities. In general, Intelligent Transportation Systems (ITS) *cooperate* to handle free space and more generally road access.

But in many applications such as search and rescue, natural disaster response, or defense and security missions, several vehicles, manned or unmanned, have to *collaborate* to achieve a common goal. For those applications, Autonomous Unmanned Ground Vehicles (AUGV) are of a particular interest for several dangerous or fastidious missions.

AUGV involve several software modules such as simultaneous localization and mapping (SLAM), perception, data fusion, path planning and then control of the robotic platform. Each functionality has to deal with some form of temporal and spatial uncertainty while representing the environment. In our work, we consider multiple agents (with manned or unmanned vehicles) which traverse a topological map and which must eventually coordinate their respective actions via a control/command (C2) system. While manned coordination with voice communications can be very efficient within a trained team of first responders, interacting with AUGV can be a challenge, especially considering temporal uncertainty.

The paper focuses on the Planning and Scheduling (P&S) software modules and, more specifically, on the temporal uncertainty resulting from the environment or from the various processing stages. These modules will be integrated in C2 systems that coordinate the different vehicles and enable interactions between human and AUGV. The P&S environment is named ORTAC, standing for Optimal Resource and Technical Action Control, and has been developped for both defense and civil security domains, even if other applications are investigated.

The problem also involves technical actions (e.g., observations, measurements, communications) to perform on some waypoints and must consider specific metrics such as security, travelling distances and durations. That is, given a graph where vertices are locations and edges are routes, the P&S problem is to find for all agents a sequence of vertices (or route segments) with pass-by dates on waypoints, optimizing a mission cost. Once the plan defined and communicated, AUGVs must automatically manage their own trajectory and follow their navigation waypoints using control algorithms and time sequence. Drivers of manned platform follow a path of timed waypoints provided by a C2 interface.

A classical candidate approach for this problem is, for example, the A* algorithm (Hart, Nilsson, and Raphael 1968) considered as a best-first search in a space of paths. Even if A* can handle several metrics, including timing, it assumes that there exists only one agent which traverses edges and vertices to reach a final location. Our approach considers constraint programming, that has been identified since the 70s (Montanari 1974) (Laurière 1978) as a powerful tool to represent and solve combinatorial problems, or Constraint Satisfaction Problems (CSP). Its real-world applications are numerous, refer to (Simonin et al. 2015) to only mention a spectacular one.

In both C2 systems and ITS, path planning for multiple agents in a topological map can be modeled and solved using constraint programming (Guettier 2007). A CSP is composed of a set of variables, their domains and algebraic constraints (together compsing a *model*), which are based on abstracting some problem. However, due to many sources of uncertainty, the passing date of an agent on a given vertex/location in the topological map might not be precisely

known at planning time: Some form of uncertainty has to be considered, in order to represent a lack of knowledge at planning time.

In this paper, we propose a new representation of uncertainty, based on confidence intervals, within a CSP model representing path planning in a topological map for multiple agents with coordination. The paper is organized as follows: The first section describes the system environment and the application; The second section presents the basic CSP model, representing path planning for multiple agents in a topological map, and then a new uncertainty model; The third section provides the implemented search strategies, and experimental results are reported in a fourth section; The last section relates our work to existing approaches, sums up our contribution and gives hints for further work.

Application domain

For each agent, the P&S module must find navigation plans and estimate passing dates, while satisfying coordination constraints with other agents. Each mission plan is composed of a set of totally ordered waypoints, for which a passing date must also be estimated. Agents plans and schedules must meet an objective and obey to terrain constraints. Global coordination between agents can be inforced by satisfying logical synchronisations on waypoints. In addition, the global mission plan should optimize a primary cost function, for instance mission duration, safety, security or observability. Without loss of generality, only mission duration is considered in this paper, that is, minimizing the maximal mission completion date for all agents.

Example

In Figure 1, both rescue AUGV and manned vehicules must perform a maximal exploration of villages (red circle) in a flooded area, looking for refugees and estimating damages. However vehicles must progress in a synchronized way for several operational reasons:

- Observations must be synchronized to avoid missing refugees;
- Operators in manned vehicles would want to see AUGV time to time in order to be able to switch to a teleoperation mode if needed;
- Communications between vehicles have to be maintained during mission progression.

In this scenario, search and rescue vehicles will start from location 1, and gather in the vicinity of node 20. All nodes circled in red have to be visited, where refugees and casualties are likely to be found. However, there are strong uncertainties concerning the time of traversal: On one hand, the manned vehicule need to master the AUGV execution in spite of uncertainty; On the other hand, the AUGV must adapt to the manned vehicule pace.

Architecture

Figure (2) presents a simplified C2 system architecture for an AUGV, with autonomous (robotic) capabilities. Building-up the situation awareness is based on various sensory data,

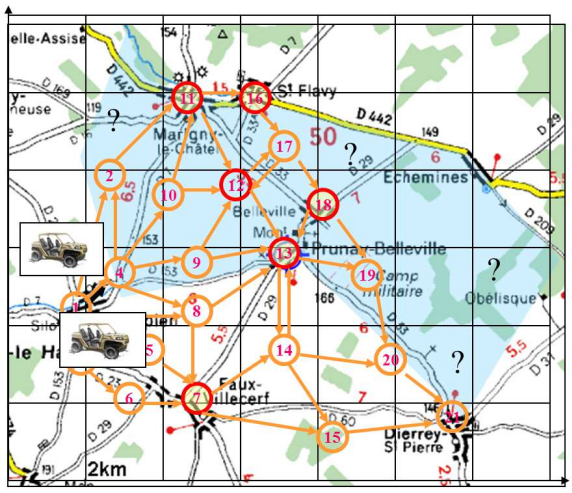


Figure 1: Search and rescue mission. Topological map for a manned vehicle and a AUGV throughout a flooded area between the Seine and the Vanne rivers in the area of Troyes, France. These specific vehicules, developped by SAFRAN E&D, can be either piloted or turned instantaneously into an AUGV.

local map and SLAM processing. These subsystems generate multiple sources of temporal and spatial uncertainty: drift, errors and bias. Moreover, time of traversal for some parts of the terrain is difficult to predict and is uncertain while planning the mission (in the example, moving in shallow water). Vehicular communications enable exchanges of mission plan as well as situation awareness (platform state such as position, velocity, time and list of observed objects). The vehicule can optionally interact with an operator or the pilot, but interactions definition are out of the scope of this paper. In addition to the vehicular P&S, we need a temporal uncertainy resolver module for several reasons:

- provide to the command staff go/no go decisions on whether the mission will continue;
- dynamically adapt the mission or trigger a replanning event;
- provide timing worst / nominal cases to the pilot or to the local operator;
- provide delays and arrival date estimates to other manned vehicules or AUGVs;
- adapt the execution controler to cope with potential delays and to maintain coordination with other vehicles.

A larger description of the tool is described in (Guettier 2007) and has been widely experimented. Example of detailed C2 system integration is described in (Guettier et al. 2011), while some fielded experiments are reported in (Guettier et al. 2009) (Guettier et al. 2015). The search algorithm baseline for a single agent is presented in (Guettier and Lucas 2016).

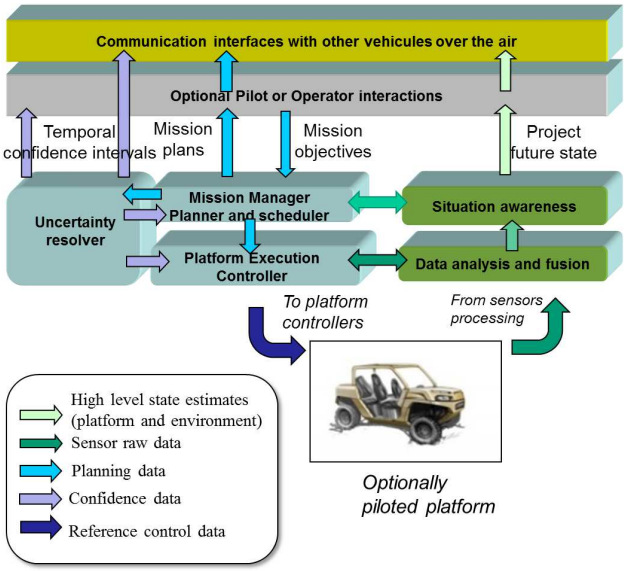


Figure 2: C2 system for optionally piloted vehicule (which can be piloted, remotely piloted, remotely operated or fully autonomous), with main components including P&S, communication, interactions and situation awareness. In spite of many sophisticated filtering techniques used to improve accuracy (location, observations) and reduce uncertainty, situation awareness is till an active area of research.

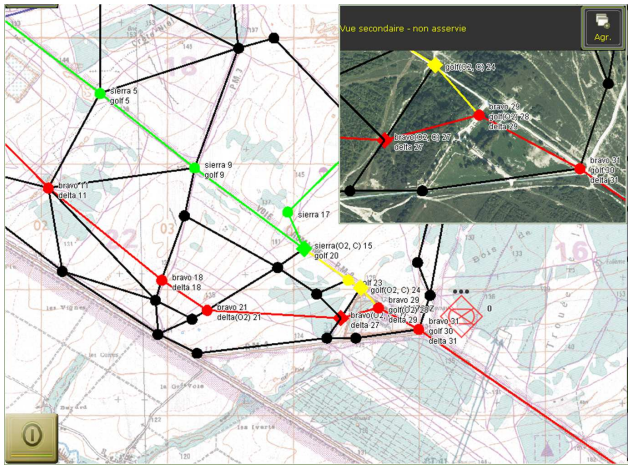


Figure 3: Operational vehicular C2 system integrating the ORTAC planner and displaying solutions. Agents paths are displayed in different colours to reach the central target. Plans are georeferenced on both satellite views and maps. The C2 systems is interfaced with inertial measurement units in order to provide locations with minimal uncertainty, even in GPS denied environments.

Model-based P&S with Uncertainty

In this section, we present a CSP-based P&S system and describe an extension to represent uncertainty.

The ORTAC P&S System

The system known as ORTAC is a model for a constraint satisfaction problem to compute paths of several agents using a directed graph, where vertices represent locations and edges represent routes (referred to as a topological map). With the given applications, graphs are defined during mission preparation, by terrain analysis and situation assessment. A path of an agent starts at a fixed initial vertex (starting location) and ends at a fixed final vertex (ending location), and is composed of a sequence of routes (i.e., chain of edges). The graph is maintained on-line during mission execution, by using fusion of sensory data (e.g., LIDAR, optronics). The P&S system also models duration and waypoint timing sequences, according to selected paths. Both manned and AUGV systems respond to an operator's (or mission commander's) demand by finding a route from a starting point to a destination, while visiting some mandatory waypoints.

In our approach, solving the P&S problem is achieved using Constraint Programming (CP) techniques, under a model-based development approach. CP is a competitive approach to solve such problems, providing completeness and optimality guarantees. With CP, a declarative formulation of the constraints to satisfy is provided which is decoupled from the search algorithms, so that both of them can be worked out independently. Both CSP formulation and search algorithms are implemented with the CLP(FD) domain of SICStus Prolog library (Carlsson 2015). It uses the state-of-the-art in discrete constrained optimisation techniques and Arc Consistency-5 (such as AC-5) for constraint propagation, managed by CLP(FD) predicates, as well as global constraints implementation.

Since more than one agent can be represented in ORTAC, the model also represents coordination among agents at given vertices of their respective paths. This is performed by expressing constraints relating two different agents on two different vertices: For instance, an agent must pass at a location *before* another agent passes at another location. Forced inclusion/exclusion of vertices/edges in a path of an agent can also be represented by additional constraints.

For each agent, the basic model relies on a graph where edges and nodes represent respectively ground mobility and accessible waypoints:

- a $\{0,1\}$ variable T_v on each vertex v, representing the fact that this vertex is included into agent path, i.e., the agent transits via vertex v;
- a $\{0,1\}$ variable $\varphi_{v,v'}$ on each edge v, v' representing the fact that the edge also belongs to the path, i.e., the agent transits from vertex v to vertex v';
- a flow constraint stating that an agent arriving at a vertex departs from it (with specific cases for the start and end vertices).

Constraint-based model for P&S More formally, a graph is a pair (V, U) where V is a set of vertices (or nodes) and U is a set of edges. Variables $\varphi_u \in \{0,1\}$ represent a possible path from $start \in V$ to $end \in V$, where an edge $u \in U$ belongs to the navigation plan iff $\varphi_u = 1$ (and 0 otherwise).

The resulting navigation plan Φ can be represented as $\Phi = \{u |\ u \in U,\ \varphi_u = 1\}$.

Path consistency from an initial position to a final one is enforced by flow conservation equations, where $\omega^+(v) \subset U$ (resp., $\omega^-(v) \subset U$) represents the outgoing (resp., incoming) edges from (resp., to) vertex $v \in V$.

$$\sum_{u \in \omega^+(start)} \varphi_u = 1, \quad \sum_{u \in \omega^-(end)} \varphi_u = 1, \tag{1}$$

$$T_v = \sum_{u \in \omega^+(v)} \varphi_u = \sum_{u \in \omega^-(v)} \varphi_u \leq 1 \tag{2}$$

Since flow variables φ_u are $\{0, 1\}$, equation (2) ensures path connectivity and uniqueness, while equation (1) imposes limit conditions for starting and ending the path. This constraint produces a linear chain of pass-by waypoints in a graph — waypoints are vertices of a topological map which are present in a path of a navigation plan Φ.

These waypoints v are labeled by passing time D_v depending on variables $V_{(v,v')}$ denoting the average velocity on edge (v, v') — this variable is within realistic ranges, depending on the physical minimum and maximum speeds of the robotic AUGV. The value $\|v, v'\|$ is the constant distance between two vertices v and v'. Variable $d_{(v,v')}$ represents the duration of traversal of an edge (v, v'), therefore the last 3 variables are related by equation (3) — the remaining variable $r_{v',v}$ is ignored, since it represents non-integer values of variables $V_{(v,v')}$ and $d_{(v,v')}$.

Given equation (3), passing times on waypoints are propagated via equation (4), which cumulates edge traversal duration along waypoints.

$$\|v, v'\| = V_{(v,v')}.d_{(v,v')} + r_{v,v'} \tag{3}$$

$$D_v = \sum_{(v',v) \in \omega^-(v)} \varphi_{(v',v)}(d_{(v',v)} + D_{v'}) \tag{4}$$

Resource consumption, security and observability can also be modelled by constraints.

Representing Uncertainty

Regarding agents transiting on a topological map, uncertainty can be temporal (uncertainty on the time at which an agent arrives to a vertex) or spatial (an agent can be located with latitude/longitude coordinates, not necessarily those of any vertex). This paper focuses on temporal uncertainty among locations, planned by agents.

For this, confidence intervals, a well-known representation of temporal uncertainty over the previous (certainty) variables, are introduced: A *confidence interval* of a finite-domain variable X in the above CSP model is an interval over integers $[X^{min}; X^{max}]$, represented by two finite-domain variables X^{min} and X^{max} associated to X. A *realization* of X is an instanciation of X which conforms to its confidence interval.

For example, a realization of the above duration variables $d_{(v,v')}$ on edges is an instanciation of all variables $d_{(v,v')}$ along the path, which conform to their confidence intervals $[d^{min}_{(v,v')}; d^{max}_{(v,v')}]$.

Velocity Model for Uncertainty Uncertainty is represented in the previous velocity model (recall equation (3) in the previous paragraph) by turning both the velocity and the duration variables into confidence intervals $[V^{min}_{(v,v')}; V^{max}_{(v,v')}]$ (refer to equation (5)) and $[d^{min}_{(v,v')}; d^{max}_{(v,v')}]$ (refer to equation (10)), and smilarly relating their lower bounds (refer to equation (6)) and upper bounds (refer to equation (7)) — as for the previous certainty model, the rounding variables $r_{v,v'}$ are ignored, since they correspond to non-integer values of confidence intervales on velocities, distances and durations.

A constant relative confidence interval $[\Delta V^{min}_{(v,v')}; \Delta V^{max}_{(v,v')}]$, where $\Delta V^{min}_{(v,v')}$ is a negative integer and $\Delta V^{max}_{(v,v')}$ is a positive integer, limits the possible expansion of the confidence interval for velocity $[V^{max}_{(v,v')}; V^{max}_{(v,v')}]$ (refer to equations (8) and (9)).

$$V^{min}_{(v,v')} \leq V_{(v,v')} \leq V^{max}_{(v,v')} \tag{5}$$

$$\|v, v'\| = V^{min}_{(v,v')}.d^{min}_{(v,v')} + r^{min}_{v,v'} \tag{6}$$

$$\|v, v'\| = V^{max}_{(v,v')}.d^{max}_{(v,v')} + r^{max}_{v,v'} \tag{7}$$

$$V_{(v,v')} + \Delta V^{max}_{(v,v')} \leq V^{max}_{(v,v')} \tag{8}$$

$$V_{(v,v')} + \Delta V^{min}_{(v,v')} \geq V^{min}_{(v,v')} \tag{9}$$

$$d^{min}_{(v,v')} \leq d_{(v,v')} \leq d^{max}_{(v,v')} \tag{10}$$

The ORTAC model also represents the duration S_v of an action performed by an agent at a waypoint v, in addition to its passing time D_v — agents not only passes at waypoints but perform durative actions there. Since duration and velocity on an edge are uncertain, representing uncertainty also involves turning variable D_v into a confidence interval $[D^{min}_v; D^{max}_v]$ on each vertex v. The realization of variable D_v in its confidence interval is represented by equation (11).

Representing uncertainty on vertices relates the arrival time D_v at waypoint v to the time of arrival $D_{succ(v)}$ at the next waypoint $succ(v)$ of the same path, using the confidence interval bounds (12 and 13) — uncertainty over confidence interval never decreases along path following, hence the direction of the two inequalities.

$$D^{min}_v \leq D_v \leq D^{max}_v \tag{11}$$

$$D^{min}_{succ(c)}) \geq D^{min}_v + d^{min}_{(v,succ(v))} + S_v \tag{12}$$

$$D^{max}_v + d^{max}_{(v,succ(v))} + S_v \leq D^{max}_{succ(v)} \tag{13}$$

Uncertain Coordination among Agents The ORTAC model can represent not only one agent traversing a topological map, but several agents: this is performed by iteratively defining the finite-domain variables representing each agent behavior and constraint postings, all indexed by each agent — a loop over variable definitions and constraint postings, indexed by agent (that is, two different missions on the same scenario and the same agents, but with different coordinations, imply slightly different CSP models). As such, it seems natural to represent coordination among these agents, using additional constraints. For example, in a disaster recovery scenario involving a damaged village (as in Figure

1), one or more agents look for refugees in the surrounding outside while another agent searches for casualties inside.

For this, ORTAC represents temporal coordination constraints between two agents on two vertices, for which the semantics is informally defined as follows (refer to (Guettier 2007) for the logical semantic definition):

- *before*: agent A performs its action on its vertex before agent B performs its action on its other vertex, within a time window;
- *after*: agent A performs its action on its vertex after agent B performs its action on its other vertex iff agent B before agent A;
- *simultaneous*: agents A and B perform their respective actions on their respective vertices during the same period of time;
- *disjunct* : agent A is disjunct from agent B on vertex v iff agent A is passing before agent B or agent B is passing before agent A on vertex v.

In order to represent temporal uncertainty into these coordination constraints, the confidence intervals of the previous section must also be considered into the above coordination formulation. Let A and B be two different agents transiting in a topological map:

- Agent A is *uncertainly simultaneous* to agent B iff their respective confidence intervals exactly overlap (see equation (14)).
- Agent A is *uncertainly disjunct* from agent B iff the upper bound of the confidence interval of agent A is less than the lower bound of the confidence interval of agent B (including the duration of the action performed by agent A on the vertex), or the opposite by switching A and B (see equation (15)).

Formally, given two confidence intervals $[D_v^{min}(A); D_v^{max}(A)]$ denoting the passing time of agent A at waypoint v, and $[D_{v'}^{min}(B); D_{v'}^{max}(B)]$ similarly for agent B at waypoint v', the following uncertain coordination formulations can be written:

$$D_v^{min}(A) = D_{v'}^{min}(B) \wedge D_v^{max}(A) = D_{v'}^{max}(B) \quad (14)$$

$$D_v^{max}(A) + S_v(A) \leq D_{v'}^{min}(B) \vee$$
$$D_{v'}^{max}(B) + S_{v'}(B) \leq D_v^{min}(A) \quad (15)$$

$$D_v^{min}(A) \leq D_{v'}^{min}(B) \wedge D_v^{max}(A) \leq D_{v'}^{max}(B) \quad (16)$$

$$D_v^{max}(A) + S_v(A) \leq D_{v'}^{min}(B) \quad (17)$$

Since temporal intervals are represented in addition to time points, the certain *before* coordination constraint is turned into *weak* (see equation (16)) or *strong* (see equation (17)) *uncertainly before* coordination constraints, depending on the existential (see equation (18)) or universal (see equation (19)) quantifier used for the realization of the variables $D_v(A)$ and $D_{v'}(B)$ for the synchronization between agents A and B. The difference between the weak and strong uncertainly before constraints can also be considered as enabling/forbidding overlaps between the two confidence intervals, which can be formalized with temporal intervals (see relations "before" and "overlaps" (Allen 1983)).

$$weak\_synchronisation\_before(A, B) \Leftrightarrow$$
$$\forall v \in V, \exists (D_v(A), D_{v'}(B))$$
$$\in [D_v^{min}(A); D_v^{max}(A)] \times [D_{v'}^{min}(B); D_{v'}^{max}(B)]$$
$$s.t. D_v(A) \leq D_{v'}(B) \quad (18)$$

$$strong\_synchronisation\_before(A, B) \Leftrightarrow$$
$$\forall v \in V, \forall (D_v(A), D_{v'}(A))$$
$$\in [D_v^{min}(A); D_v^{max}(A)] \times [D_{v'}^{min}(B); D_{v'}^{max}(B)]$$
$$s.t. D_v(A) \leq D_{v'}(B) \quad (19)$$

Finaly, the same formal model and constraints are defined for the *after* uncertain coordination constraint, by switching agents A and B in the previous *before* uncertain coordination model.

Compound search algorithms

Two search algorithms are considered, one to solve the initial coordinated P&S problems, and then one to solve the temporal uncertainty resulting from the coordinated paths.

Solving the P&S problem with ORTAC

The global search technique under consideration guarantees completeness, solution optimality and proof of optimality. It relies on three main algorithmic components:

- Variable filtering with correct values, using specific labeling predicates to instantiate problem domain variables. the constraint propagator being incomplete, value filtering guarantees the search completeness.
- Tree search with standard backtracking when variable instantiation fails.
- Branch and Bound (B&B) for cost optimisation, using minimise predicate.

Designing a good search technique consists in finding the right variables ordering and value filtering, accelerated by domain or generic heuristics. A static probes provides an initial variable selection ordering, computed before running the global branch and bound search (Guettier and Lucas 2016). In the approach, the variable selection order provided by the probe can still be iteratively updated by the labeling strategy that makes use of other variable selection heuristics. In general, dynamic probing techniques use solutions to some relaxations of the original problem and consider these 'partial' solutions as tentative values, see for example (Sakkout and Wallace 2000) and (Ruml 2001). In ORTAC, the search strategy uses a static prober which orders problem variables before the search. This ordering is based on the relations between problem structure and the partial solution found. Then, the solving relies on a standard CP branch and bound search strategy, combining variable filtering, AC-5, generic heuristic and B&B. The probing technique proceeds in three steps:

- Establish the relaxed problem, abstracting away mandatory waypoints and coordination constraints.

- Compute a shortest reference path between starting and ending vertices, using Dijkstra or A*.
- Establish a minimal distance between any problem variable and the solution to the relaxed problem.

The last step considers the following distance between partial solution values X_s and all original problem variables X.:

$$\forall x \in X, \delta(x) = \min_{x' \in X_s} ||(x, x')|| \quad (20)$$

where $||.||$ is the distance metric, corresponding to the number of vertices between x and x'. The last step uses the resulting partial order to sort problem variables in ascending order, using $\delta(x)$. Problem variables are explored following that order in the global search. The probe construction is polynomial and does not change completeness nor optimality properties of the global branch and bound loop.

Search with Uncertainty

As explained above, confidence intervals are intervals over integers representing temporal uncertainty at planning time around an exact (certain) planned integer value. That is, involving confidence interval $[D_v^{min}(X), D_v^{max}(X)]$ of passing time $D_v(X)$ of agent X on vertex v, in which $D_v^{min}(X) \leq D_v(X) \leq D_v^{max}(X)$.

We follow this definition by using a labeling search on uncertainty after the labeling search on the exact (certain) value of the passing time $D_v(X)$ of agent X at each vertex v. Hence, paths and passing times on vertices are known (i.e., $D_v(X)$ finite-domain variables are instantiated) before search on uncertainty is performed.

In order to increase performances, a static heuristic on variables is used: If a path of length n is composed of waypoints $v_1, v_2 \ldots, v_i, \ldots v_n$, this heuristic reorders variables $D_{v_i}^{min}(X)$ and $D_{v_i}^{max}(X)$ according to the path from start to end in the forward direction, by increasing i. This heuristic on variables considers the uncertainty variables $D_{v_i}^{min}(X)$ and $D_{v_i}^{max}(X)$ for agent X in the following order (21):

$$D_{v_1}^{min}(X) \prec D_{v_1}^{max}(X) \prec D_{v_2}^{min}(X) \prec D_{v_2}^{max}(X)$$
$$\ldots D_{v_i}^{min}(X) \prec D_{v_i}^{max}(X) \cdots \prec D_{v_n}^{min}(X), D_{v_n}^{max}(X)) \quad (21)$$

Since the confidence interval $[D_{v_1}^{min}(X); D_{v_1}^{max}(X)]$ at the starting location v_1 of agent X is known, labeling is sufficient to instantiate these confidence intervals along the path. If a coordination constraint creates an empty domain of any finite-domain variable $D_{v_i}^{min}(X)$ or $D_{v_i}^{max}(X)]$ for any agent X on any vertex v_i, CP backtracking occurs inside the search on uncertainty (i.e., on confidence intervals) and then possibly inside the search on certainty (i.e., on exact variables) — finding other confidence intervals for the same realization of $D_{v_i}(X)$, or finding another realization.

Experimental results

Benchmarks

Experiments on four benchmarks are presented, which are representative of peace keeping missions or disaster relief.

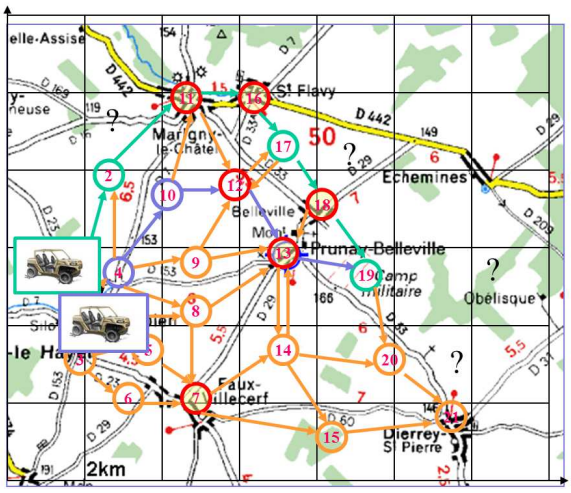


Figure 4: Paths solution for coordinated manned vehicle and the AUGV

Missions must be executed in less than 30 minutes. Areas range from 5x5 kms to 20x20 kms.

1. Recon villages: Observing different villages after a major water flooding event, described as a running example in Fig. (1) and for which a solution to a 2 agents problem is given in Figure (4);
2. Reinforce UN: Bring support to a United Nations mission by deploying observers in an unsecure town;
3. Sites inspections: Observing different parts of a town during inspection of suspect sites;
4. Secure humanitarian area: Observing different threats before securing refugees, over a large area.

On the first benchmark, Figure 4 shows the two paths found by the first P&S algorithm. Resolving uncertainty then provides the confidence intervals given in Figure 5 for the two coordinated agents.

Performances of the Uncertainty Model

In order to measure the additional computational cost of the solving process due to the uncertainty model, ORTAC has been run on 4 topological maps, composed of 22 / 33 / 23 / 22 vertices and, respectively, 74 / 113 / 76 / 68 edges. Each example involves 2 to 8 agents. The experiments were carried out on a computer with processor i7 at 2GHz with 4Gb RAM on a virtual machine. The computation time is measured for the certainty search and for the uncertainty one — see Figure 6. Further experiments have been carried out with a topological map representing the streets and intersections of Paris: solving time takes more than 2 hours under the same experimental conditions.

Related work and Discussion

First, Nilsson et al. (Nilsson, Kvarnström, and Doherty 2015) define Simple Temporal Networks with Uncertainty (STNUs) as an extension of Simple Temporal Networks

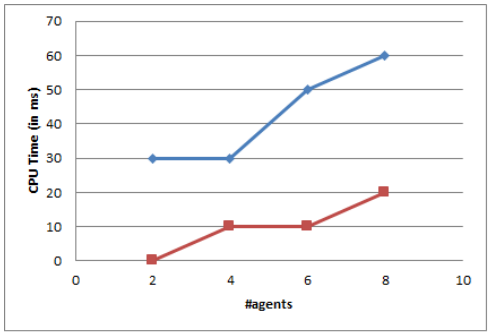
```
- Uncertain coordination simultaneous between:
unit1 on node 11 and unit2 on node 12
--- Agent : unit1
 Absolute uncertainty on node 2 : -2 =< 0 =< 3
 Absolute uncertainty on node 11 : 10 =< 32 =< 35
 Absolute uncertainty on node 16 : 55 =< 77 =< 80
 Absolute uncertainty on node 17 : 59 =< 81 =< 84
 Absolute uncertainty on node 18 : 65 =< 87 =< 90
 Absolute uncertainty on node 19 : 71 =< 93 =< 96

--- Agent : unit2
 Absolute uncertainty on node 1 : -2 =< 0 =< 3
 Absolute uncertainty on node 4 : 2 =< 4 =< 7
 Absolute uncertainty on node 10 : 5 =< 7 =< 10
 Absolute uncertainty on node 12 : 10 =< 12 =< 35
 Absolute uncertainty on node 13 : 16 =< 18 =< 41
 Absolute uncertainty on node 19 : 82 =< 84 =< 107
 Absolute uncertainty on node 20 : 88 =< 90 =< 113
```

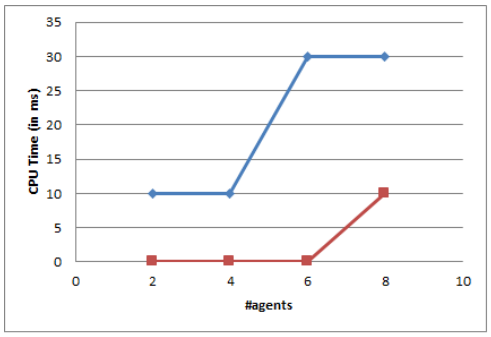
Figure 5: Excerpt of output of ORTAC for 2 units "unit1" and "unit2" with the coordination constraint "simultaneous" between vertices 11 for unit1 and 12 for unit2. Each line shows the lower bound of the confidence interval $D_v^{min}(X)$, the exact (certain) passing time $D_v(X)$ on each vertex v, and the upper bound of the same confidence interval $D_v^{max}(X)$. Times are given in minutes and progression time in search and rescue is expressed in meters per minute.

(STNs) (Dechter, Meiri, and Pearl 1991) towards representing uncertainty — this has been extended towards continuous uncertainty with Probabilistic STNUs (Santana et al. 2016). A temporal action in a STNU is represented as start and end times, with a bounded duration: for every temporal action A, $duration(A) = end(A) - start(A) \in [min(A), max(A)]$. These authors propose an algorithm with $O(n^3)$ complexity to incrementally verify that there always exists a solution for the start and end times of each action (dynamic controllability), regardless of what happens at execution time — these start and end times are constrained by uncontrollable/contingent phenomena (e.g., wind, weather). In contrast, our approach does not consider one agent only, as with STNUs, but several, which is modelled by a flow constraint (refer to equation (2)). As such, our model can represent coordination constraints among agents (crucial for our application on AUGVs), which cannot be represented by STNUs' binary constraints. A common ground between STNUs and our approach would be to define a CP global constraint, called dynamic controllability verification, to ensure consistency of a subset of our CP constraints model.

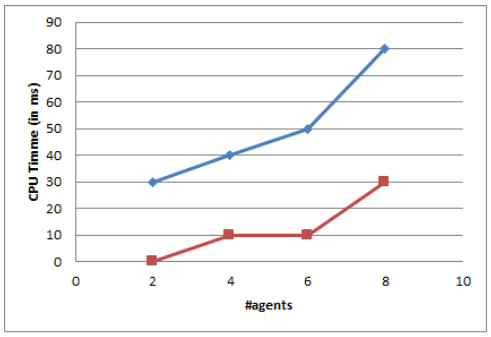
Second, Fargier et al. (Fargier, Lang, and Schiex 1996) extend the CSP framework to deal with reasoning under incomplete knowlege: they propose an anytime algorithm (implemented in (Guettier and Yorke-Smith 2005) for an application in the aerospace domain) based on a set X of uncontrollable variables and on another set Y of controllable variables — hence its name *mixed*-CSP. The algorithm proposed by these authors covers realizations of variables of X, one by one, with CSP resolution over variables of Y and iterates on realizations until they are all covered. This algorithm exhibits an anytime property, since uncontrollable variables are considered first one by one: interrupting this algorithm leaves covered a subset of X. In contrast, our approach is based on uncertainty by extending a certainty reasoning, as



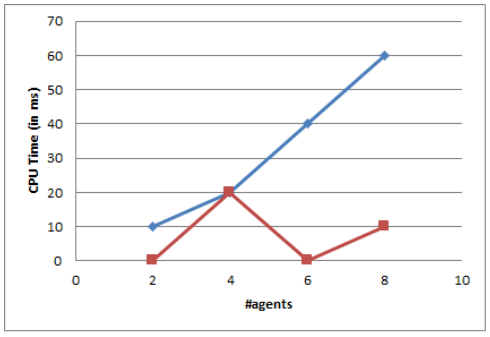
6.a Recon village after flooding



6.c Suspect sites inspection



6.b Reinforce UN in town



6.d Secure humanitarian area

Figure 6: Performance on benchmarks according to the number of agents and one coordination constraint per run: execution time in blue and red, respectively for the reference P&S problem, and the scheduling under uncertainty

STNU extends STN, whereas mixed-CSP considers uncontrollability first and then responds to it by controllability — an approach which suffers from severe algorithmic complexity.

Third, one could argue that mixed-integer programming (MIP), instead of CP, could be used to solve our model. That is, equation (2) would be interpreted as an integrity equation, common in MIP, whereas the rest of the model would be turned into linear inequalities among variables on integer or real values. Unfortunately, our velocity model is not linear but quadratic (refer to equation (3)).

However, following this idea anyway, our model is based on finite-domain variables (i.e., on variables over integers), as in every CSP, and it would be interesting to mix integers and real numbers, as in MIP. For example, for representing continuous values of temporal variables in our model, such as passing time D_v at waypoint v or duration S_v. Indeed, the implementation language, Sicstus Prolog, includes a continuous solver (Carlsson 2015), but that latter solver and the CSP solver hardly cooperate. A more interesting approach towards mixing discreteness and continuity in CP is the CSP solver CHOCO (Prud'homme, Fages, and Lorca 2017), harmoniously integrated to the continuous solver IBEX (Chabert and Jaulin 2009). But porting ORTAC onto these two solvers would entail large software engineering work.

Finally, the incremental property of STNU's verification algorithm and the anytime property of mixed-CSP are interesting, which would lead in our context to what could be called *anytime CSP*, meaning interrupting a CSP solver before completion and having a partial solution where some quality would increase over the allocated time. But that would be another story — after all, time that passes can also be considered as an uncontrollable continuous variable.

Conclusion

A discrete representation of temporal uncertainty based on confidence intervals in a CSP-based planning and scheduling system has been presented. This extends a system known as ORTAC (Guettier 2007) which finds paths in a topological map for multiple agents with coordination constraints — its applications include planning paths of tactical units in a wargame, finding routes in a road network while minimizing consumed energy and planning medical visits of patients. Early experiments show that adding an uncertainty model to a certainty one does not significantly degrade the solving performances of the whole system.

Future work includes: Considering a higher level language inspired by ANML (Smith, Franck, and Cushing 2008), which seems more appropriate than PDDL (McDermott et al. 1998) for P&S robotic applications (Dvorak et al. 2014); And connecting ORTAC to a wargame simulating AUGVs, before porting the system to AUGVs for real.

Acknowledgments

The authors thank Jean-Francois Tilman (SAFRAN E&D) for numerous fruitful discussions and anonymous reviewers for helpful suggestions.

References

Allen, J. 1983. Maintaining knowledge about temporal intervals. *Communications of the ACM* 832–843.

Carlsson, M. 2015. *SICSTUS Prolog users manual*.

Chabert, G., and Jaulin, L. 2009. Contractor programming. *Artificial Intelligence* 173:1079–1100.

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

Dvorak, F.; Bit-Monnot, A.; Ingrand, F.; and Ghallab, M. 2014. Plan space hierarchical planning with the action notation modelling language. In *ICTAI*.

Fargier, H.; Lang, J.; and Schiex, T. 1996. Mixed constraint satisfaction: a framework for decision problems under incomplete knowledge. In *AAAI*. AAAI Press.

Guettier, C., and Lucas, F. 2016. A constraint-based approach for planning unmanned aerial vehicle activities. *The Knowledge Engineering Review* 31(5):486497.

Guettier, C., and Yorke-Smith, N. 2005. Enhancing the anytime behavior of mixed csp-based planning. In *ICAPS Workshop on planning under uncertainty for autonomous systems*.

Guettier, C.; Sechaud, P.; Yelloz, J.; Allard, G.; Lefebvre, I.; Peteuil, P.; Ponthereau, P.; Cuisinier, F.; and Martinet, J. 2009. Improving tactical capabilities with net-centric systems: the phoenix'08 experimentation. In *Proceedings of IEEE MILCOM Military Communications Conference*.

Guettier, C.; Yelloz, J.; Cherrier, O.; Mayk, I.; and Lamal, W. 2011. Interoperable joint planning and execution web service with titan. In *MILCOM 2011 Military Communications Conference*, 20252030.

Guettier, C.; Lamal, W.; Mayk, I.; and Yelloz, J. 2015. Design and experiment of a collaborative planning service for netcentric international brigade command. In *IAAI*.

Guettier, C. 2007. Solving planning and scheduling problems in network based operations. In *Proceedings of Constraint Programming (CP)*.

Hart, P.; Nilsson, N.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems, Science and Cybernetics* 4(2):100–107.

Laurière, J.-L. 1978. A language and a program for stating and solving combinatorial problems. *Artificial Intelligence* 10:29–127.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ashwin, R.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. Pddl — the planning domain definition language. Technical report, Yale Center for Computational Vision and Control, New Haven, CT. Technical Report CVC TR98003/DCS TR1165.

Montanari, U. 1974. Networks of constraints: Fundamental properties and applications to picture processing. *Information Sciences* 7:95–132.

Nilsson, M.; Kvarnström, J.; and Doherty, P. 2015. Efficient processing of simple temporal networks with uncer-

tainty: Algorithms for dynamic controllability verification. *Acta Informatica*.

Prud'homme, C.; Fages, J.-G.; and Lorca, X. 2017. *Choco Documentation*. TASC - LS2N CNRS UMR 6241, COSLING S.A.S.

Ruml, W. 2001. Incomplete tree search using adaptive probing. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, volume 1, 235–241. Morgan Kaufmann Publishers Inc.

Sakkout, H. E., and Wallace, M. 2000. Probe backtrack search for minimal perturbations in dynamic scheduling. *Constraints Journal* 5(4):359–388.

Santana, P.; Vaquero, T.; Toledo, C.; Wang, A.; Fang, C.; and Williams, B. 2016. Paris: a polynomial-time, risk-sensitive scheduling algorithm for probabilistic simple temporal networks with uncertainty. In *ICAPS*.

Simonin, G.; Artigues, C.; Hebrard, E.; and Lopez, P. 2015. Scheduling scientific experiments for comet exploration. *Constraints* 20:77–99.

Smith, D. E.; Franck, J.; and Cushing, W. 2008. The anml language. In *ICAPS Workshop on Knowledge Engineering for Planning and Scheduling (KEPS)*.

Extensions of a Simple Temporal Network Coordinating Emergent Knowledge Processes in a Collaborative System-of-Systems

Michael A. Schaffner
Sandia National Laboratories
P.O. Box 5800, Albuquerque, NM 87185-1192
maschaf@sandia.gov

Abstract

The Z Machine is the world's most powerful x-ray source, routinely delivering over 20 MA of electrical current to targets in support of US nuclear stockpile stewardship and in pursuit of inertial confinement fusion. The large-scale, multi-disciplinary nature of experiments ("shots") on the Z Machine requires resources and expertise from disparate organizations to execute Emergent Knowledge Processes with independent functions and management, forming a Collaborative System-of-Systems. A previous work identified the resulting significant challenges of distributed planning and coordinating a given experiment day, and described one potential approach to scheduling based on a Simple Temporal Network with only minimum times between activities defined. The present work extends that approach in two ways. First, a method is proposed to establish latest cutoff times for activity starts through the setting of a single operational goal to back-schedule all latest times of when activities might begin to achieve that goal (so that unlike the lower bounds which are physically possible intervals, the upper bounds reflect operationally required times if the goal is to be achieved). Second, the present work implements a real-time web-based software tool to enable more informed planning of "shot day" activities and presenting information relevant on shot day to aid as an enabling interface between workers among the varied groups involved in planning and execution. The resulting software product is a scheduling tool that displays windows of time during which each activity could and should begin. The software's initial results are evaluated, and future areas for improvement are discussed.

Introduction

"Thou goest thine, and I go mine – many ways we wend;
Many days, and many ways, Ending in one end."
-George MacDonald, *Phantastes*

The Z Machine (hereafter "Z") is the world's most powerful x-ray source, routinely delivering over 20 MA of electrical current to targets in support of various programs, including US nuclear stockpile stewardship and pursuit of inertial confinement fusion. A single experiment (or "shot") requires months of planning, design work, specialized hardware fabrication, and diagnostics configuration, all involving experts from a variety of specialized back-grounds such as plasma physics, hydrodynamics, dynamic material properties, laser technologies, atomic spectroscopy, neutron diagnostics, electrical engineering, mechanical engineering, and electro-mechanical controls, among others. Execution of a shot can often be achieved in one day; regular operation of Z on a daily basis requires specialists from all of the fields above as well as technicians and installers performing regular machine maintenance and configuration. These personnel are involved in activities ranging from operating heavy machinery to refurbishing equipment, performing routine mechanical and electrical work, and even underwater diving.

A previous work (Schaffner 2017) identified Z and its participants as a Collaborative System-of-Systems (SoS) (Maier 1998) replete with Emergent Knowledge Processes (EKPs) (Markus et al. 2002), exhibiting independent management and operation, volunteer-like participation, and unpredictable and emergent arrangements of people and systems. All of these traits create significant challenges to planning and scheduling activities for a given Z experiment (which can take anywhere from half a day to multiple days), which in turn create significant challenges to coordination of the various participants involved in the experiment – especially as configurations are redefined and as new needs emerge over the experiment's preparation and execution. This latter emergent condition is common, given the nature of EKPs, and yet it is also the most likely to cause previously communicated planning and scheduling information to become obsolete, whether or not all participants are aware of the developments.

Motivation: Aiding coordination of EKPs with an information system

Despite these challenges posed to planning and scheduling, "encouraging cooperation" of participants remains the primary goal of the present work in keeping with Maier's (1998) architectural principles for an SoS; efficient coordination is one of the means by which the present work attempts to encourage cooperation. In the present case, efficient coordination is enabled through an information system intended to provide a higher-level understanding of the anticipated and actual temporal behavior for a given experiment, which respectively reflect two of Maier's (2005) research challenges for the study of an SoS: Upper Layer Description and Upper Layer Analysis. Rhodes and Ross (2010) identify some of the behavioral and perceptual factors at play when attempting to describe the complexity of systems like Z, as described in (Schaffner 2017): success can only be achieved when the temporal behavior of a Z experiment is captured and presented in a way that can account for the varying perceptions of what that behavior means for individual participants, ultimately encouraging cooperation from each individual participant. Similarly, (Markus et al. 2002) state that since "the mix of backgrounds and expertise brought to bear on emergent knowledge processes can differ each time the process is performed," a meta-requirement for information systems that support EKPs "…must meet the diverse and some-times contradictory needs of different user groups." In addition, since "no one individual or group has a complete grasp of both the general and specific knowledge that applies," a supporting information system "…must incorporate the frameworks and perspectives of several different kinds of participants" (Markus et al. 2002).

Method

Schaffner (2017) discussed some of the challenges to both the feasibility and usefulness of probabilistic information about scheduled activities for a Z experiment, concluding that communicating earliest time estimates (ETEs) instead of "likely" (or any other probabilistic) time estimates is the most effective approach to wide-scale coordination of independent agents in environments with large exogenous uncertainties (such as EKPs). Mass transit systems (e.g., planes, trains, buses) engage in widespread application of this idea of communicating earliest times for more efficient coordination of independent participants. However, stakeholders and research readers will still inevitably raise the question: why not use triangle distributions, or beta distributions, or PERT, or [insert favorite probabilistic method here] to describe each activity in the Simple Temporal Network, and therefore communicate so much more forecasting information to participants?

An evaluation of Earliest Time Estimates (ETEs) and probabilistic time estimates in EKPs

While Simon (1992) nicely sums up the big picture of why probabilistic methods are not appropriate in this context ("the heart of the data problem for design is not fore-casting but constructing alternative scenarios for the future"), Markus et al. (2002) offer several specific, relevant meta-requirements for information systems that support EKPs. These meta-requirements are presently described and used to evaluate the appropriateness of earliest times vs. probabilistic times for coordination of participants in EKPs.

Meta-requirement: Because "knowledge must be actionable in application," an information system supporting EKPs "must be directed at improving off-line behavior and must tie knowledge to concrete practical action" (Markus et al. 2002). The question then arises, how is concrete practical action enabled by a probabilistic estimate of an activity's scheduled time? How should (or will) a participant change their actions if they receive an estimate of 30% confidence as opposed to 60% confidence, for example? Keeping in mind that participants each have their own cognitive biases, independent management, and voluntary participation with other participants, can they be expected to respond consistently, much less uniformly, to such information? (In addition, in a Collaborative SoS, who could enforce uniform responses even if they were desired by some participants?) Providing earliest times, on the other hand, provides the concrete action of "checking in" – communication is simple in the modern age, and a quick "check-in" is all it takes for a participant to find out whether to start an activity or check-in at a future time. With an ETE, therefore, a participant can be consistently prepared to begin an activity when needed, rather than showing up late because it was "not probable" that they'd be needed. In addition, participants can be off-line as much as they like after receiving an ETE, since (ideally) the estimate will not be invalidated by any future updates to that ETE.

Meta-requirement: Because "it is not possible to know in advance who will be involved" in an EKP, a supporting information system "must employ terms, operations, and an interface that are usable by participants who are un-known in advance" (Markus 2002). Setting aside the obvious (and significant) problems with relying on probabilistic information about processes that include unknown participants, the fact that unknown participants may be present compounds the challenges of the previous meta-requirement. Not only would the unknown participants need to be able to use the existing information effectively, but they would need to be able to provide their own probabilistic information so that the information system could update the prob-

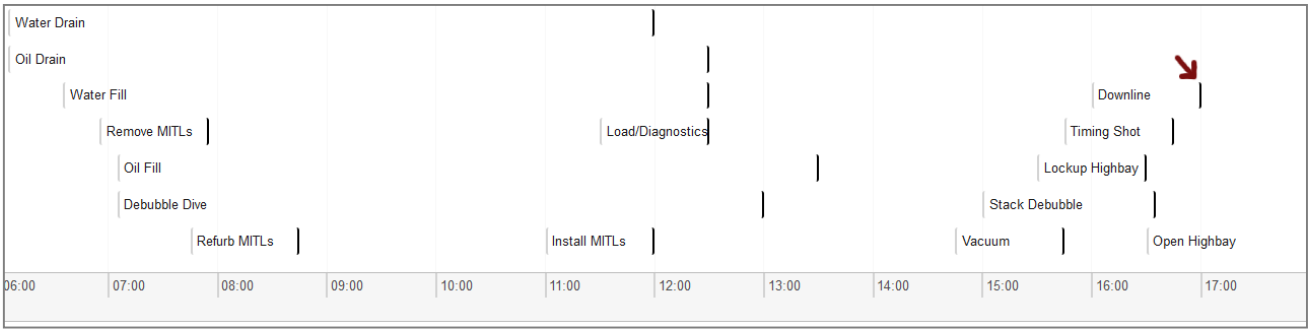


Figure 1. ETEs of activities (denoted by gray lines on each activity's left-hand side) are derived by scheduling the earliest times of activities starting at 0600. LCTs (black lines on right-hand sides) are derived by choosing a goal for Downline of 1700 (indicated by bold arrow), which allows minimum bounds to be back-propagated through the network to generate LCTs for all other activities.

abilistic information about all following activities (likely invalidating the previously calculated values). It remains unclear how or when such information would be obtained from such participants (not to mention how such collection would be enforced), and it remains especially unclear how such information could be validated (since, as (Markus et al. 2002) point out, many participants "may participate only infrequently or in circumstances that do not reoccur"). ETEs, on the other hand, are agnostic to unknown participants, since the ETEs can always be pushed later in time without invalidating previously calculated ETEs of an activity or any of its successors.

Meta-requirement: Because participants in EKPs "may have considerable discretion over their use of methods and tools," a supporting information system must ensure "that decisions are easier to make with the system than without it." With probabilistic information, a participant could be confused, misled, or have other malformed cognitive functioning with respect to their actions in support of scheduled activities, making their real-time decisions potentially much more stressful and much more difficult to later analyze reflectively. In contrast, ETEs make decisions easier for every participant, since it is straightforward to infer when they might need to check-in in the future.

The meta-requirements above demonstrate the challenges to and ultimate insufficiency of providing probabilistic information to Z SoS participants as a means of effecting efficient coordination. They also demonstrate the appropriateness of providing ETEs in the context of Z's SoS, where EKPs are found throughout an experiment's lifecycle.

Providing Latest Cutoff Times (LCTs)

Schaffner (2017) raises the question of whether latest time estimates for activities could be provided in the same manner as ETEs, since Simple Temporal Networks naturally support both constructs. The meta-requirements above demonstrate that in addition to providing ETEs, providing latest time estimates for activities could also help participants (e.g., through enabling more appropriate off-line behavior and more concrete action). The SoS-style collaboration in EKPs in Z experiments, however, implies that upper bound estimates on experimental activities' durations are dubious at best, and misleading at worst. For many of the same reasons as with ETEs, probabilistic estimates cannot help here.

If lower bound temporal relationships between activities are established, however, then latest cutoff times (LCTs) for those activities could be derived through backward propagation of the lower bounds from some future "goal" time (i.e., back-scheduling). Since in the case of Z experiments all activities eventually lead to a Z Shot, an operational goal for the activity of shooting the machine can be de-fined (e.g., "Fire the machine by 5pm today"), which can be back-scheduled through the network to provide the latest cutoff times (LCTs) for all preceding activities[1]. An illustration of ETEs and LCTs can be seen in Figure 1.

Example derivation of ETE and LCT

Let G be the directed edge-weighted graph defined as a tuple **G** := <**V**, **E**>:

V: set of nodes, each representing the start of an activity (e.g., "Begin Water Fill")

E: set of edges ω_{ij} representing the minimum minutes between nodes, of form $j_{startTime} - i_{startTime} > \omega_{ij}$, where

j,i ∈ **V**

i = source node for the directed edge

j = destination node for the directed edge

$\omega_{ij} \in \mathbb{R} > 0$

No cycles exist.

[1] The operational goal could also be mixed with other operational constraints (e.g., upper bounds of resource availability for individual activities) to incorporate more complete information on LCTs, but for present purposes only the shot event is considered.

Once **G** is constructed, the ETEs can be derived by scheduling all activities at their earliest times. To do so, the network is first ordered topologically using Kahn's (1962) algorithm, and then nodes without predecessors are assigned an ETE[2]. Successive nodes are then assigned ETEs in topological order through the calculation of:

$$ETE_j = \max \{ ETE_i + \omega_{ij}, \forall i, \omega_{ij} \in \boldsymbol{E}^-(j) \} \quad (1)$$

In other words, a node's ETE can be assigned by adding each incoming edge ω_{ij} (edge weight, in minutes) to that edge's source node's ETE, and taking the maximum (i.e., latest time) of this calculation for all incoming edges ($\boldsymbol{E}^-$).

Once the ETEs have been assigned, the latest cutoff time (LCT) for the goal node can be established (e.g., "5pm" for the Z Shot):

$$LCT_{ZShot} = 5pm$$

and then LCTs for all preceding activities can be assigned through back-propagation, or the inverse of the ETE calculations in Equation 1:

$$LCT_i = \min \{ LCT_j - \omega_{ij}, \forall j, \omega_{ij} \in \boldsymbol{E}^+(i) \} \quad (2)$$

The basic idea of Equation 2 is that a node's LCT can be assigned by calculating, for each of its outgoing edges ($\boldsymbol{E}^+$), the edge weight (in minutes) subtracted from the edge's destination node's LCT; the minimum time calculated from all outgoing edges ($\boldsymbol{E}^+$) is then assigned as the node's LCT. By beginning at the goal node (Z Shot) and working topologically backward, each node in the network is assigned its LCT after all its successors' LCTs have already been determined.

In this way, the STN from Schaffner (2017) can be extended to also include maximum times for activities, but unlike the lower bounds between activities derived from the *minimum physically possible* times, the upper bounds are derived from the *maximum operationally desirable* times (i.e., without sacrificing the overall operational goal for the day). The resulting network is similar to a Simple Temporal Network with Uncertainty (Vidal 1999), as it implicitly reflects "strong controllability" – though with the important difference that, in the present application, it is still entirely possible for activities' durations to violate the upper bounds, which violation would simply indicate that the operational goal can no longer be reached by the desired time given the structure of the network.

[2] Initial experiment start time or estimated time of resource availability can be used to determine ETEs for source nodes (i.e., no predecessors).

(More) Functional Distributed Reasoning

Through the construction of ETEs and LCTs, then, the present work aids in the goal of Schaffner (2017): that of participants' functional reasoning laid out by both Simon (1992) and Markus (2002), leading to more efficient coordination. While the present work extends the information incorporated and presented to the reasoning agents, it still continues the two means set out by Schaffner (2017):

1) Require as little information as possible from participants while still reliably modeling shot activities (e.g., only one common goal is required for calculation of all LCTs – though if a participant provides resource availability constraints, those can be easily incorporated), and

2) Provide consistently actionable information regarding alternative scenarios to Z SoS participants in order to aid them in their own planning, execution, adapting, and interfacing with other entities.

Including LCTs can provide SoS participants with actionable information to help coordinate work through reasoning that is more functional in several ways than the ETE-only STN constructed by Schaffner (2017). First, it can provide at-a-glance information regarding slack in the experiment's schedule: when viewing a timeline, a participant can easily ascertain the window of time for each activity to begin, and the length of window for each activity's start time (Figure 1) relative to other windows can indicate that activity's proximity to the (anticipated) critical path. Second, it can provide an easy heads-up for those activities that will most impact the developing timeline of operations (and resultantly most impact the chances that the operational goal is achieved). If a participant perceives that the window for beginning an activity is uncomfortably small, they may adjust their behavior or the activity's scope, prepare in advance, garner additional resources, or even attempt to communicate/collaborate with others who may be impacted. Third, the bounded-window view compactly summarizes Simon's (1992) "alternative scenarios" (i.e., by showing a range of time over which each activity might happen, rather than a single prediction) – even showing the alternate ways an experiment is at risk of failing to achieve the operational goal. Through all of these means, this view increases understanding of the *behavioral* aspect of an experiment's schedule of activities for all participants (a la Rhodes & Ross 2010), and at once contributes to both the Upper Layer Description and Upper Layer Analysis of the SoS operations during a given day (a la Maier 2005).

One caveat in the present work's method of LCT calculation is that, since LCTs are derived from back-scheduling an operational goal through the minimum bounds of the network, the calculated LCTs are highly optimistic. If an activity is started at its latest cutoff time, the operational goal

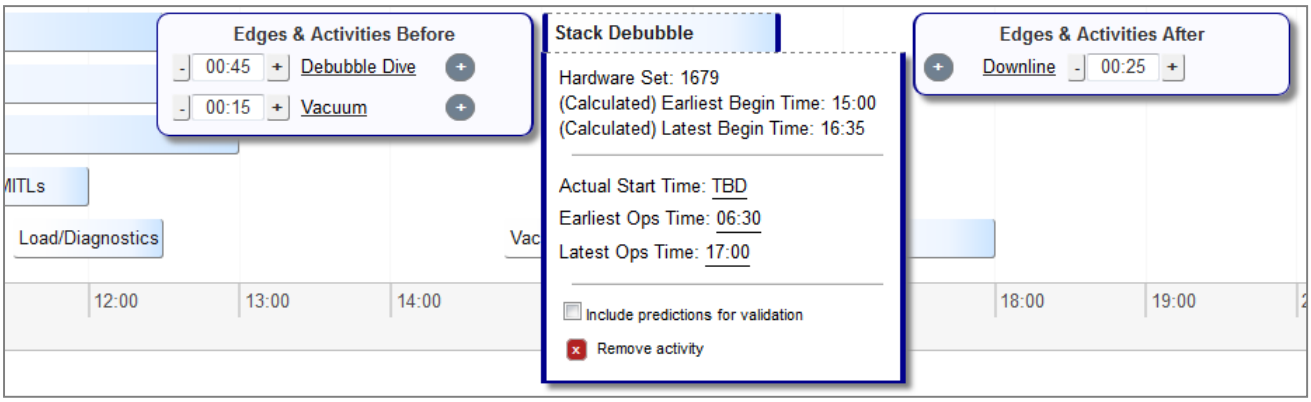


Figure 2. Meta-information available about an activity shown when a user selects that activity on the timeline. Edges and edge lengths are shown on each respective side of the info box, where they can be edited as needed (prompting rescheduling of the network). Calculated ETEs and LCTs as well as resource constraints (06:30 and 17:00) are shown in the info box.

might still be able to be achieved, but only if all following activities' edges hold to their respective lower bounds (each of which are individually optimistic). This optimism raises potential concerns for the perceptual aspect of complex systems interpretation (especially for unknown/unanticipated future participants in experiments, who may not realize just how optimistic the LCTs are). To help address this perceptual challenge, this high degree of built-in optimism to the LCTs should always accompany the LCTs through whatever means they are communicated, and the upper bounds of activities' windows should be clearly identified as challenging and undesirable locations to be occupying on the shot timeline – which is helped, for example, by including "cutoff" in the name.

Results

As discussed above in **Method**, the present work uses the minimum-bound STN constructed by Schaffner (2017) to provide a template for a Z experiment. All activities are then scheduled to begin at their earliest times (thereby providing ETEs) on experiment day. The Z Shot activity is then defined as the operational goal, using a latest time determined by managers and experimenters. This goal node-time is then back-scheduled through the network in order to provide the upper bounds on activities' execution. The design of the database-driven web application created for use in storing, viewing, and modifying the plans and schedules of Z experiments is now discussed.

Creation of the Software Application

The scheduling of activities' ETEs and LCTs has been incorporated into an ASP.NET interactive web application, deemed PSYCHE (Planning with SYstematic CHronological Estimates). The resulting information system allows Z participants to capture and view information about activities before shot day, aiding them in making more informed decisions about their own work vis-à-vis the potential timeline developments of the experiment. Allowing the automated scheduling information to be viewed during planning stages is consistent with Smith's (2003) observation that in many applications, "planning…and scheduling…are not cleanly separable" and helps address the need Smith raises for "the design of more tightly integrated planning and scheduling processes" (2003). The vis.js framework (visjs.org) was used on the front-end to enable smooth, intuitive interactivity with the timeline, so that clicking on any activity on the timeline will provide more information about that activity, including valid operational hours of the activity as well as edges and lengths to other activities on the timeline. An example of this information can be seen in Figure 2.

In addition to aiding planners before an experiment is executed, PSYCHE aids experiment execution on shot day by including back-end interfaces with embedded facility diagnostics to provide real-time updates to the windows of time for which activities are scheduled. PSYCHE includes a scheduler that runs once each minute, updating the earliest times of all scheduled activities' windows to provide accurate up-to-the-minute information to shot participants on shot day. As real-time information comes in regarding when activities actually began (or as minutes pass by and activities do not begin), dependent activities' execution windows can be updated (i.e., their earliest time estimates shift to be later in time). These real-time updates do not affect the Latest Cutoff Times, since LCTs only depend on the predefined operational goal and the minimum edges defined in the temporal network. These minute-by-minute updates to an activity's ETE means that a scheduled activity's window "closes" as the day progresses, which is the way many participants already think about execution opportunities; therefore, the software's "closing window" depiction is in line with Markus et al.'s (2002) recommendation to match user intuition when possible.

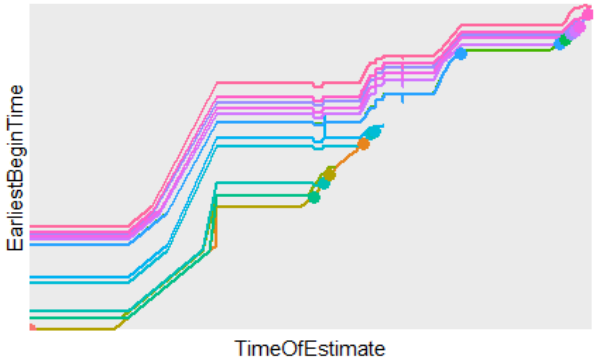


Figure 3. Traces of all activities' ETEs throughout a day; each line represents the trace of one activity's ETEs over the day (dot represents activity start). ETEs only increase as time increases.

Verification: Measuring Accuracy

One of the primary ways that PSYCHE's performance is being measured is by recording activities' ETEs minute-by-minute and comparing those estimates against activities' actual begin times. As discussed in **Method**, an ETE for an activity should always remain valid as an estimate of the *earliest* start time; if the estimate is adjusted, it should only be adjusted to be later in time (i.e., the left-hand side of an activity's execution window on the timeline should only move to the right), so that the original estimate remains valid as an earliest time (following Markus et al's (2002) recommendation to "improve offline behavior"). If any of the ETEs provided over a day for a given activity ends up being later than the actual start time of that activity, it is an inaccurate ETE. In addition, if any of an activity's ETEs estimate a time that is earlier than a previously provided ETE for that activity, then the previously provided ETE(s) should be considered inaccurate. By recording all of the minute-by-minute ETEs of an activity up to the actual begin time of that activity, it becomes possible to measure both of these conditions in order to check that the adjustments of estimates are behaving as desired (i.e., are only adjusted to estimate *later* start times as the experiment unfolds).

Motivated by the above observations, the calculated earliest times for a day's activities are recorded on a minute-by-minute basis throughout the day (since PSYCHE's real-time scheduler runs each minute, potentially updating the earliest times for each activity that day). The resulting records clearly show the behavior of estimates of all activities involved in a given experiment; an instance of the recorded minute-by-minute estimates for all activities in a shot can be seen in Figure 3. Each estimate can be seen to increase as a function of the time of the estimates, which is the expected and desired "accurate" behavior. The working definition for "accuracy" is defined for the present work as "the proportion of minute-based estimates of earliest begin time that are earlier than an activity's actual begin time."

In order to avoid surprise and encourage the check-in behavior discussed above in **Method**, it is necessary to maximize accuracy of the minimum bounds; however, it should be pointed out that one of the rather severe risks of achieving 100% accuracy by this definition is that *the minimum bounds between activities might be too small*. This would cause the LCTs of activities to be more optimistic than is appropriate (due to back-scheduling's use of minimum bounds). To address this risk, it may be desirable for some very small portion of estimates *to be inaccurate* – meaning an occasional activity's actual begin time is earlier than its minimum time estimate. Formalization of these latter concepts is ongoing, with the definition of "precision" of estimates being an area identified as a needed step in future work. In the meantime, the working definition of accuracy presently outlined serves as a practical measure of performance in serving the purpose of coordination.

Validation: Z SoS Participant Feedback

If Z participants are ostensibly those being served by the approach outlined in the present work, it stands to reason that they should be consulted on the perceived value of the work. Initial feedback along these lines has been obtained in several ways, albeit all anecdotal. First, casual conversations with installers and technicians have been conducted, inquiring as to which information would they rather be given: a "likely time" or even quantified probabilistic estimate, or a window of time during which an activity may occur. The results of these conversations fairly consistently reflect a desire for the "window" option.

Second, in response to direct questioning by a Z participant of "When will Activity X happen?", the response has been given in terms of a window of time (sometimes showing the PSYCHE timeline) and asked if that was satisfying. The answer was usually "Yes", though on occasion the reply was "Sure, but what time do you *think* it will happen?" (This latter response is not unexpected, since it reflects established cultural norms around gauging various individual perceptions in forming one's own opinion of likelihood, which is one of the asystematic behaviors that the present work is attempting to address.)

Finally, the timeline has been consulted and shown to decision makers on particularly complex shot days in order to communicate the slim margins of time associated with the operational goal that day. The decision makers were more informed by the visual timeline than they otherwise would have been, and were able to take action accordingly. Through continuing these types of interactions, it is hoped that feedback will continue to affirm the usefulness of the provided windows of time.

It is anticipated that as Z participants and decision makers interact with the visualization of PSYCHE, their perceptions and valuations of hypothetical and actual outcomes will change. This expectation is derived from previous applications of established psychology research to interactive visualizations, such as (Ricci et al. 2014). As discussed in that work, stakeholder's mental models of complex systems differ from the constructed models of systems used in engineering design. Interacting with visualizations of the constructed models (more specifically, being able to see the estimated results of different design choices), allows adjustment of both a stakeholder's mental model of a system and the constructed model of that system, leading to "better" decisions (defined by Ricci et al. as "trusted, truthful" decisions). It is further hypothesized in the present work that as Z participants observe and respond to the ETEs and LCTs provided to them (i.e., interact with the constructed model of an experiment), they will grow to trust the model and allow it to update their mental model as appropriate, leading to more consistent (and more confident) distributed functional reasoning

Further Work

With an initial prototype of PSYCHE complete, work can now transition to several fronts. First, the embedded sensors in the machine often provide false, conflicting, or irrelevant information. Real-time filtering and state estimation is needed to ensure that PSYCHE's estimates reflect the actual states of the machine. In tandem with the effort of creating such a filter/estimator, one of the next major improvements could be automated planning on subsets of activities to handle unscheduled events (e.g., rework) when the machine states indicate so.

Second, the activities chosen for the initial version of PSYCHE were chosen based on machine states which are already automatically diagnosed by embedded sensors. This set of states does not equal the set with which all Z participants are concerned, however. A more complete (and hopefully more broadly useful) state model is under construction, along with analysis of how embedded diagnostics could reliably indicate those states.

Finally, the methods of communication of the information captured and calculated by PSYCHE is an essential area ripe with opportunity. Live schedule updates could be communicated in multiple ways to various parties on the machine, increasing dissemination of progress and risks throughout a shot's execution. It is envisioned that this increased level of communication will further strengthen the interfaces between participants and implicitly encourage further cooperation.

Adding Probabilities and Entropy for Estimates

With a subset of the current states, and as more states are added, some of the temporal relationships between states will be able to be described reliably with probabilistic information. Because the current application does not incorporate any such information, it may need to be expanded in some way in order to provide as much useful information as possible when planning a Z experiment and increasing the chances of success in following a plan (even in the face of other activities/uncertainties that cannot be so characterized). One example of a potential improvement in this respect would be to implement a Probabilistic Simple Temporal Network with Uncertainty (PSTNU), which marries STNUs with probabilistic information so that a planner may incorporate as much information as possible to minimize risk (Santana et al. 2016). It is possible that a modified form of the PSTNU would allow more informed planning of Z experiments by experimenters interested in minimizing specific risks.

Importantly, however, this probabilistic information would only be intended to help planners before shot day: all of the challenges identified in **Method** remain for broad communication of such probabilistic information to the participant community, and it is not currently perceived that this added capability would contribute toward the present work's SoS-level goals of encouraging cooperation and leveraging interfaces for real-time execution.

An intriguing middle ground for broad communication of pseudo-probabilistic information might be an ordinal ranking of entropy for any ETE provided, so that if an activity were deemed fairly well known/described (e.g., automated fluid drains), successive activities' ETEs could be deemed "high quality", since less entropy is introduced into the temporal network from less-EKP-like activities. This measurement of "quality of estimate" could be useful to some participants, and those who are not helped by it could still safely ignore it, using only the ETE provided.

Adding System States

Another area of ongoing work is the expansion of the set of activities included in the planning and scheduling of Z shots. The currently included set comprises activities which are associated with already-existing embedded sensors. While the sensors are useful for the proof of concept of the present work, it is hypothesized that more useful states can be derived from stakeholder analysis and state machine studies, which are presently underway. The state machine(s) constructed will not only be able to inform the filtering of sensor input for more reliable real-time updates, but should also prove to be useful in any efforts toward automated planning on a subset of machine states.

Rolling Out Live Status Indicators

Once the initial version of PSYCHE has been operational for some time providing accurate estimates of the earliest times that activities can begin, those estimates (along with the latest times) can begin to be automatically communicated to Z participants. The aforementioned stakeholder analysis will likely aid in determining which types of participants need what information regarding schedule updates, as well as how often those updates are needed and the most effective methods for communication. Prior to the completed stakeholder analysis, various communication methods are already being considered, ranging from electronic kiosks with PSYCHE's view of the higher-level schedule for a shot, to LED matrix signs placed throughout workspaces, to automated email and PA announcements, among others. Since communication itself is one of the primary components of "the interfaces" between participants, the choices of what information to communicate, to whom to communicate that information, and the frequency and method of communication, have potentially drastic effects on the eventual success or failure of the present goals of leveraging interfaces and encouraging cooperation.

Conclusion

This work continues previous efforts to improve the interfaces of the Z System-of-Systems through distributed planning and automated scheduling of activities. Goals were defined for higher-level planning and scheduling activities to "leverage interfaces" and "encourage cooperation" by 1) requiring minimal information from each participant regarding their own planned activities, and 2) aiding in functional reasoning around the execution of activities for a given experiment. The present extension of the originally proposed STN is an STNU-like construct relating each activity with others, scheduling activities' Earliest Time Estimates (ETEs) and then using a single operational goal to provide the Latest Cutoff Times (LCTs) for all activities. The combination of ETE and LCT ultimately provide an execution window for each activity, which was incorporated into an interactive software tool, PSYCHE, intended to support both planning and execution. On-going areas of work were then discussed, including validation of estimates of earliest times, elicitation of participant feedback, and expanding the set of activities, sensors, and communications with which PSYCHE interfaces.

Acknowledgements

The author is grateful to Sandia National Laboratories for funding this research. The author is additionally grateful to Ryan Kamm for feedback and to Trent Yocom and Rafael Aragon for their patience and their sharing of knowledge.

References

Dechter, R., Meiri, I. and Pearl, J., 1991. Temporal constraint networks. Artificial intelligence, 49(1-3), pp.61-95.

Kahn, Arthur B., 1962. Topological sorting of large networks, Communications of the ACM, 5 (11): 558–562.

Maier, M.W., 1996, July. Architecting principles for systems-of-systems. *In INCOSE International Symposium* (Vol. 6, No. 1, pp. 565-573).

Maier, M.W., 2005, October. Research challenges for systems-of-systems. *In 2005 IEEE International Conference on Systems, Man and Cybernetics* (Vol. 4, pp. 3149-3154). IEEE.

Markus, M. L., Majchrzak, A., and Gasser, L., 2002. A Design Theory for Systems that Support Emergent Knowledge Processes, MIS Quarterly (26:3), September, 2002, pp. 179-212.

Rhodes, D.H. and Ross, A.M., 2010, April. Five aspects of engineering complex systems emerging constructs and methods. *In Systems Conference*, 2010 4th Annual IEEE (pp. 190-195). IEEE.

Ricci, N., Schaffner, M.A., Ross, A.M., Rhodes, D.H. and Fitzgerald, M.E., 2014. Exploring stakeholder value models via interactive visualization. Procedia Computer Science, 28, pp.294-303.

Santana, P., Vaquero, T., Toledo, C., Wang, A., Fang, C. and Williams, B., 2016. PARIS: a Polynomial-Time, Risk-Sensitive Scheduling Algorithm for Probabilistic Simple Temporal Networks with Uncertainty. *In Proc. of ICAPS* (Vol. 16, pp. 267-275).

Schaffner, M.A., 2017. A Simple Temporal Network for Coordination of Emergent Knowledge Processes in a Collaborative System-of-Systems. *In Proc. 16th Int. Conf. on Accelerator and Large Experimental Control Systems (ICALEPCS'17)*, Barcelona, Spain, Oct. 2017 (paper THMPL01, pp. 1252-1256).

Simon, H.A., 1996. The sciences of the artificial. Cambridge, Mass.: MIT press.

Smith, Stephen F., 2003. Multidisciplinary Scheduling: Theory and Applications. *In 1st International Conference, MISTA '03*, Nottingham, UK (Vol. 13, No. 15, pp. 3-17).

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

Area Coverage Planning with 3-axis Steerable, 2D Framing Sensors

Elly Shao and **Amos Byon** and **Chris Davies** and **Evan Davis** and
Russell Knight and **Garett Lewellen** and **Michael Trowbridge** and **Steve Chien**
Jet Propulsion Laboratory, California Institute of Technology
{elly.j.shao, amos.j.byon, christopher.j.davies, evan.w.davis,
russell.l.knight, michael.a.trowbridge, steve.a.chien}@jpl.nasa.gov

Abstract

Existing algorithms for Agile Earth Observing Satellites((Lemaître et al. 2002)) were largely created for 1D line sensors that acquire images in linear swaths. However, imaging satellites increasingly use 2D framing sensors (cameras) that capture discrete rectangular images. We describe tiling step-stare approaches that are more suited to rectangular image footprints than are 1D swath-based algorithms. Optimal area planning for these 2D framing instruments is an NP-complete problem and intractable for large areas, so we present four approximation algorithms. Strategies are compared against a prior 2D framing instrument algorithm (Knight 2014) in three computational experiments. The impact of observer agility on schedule makespan is examined. Makespans vary more as observer agility decreases toward a critical point, then vary less after the critical point, suggesting a possible problem phase transition.

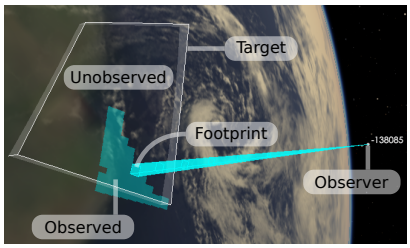


Figure 1: Framing sensor observations (teal) of a target area (white).

Introduction

Most existing area coverage algorithms for agile Earth-observing satellites are intended for 1D sensors and adapt poorly to 2D framing instruments with rectangular fields of view. These pushbroom algorithms continuously sweep the sensor across the target, which would smear the image captured by a framing sensor. An obvious alternative is to track a target point, which informs a step-stare strategy where the target is decomposed into a rectangular grid, and the sensor tracks each grid point for the duration of an image capture before moving on to the next(figure 2). The challenge

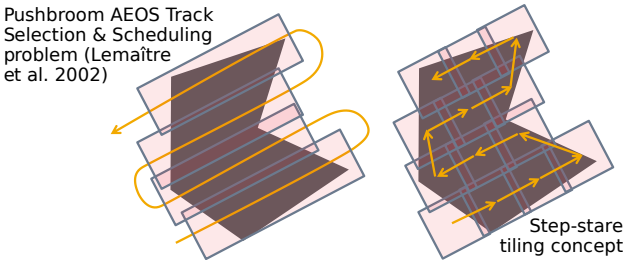


Figure 2: Comparison of pushbroom and step-stare.

in determining grid layout is that the the imager footprint (projection of instrument field of view onto body surface) change as the observer flies past (figure 3).

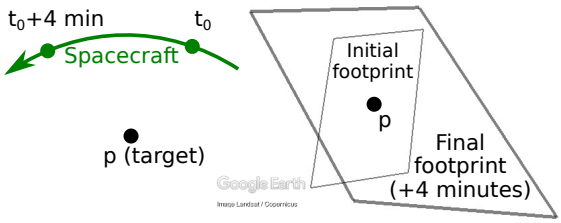


Figure 3: Imager footprint changes size and shape

We wish to image the entire target area while minimizing makespan (schedule duration). The area visible to the sensor is time dependent, and the cost to slew between target points is time varying and asymmetric (Lewellen et al. 2017a). This paper discusses the difficulty of this optimization problem, presents optimal solution approximation algorithms, then evaluates them in three computational experiments.

Contact author: Michael Trowbridge

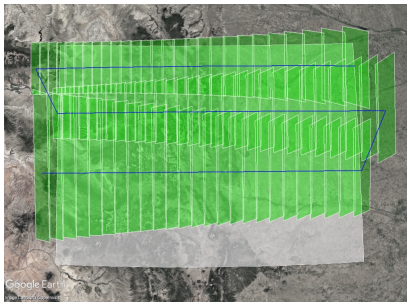


Figure 4: 1D (Lemaître et al. 2002) algorithm adapted to use a 2D framing sensor, with sweep lines (blue line) and image footprints (green). Only 81% of the target (white) was satisfied

Related Work

Lemaître et al. present a swath-based Boustrophedon (lawnmower) decomposition algorithm for area coverage with agile satellites (Lemaître et al. 2002). They argue that the track selection problem is NP-hard, so approximation algorithms are acceptable. The paper assumes 1D sensors and does not discuss 2D sensors.

Our adaptation of their algorithm for 2D framing instruments performed poorly when slew rate was constrained to the slower readout times of 2D framing instruments. On our easiest test case[1], the swath-based algorithm did not cover the entire target (figure 4).

Knight identified area coverage planning for 2D framing instruments as NP-hard by analogy to finding a Hamiltonian path through a grid-graph, then presented a concentric ring (Milling) tiling algorithm (Knight 2014). While provably optimal for even grid-graphs, a target polygon's grid decomposition could be odd, producing sub-optimal ring stitch points. The algorithm subdivides grid tiles into either two or four sub-tiles, which makes the graph even, but reduces area satisfied per image capture and increases makespan due to overlapping imager footprints. This paper presents algorithms that do not subdivide.

Formulation

Problem Statement

Choose observations (real-valued target points, rotations, observation times) s.t. the union of all images captured by those observations when projected onto the target body (modeled as a triaxial ellipsoid) cover the target great-circles polygon within a bounded temporal interval (single overflight), subject to constraints, with sufficient slew time between observations.

[1]GOLIAT/CICLOP CubeSat, using ON Semiconductor AR0331 subwindowed to 16:3 aspect ratio (Semiconductor Components Industries, LLC 2017)

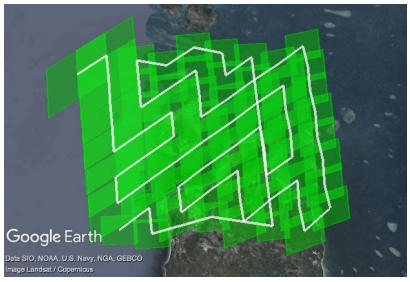


Figure 5: Milling (Knight 2014) algorithm grid points (connected by white line) and footprints (green). Subdivision causes excessive overlap and some skipped tiles that cover no target area (see edge crossing).

Nomenclature

p	A point on the target body
$(p_1, \ldots, p_n)$	A great circles polygon on the target body with n vertices
P	A set of target great circles polygons
$\mathbf{r}_{\text{obs}}$	Position vector of the observer at a given time t, determined by spacecraft orbit
$\mathbf{r}_{\text{tgt}}$	Position vector of the observer's imaging target (center of camera field of view)
θ	A rotation of the observer about its look vector $\mathbf{r}_{\text{look}}(\mathbf{r}_{\text{tgt}} - \mathbf{r}_{\text{obs}})$
t	Time. t_0 is the start time of the planning horizon, t_f is final time.
b	A single observation that captures an image: $(\mathbf{r}_{\text{obs}}, \mathbf{r}_{\text{tgt}}, \theta, t)$
B	The set of all possible valid observations b
A	The observation tour (schedule), an ordered list of scheduled valid observations a
m	Makespan of A, $m = [\min t \in A, \max t \in A]$
$Tile$	A image footprint polygon: intersection of the body surface and the observer's field of view
g	A *tile* corresponding to a scheduled observation
G	The set of scheduled image footprint polygons.

Formal Problem Statement

Given a set P of polygons on the target body,

$$P = \{(p_1, p_2, p_3)_i\} \tag{1}$$

the set B of all possible valid observations b that fall within the planning horizon $[t_0, t_f]$

$$\forall\, (b = \{\mathbf{r}_{\text{tgt}}, \theta, t\}) \in B, t_0 < t < t_f \tag{2}$$

a function to create a polygon g from an observation, representing the image footprint[2]

$$g \leftarrow \text{footprint}(\mathbf{r}_{\text{tgt}}, \theta, t) \tag{3}$$

[2]The image's angular field of view depends on instrument design

a Boolean valued function to check if a slew between observations b_i and b_j is valid

$$Boolean \leftarrow \text{slewOk}(b_i, b_j) \tag{4}$$

Some tour $A \subseteq B$ is valid iff

$$P \subseteq \text{union}\left(\{\text{footprint}\,(a_i)\,|i \in 1, 2, \ldots, |A|\}\right) \tag{5}$$

and

$$\bigwedge_{i=1}^{|A|-1} \text{slewOk}\,(a_i, a_{i+1}) \tag{6}$$

Constraints

The observation schedule A must have a makespan m within the visibility window $[t_{v0}, t_{vf}]$ where there is a line of sight from the observer to the target:

$$m \subseteq [t_{v0}, t_{vf}] \tag{7}$$

We enforce this by scoping B to the planning horizon ($[t_0, t_f] \subseteq [t_{v0}, t_{vf}]$). The geometric visibility window $[t_{v0}, t_{vf}]$ is a finite planning horizon.

Observations $b \in B$ have minimum duration

$$\Delta t_{\text{obs}} > 0 \tag{8}$$

Observations cannot be concurrents. Observation transition time is strictly positive and depends on spacecraft agility.

Geometric constraints are satisfied by restricting our search to the visibility interval $[t_{v0}, t_{vf}]$ determined with existing software[3].

Tractability of the Optimization Problem

OptFramePlan is the optimization formulation of the framing instrument area coverage scheduling problem. The goal is to find the shortest makespan schedule that satisfies conditions 5 and 6, subject to the previously listed constraints.

Theorem 1. *Finding the makespan-optimal area coverage plan for a space-based 2D framing instrument is NP-complete.*

Lemma 1.2 shows that the problem belongs to NP because an arbitrary solution is polynomial-time verifiable. Lemma 1.3 shows that the problem is at least as hard as finding a Hamiltonian path in a grid graph, which is NP-complete (Itai, Papadimitriou, and Szwarcfiter 1982).

Lemma 1.1. *The size $|A|$ of a solution schedule $A \subseteq B$ is bounded from above by $\frac{t_f - t_0}{\Delta t_{obs}}$.*

Proof. By contradiction/pigeon-hole principle. Assume A contains $\frac{t_f - t_0}{\Delta t_{\text{obs}}} + 1$ observations. Neglecting transition costs between observations, the horizon $[t_0, t_f]$ can accommodate $\frac{t_f - t_0}{\Delta t_{\text{obs}}}$ non-overlapping observations. If $|A| = \frac{t_f - t_0}{\Delta t_{\text{obs}}} + 1$, then at least two observations overlap, contradicting the problem constraints. □

[3]Systems Toolkit (STK) (Analytical Graphics Inc. 2017), Satellite Orbit Analysis Platform (SOAP) (Stodden and Galasso 1995), SPICE (Acton et al. 2016)

Lemma 1.2 (OptFramePlan $\in NP$)**.** *An arbitrary area coverage plan for a space-based 2D framing instrument is verifiable in polynomial time.*

Proof. Lemma 1.1 proves that $|A| = |G|$ is bounded linearly by the planning horizon and minimum observation duration. A plan that contains $|A|$ observations has at most $|A| - 1$ slews to validate using the constant-time slewOk function. Validating constraint compliance of each observation is also linear in $|A|$ observations. Unioning G (a set of sets) has no worse than $O(|G|^2)$ time complexity (Cormen et al. 2009). □

Lemma 1.3 (HamiltonianPath $\leq_P$ OptFramePlan)**.** *Finding the makespan-optimal area coverage plan for a space-based 2D framing instrument is polynomial-time mappable onto an instance of finding a Hamiltonian path in a grid graph.*

Proof. Relax the makespan-optimal area coverage planning problem by discretizing the target polygons into a uniform, target-fixed grid of points V. Omit rotation about the look vector θ and set the planning horizon short enough that skew may be neglected. Assume that the observer can slew equally well in any direction, making slew distance metric within the grid graph.

Define general grid graph $\Gamma = \{V, E\}$, where time cost $c_{ij}(t)$ between $v_i \in V$ to $v_j \in V$ via edge $e_{ij} \in E$ is a function of arrival time t, the sum of all previous costs added to the tour start time. An edge e_{ij} exists only if arrival time $t \leq t_{end} \leq t_f$, where t_{end} is the end time of the shortest known schedule. The makespan-optimal area coverage plan for this target discretization will visit each point in V once (i.e. be a Hamiltonian path). □

OptFramePlan is NP-complete, so makespan-optimal framing instrument scheduling of large areas is expected to be intractable. Thus, we limit our discussion to approximation algorithms.

Approximation Planning Algorithms

This section presents four deterministic step-stare tiling algorithms that approximate a makespan-optimal solution to OptFramePlan. The algorithms differ in when they commit the plan to target points $\mathbf{r}_{\text{tgt}}$, how far ahead they plan and how they maintain the plan.

Sidewinder: Target-fixed Boustrophedon

Sidewinder is an adaptation of the Boustrophedon (lawnmower) algorithm (Choset and Pignon 1998). Construct a grid of ground points R with a 2D flood-fill algorithm (Lee, Pan, and Chu 1987) and walk the grid in alternating rows (algorithm 1). Each $\mathbf{r}_{\text{tgt}} \in R$ is fixed at plan start time t_0 - but the other time-varying elements of the observation tuple a are fixed at schedule time t (algorithm 2).

Algorithm 1 Sidewinder planTour

```
function planSidewinderTour(P, t)
    Tour ← ∅                                   ▷ Planned tour
    bBox ← computeBoundingBox(P)
    closestSide ← findClosestSide(bBox, t)
    Tiles ← discretize(bBox)
    bearing ← true                  ▷ Alternates row direction
    (x_min, x_max, y_min, y_max) ← gridExtrema(Tiles)
    if closestSide ∈ {NORTH, SOUTH} then
        for i ← y_min to y_max do
            y ← y_max − i + y_min
            if closestSide = NORTH then
                y ← i
            end if
            for j ← x_min to x_max do
                x ← x_max − j + x_min
                if bearing then
                    x ← j
                end if
                Tour.add(x, y)
            end for
            bearing ← ¬bearing
        end for
    else                          ▷ East or West side is closest
        for i ← x_min to x_max do
            x ← x_max − i + x_min
            if closestSide = EAST then
                x ← i
            end if
            for j ← y_min to y_max do
                y ← y_max − j + y_min
                if bearing then
                    y = j
                end if
                Tour.add(x, y)
            end for
            bearing ← ¬bearing
        end for
    end if
    return Tour
end function
```

Algorithm 2 Sidewinder

```
while P ≠ ∅ do
    Tour ← planSidewinderTour(P, γ, t)
    while Tour ≠ ∅ do
        a_i ← pop(Tour, t)
        append a_i to A
        g ← footprint(a_i)
        P ← P − g
        t ← t + Δt_obs + slewDur(t, a_{i−1}, a_i)
    end while
end while
```

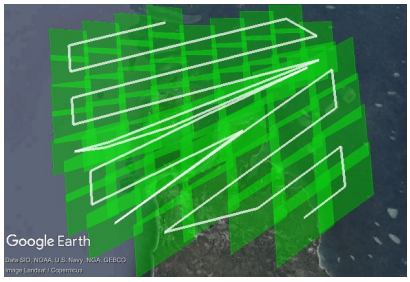


Figure 6: Sidewinder: rotation and skew invalidate the initial and second plans, prompting restarts by the outer loop (while $P \neq \emptyset$).

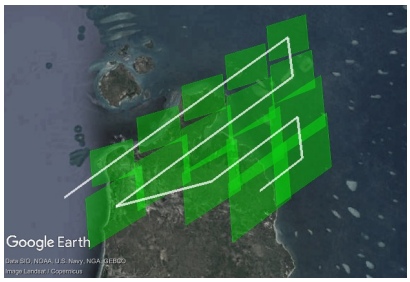


Figure 7: Fixing the grid to the target at t_0 causes gaps.

Replanning Sidewinder

Sidewinder commits grid points to target-fixed points too early. Image footprints change, so the plan develops gaps over time (figure 7). Replanning Sidewinder replans the tour after each move (algorithm 3) and only commits a planning grid point to a target point $\mathbf{r}_{\text{tgt}}$ at schedule time t.

Algorithm 3 Replanning Sidewinder

```
while P ≠ ∅ do
    γ_{i−1} ← pop(Tour) or center(P) if Tour = ∅
    γ ← optimizeGridOrigin(γ_{(i−1)})
    Tour ← planSidewinderTour(P, γ, t)
    a_i ← pop(T, t)
    append a_i to A
    g ← footprint(a_i)
    P ← P − g
    t ← t + Δt_obs + slewDur(t, a_{i−1}, a_i)
end while
```

Five pieces of plan state persist between replans: the most recently scheduled point $\mathbf{r}_{\text{tgt},i}$, next target point $\mathbf{r}_{\text{tgt},i+1}$, $\tau_{\text{row}} \in \{+,-\}$, $\tau_{\text{col}} \in \{+,-\}$ and row alignment axis $\alpha \in \{w,h\}$. The next grid origin point γ_{i+1} is $\mathbf{r}_{\text{tgt},i+1}$ and the algorithm usually[4] scans row 0. When $\mathbf{r}_{\text{tgt},i}$ discretizes to a row that is in the τ_{row} direction from $\mathbf{r}_{\text{tgt},i+1}$, τ_{col} is negated.

[4] If the next tour point is the final point in a row, and is in the tour at γ_{i-1} but out of the tour at γ_i, then the next

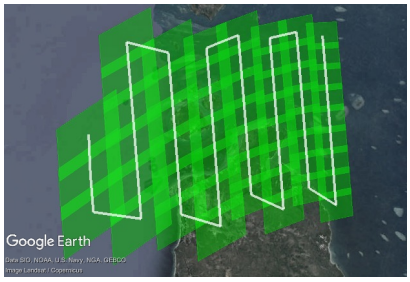


Figure 8: Replanning Sidewinder: adaptive row width

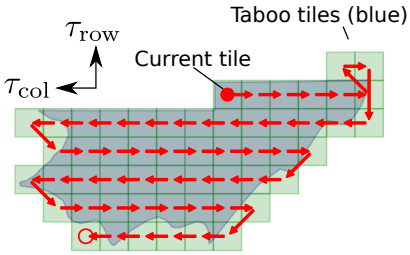


Figure 9: Two taboo tiles (circled in blue).

Changing geometry can move target area into a previously visited grid cell (the blue circle of figure 9), requiring a revisit of either a prior row (in the τ_{row} direction), or a different column in the current row (in the τ_{col} direction). Tiles requiring backwards moves are *taboo*. The tour can cycle between two taboo tiles until the opportunity interval is exhausted, so we remove taboo tiles by shifting the row with 1d constrained local optimization (algorithm 4). Perturb the origin by δ until the grid with origin $\gamma_{i+1} + \delta$ has no taboo tiles. Minimize δ in the τ_{row} direction (shifting rows backward).

Online Frontier Repair

This strategy updates a global plan after each action. The target is discretized into a grid using an 8-neighbor flood-fill based on the $O(n \log n)$ stack flood-fill algorithm (Lee, Pan, and Chu 1987). The initial plan (figure 10) is a rectangular Boustrophedon decomposition (Choset and Pignon 1998). Tiles are converted from observer planning space to ground coordinates at schedule time t (algorithm 5).

point in the tour will belong to row 1, not 0.

Algorithm 4 Optimize Grid Origin $\gamma_{(0,0)}$

T ▷ set of taboo tiles in the grid
w_{tile} ▷ width of a tile
h_{tile} ▷ height of a tile
α ▷ grid alignment direction (w or h)
function OPTIMIZEGRIDORIGIN($\gamma, \tau_{\text{row}}, \tau_{\text{col}}$)
  $D \leftarrow \begin{cases} \left[0, \frac{w_{\text{tile}}}{2}\right), & \text{if } \alpha = w \\ \left[0, \frac{h_{\text{tile}}}{2}\right), & \text{otherwise} \end{cases}$ ▷ search domain
  $\delta_\alpha \leftarrow \text{MINIMIZE}(|\delta|, \delta \in D \text{ s.t. } T|_{\gamma+\delta} = \emptyset)$
  return $\gamma + \delta_\alpha$
end function

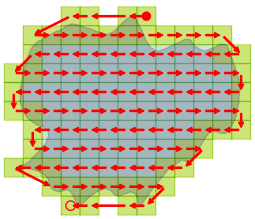


Figure 10: Initial plan (red arrows) from Online Frontier Repair algorithm. Frontier tiles are yellowed.

Algorithm 5 Online Frontier Repair

Plan $Tour$
while $P \neq \emptyset$ **do**
  UPDATEGRID($Tour, F, N, X$)
  REMOVE($Tour, x \in X$) ▷ tiles we no longer need
  INSERTCHEAPEST($Tour, n \in N$) ▷ New tiles
  $a_i \leftarrow \text{POP}(Tour, t)$
  append a_i to A
  $g \leftarrow \text{FOOTPRINT}(a_i)$
  $P \leftarrow P - g$
  $t \leftarrow t + \Delta t_{\text{obs}} + \text{SLEWDUR}(t, a_{i-1}, a_i)$
end while

Update the grid by seeding the flood-fill algorithm with the prior iteration's flood-fill result (algorithm 6). In the best case, only the outer edge changes. In the worst case, every tile is on the frontier, so updating is as slow as creating a new discretization – $O(|T| \log |T|)$ per update, where T is the set of grid points in the tour. In practice, interior points are infrequently re-evaluated.

We use the cheapest insertion heuristic (Rosenkrantz, Stearns, and Lewis 1977) with a Manhattan distance cost function when adding new tiles. Future work should examine more sophisticated heuristics.

Figure 11: Online Frontier Repair. Note suboptimal repairs on the right side (final leg).

Algorithm 6 Frontier Flood-fill Update

function UPDATEGRID($Tour, F, N, X$)
  $C_{in} \leftarrow \emptyset, C_{out} \leftarrow \emptyset$ ▷ closed lists
  $O \leftarrow F$ ▷ Open list O starts as frontier set F
  if $O = \emptyset$ **then**
    $O \leftarrow$ DISCRETIZE(unsatisfied target vertices)
  end if
  $S \leftarrow O$ ▷ Seeds: initial open list
  while $O \neq \emptyset$ **do**
    $o \leftarrow O$.pop()
    if COVERS(o, targets) **then**
      $C_{in} \leftarrow o$
      $membershipChanged \leftarrow (o \ni Tour)$
    else
      $C_{out} \leftarrow o$
      $membershipChanged \leftarrow (o \in Tour)$
    end if
    if $membershipChanged \vee (o \in S)$ **then**
      for all $n \in$ neighbors (o) **do**
        if $(n \ni C_{in}) \wedge (n \ni C_{out})$ **then**
          O.push (n)
        end if
      end for
    end if
  end while
  $N \leftarrow C_{in} - Tour$ ▷ identify new tiles
  $X \leftarrow Tour \cap C_{out}$ ▷ tiles to remove from tour
  $F \leftarrow \emptyset$ ▷ rebuild frontier list
  for all $tile \in Tour$ **do**
    if $|$NEIGHBORS$(tile) \cap Tour| < 8$ **then**
      F.push$(tile)$
    end if
  end for
end function

Algorithm 7 Grid Nibbler checkNeighbors

function CHECKNEIGHBORS($\mathbf{r}, t$)
  $C \leftarrow$ GETCARDINALNEIGHBORS$(\mathbf{r}, t)$ ▷ ↑↓←→
  $D \leftarrow$ GETDIAGONALNEIGHBORS$(\mathbf{r}, t)$ ▷ ↙↖↗↘
  $best \leftarrow$ ARGMAX(score(c, t),$c \in C$)
  if $best$ did not finish a polygon **then**
    x ▷ Bias against diagonal neighbors
    $bestScore \leftarrow$ SCORE$(best, t)$ $\times x$
    for d in D **do**
      if d finishes a polygon **then**
        $best \leftarrow d$
      else if SCORE$(d, t) > bestScore$ **then**
        $best \leftarrow d$
        $bestScore \leftarrow$ SCORE(d, t)
      end if
    end for
  end if
  return $best$
end function

Local Grid Planning

This approach uses AI-inspired local planning with globally-informed heuristics such as radial distance of

Figure 12: Grid Nibbler: Radial distance heuristic.

Algorithm 8 Nibbler

while $P \neq \emptyset$ **do**
  $best \leftarrow$ CHECKNEIGHBORS$(prev)$
  if AREA$(best, t) < \epsilon$ **then**
    $newStart \leftarrow closesttargetcorner$
    $best \leftarrow$ CHECKNEIGHBORS$(newStart, t)$
    if SCORE$(newStart, t) >$ SCORE$(best, t)$ **then**
      $best \leftarrow newStart$
    end if
  end if
  $a \leftarrow$ MAKEOBSERVATION$(best, t)$
  append a_i to A
  $g \leftarrow$ FOOTPRINT(a_i)
  $P \leftarrow P - g$
  $t \leftarrow t + \Delta t_{\text{obs}} +$ SLEWDUR(t, a_{i-1}, a_i)
end while

$\mathbf{r}_{tgt}$ from center of target polygons P, area of P that footprint g satisfies and number of polygons g eliminated from P.

Consider a 3×3 grid of tiles centered on the target corner closest to the previous pointing. Score the eight neighbors with some global heuristic, and add the highest-scoring neighbor to the tour (algorithm 7). Nibble (subtract) the imager footprint from the target. Repeat, centering the grid on the previous tour point, until no target remains (algorithm 8). To prevent gridlock, the previous direction is taboo.

Experiment Methodology

Each algorithm is used to schedule a target polygon in five experiments, with these metrics:

- Completeness: fraction of target satisfied
- Schedule efficiency: shortest makespan (duration)
- Computational efficiency: lowest CPU runtime
- Memory consumption (minus fixed overhead)

In one experiment, we fix the orbit, imaging payload and target, then vary observer agility to characterize the planning problem. The other experiment tests algorithm performance over both target difficulty and observer capability. Target difficulty is based on size (226381 km^2 vs 8181 km^2, respectively). Both the hard

and easy targets have almost direct overflights during the planning horizon.

Table 1: Hard and easy observer configurations

	Easy	**Hard**
Agility	GOLIAT	Commercial
Imager	CICLOP	THEIA
Orbit Altitude (km)	309 × 1441 [5]	615 [6]

Two key traits influence observer suitability for minimum-makespan scheduling: agility and field of view (FOV). The actual GOLIAT/CICLOP CubeSat (Balan and Piso 2008; Dumitru 2006) is our capable (easy) observer with a wide FOV and high agility (180° slew in 30 seconds). A hypothetical[7] observer with typical commercial imagery satellite orbit and agility (180° slew in 120 seconds), but a smaller 1° FOV THEIA framing imagery payload (Ellison et al. 2013) is our less capable (hard) observer. Table 2 shows the instrument models used in this experiment. Agility is modeled as two-point linear interpolation of eigenaxis slew angle[8] between targets, with fixed settle time.

Table 2: Imaging Instruments

	CICLOP [9]	**THEIA** [10]
Shape	Rectangular	Rectangular
Horizontal FOV	5.73°	1°
Vertical FOV	4.26°	1°
Image duration	0.17s	1.0s

The experiments run on a 2015 Macbook Pro (2.6 GHz Intel Core i7, 16 GB RAM).

Results

Agility and Problem Difficulty

Figure 13 demonstrates agility ranging from that of GOLIAT (upper limit) to ALL-STAR (lower limit), with a band covering some typical commercial imagery satellite agilities (Hutin 2009; Satellite Imaging Corporation 2017; MDA DigitalGlobe 2017). A commercial imagery satellite bus should be capable of satisfying the target area in a single overflight.

The problem is easier (more constrained) for less agile spacecraft because they slew too slowly to cover the entire target. For a more agile spacecraft, it is easy to find a valid solution, but harder to find the optimal solution. Future work should examine the critical point at the slow end of the commercial agility band (figure 13) as a possible problem phase transition.

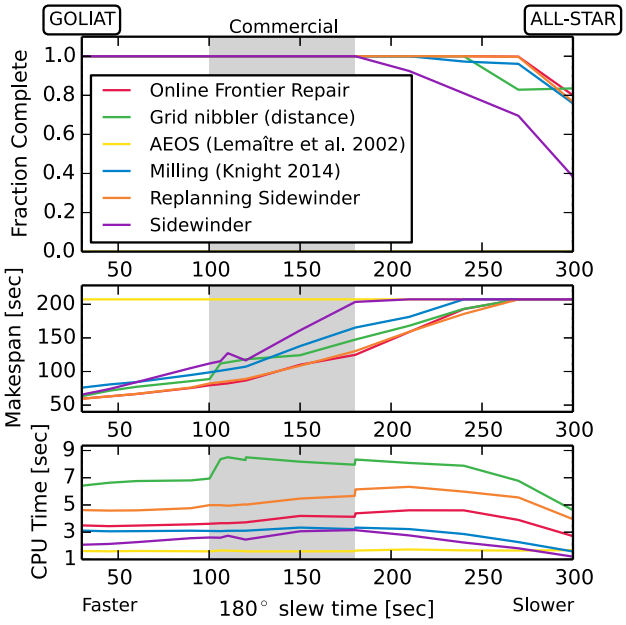


Figure 13: Performance under varying observer agilities

Observer Capability vs. Target Difficulty

When the observer is very agile and has a large FOV, path quality is less important. The easy/easy case is degenerate because CICLOP can satisfy the target with one image. In the easy observer, hard target case, Grid Nibbler generates the shortest schedule by requiring fewer images. The hard observer, easy target case shows the opposite - optimizing global planners generate shorter schedules with efficient pathing, despite requiring more images. We infer that lower agility requires more efficient paths because slews are more costly. With high agility, image number dominates path quality.

No algorithm completely satisfied the hard/hard case, even with multiple overflights. All algorithms consumed 10× more CPU and memory. Both Grid Nibbler variants covered more target area than the other algorithms.

Overall, Sidewinder used the least CPU time in all cases and the least memory in 3 of the 4. Grid Nibbler generally consumed more CPU time and memory than the other algorithms, with Online Frontier Repair and Replanning Sidewinder somewhere in between.

Overall Experiment Evaluation

All algorithms can produce admissible solutions, but no single algorithm is universally best. We recommend a

[5]Elliptical. Obtained from GOLIAT tracking TLE (Romanian Space Agency (ROSA) 2012).

[6]Circular. Reasonable when compared to commercial imagery satellite data sheets (Satellite Imaging Corporation 2017; MDA DigitalGlobe 2017).

[7]ALL-STAR is not agile enough and commercial imagery satellites (Hutin 2009; Satellite Imaging Corporation 2017; MDA DigitalGlobe 2017) do not use framing instruments

[8]The angle about an Euler rotation axis

[9]Data derived from (Dumitru 2006)

[10]Data obtained from (Ellison et al. 2013). No image duration value published for THEIA, assuming 1s.

Table 3: Comparison of algorithm performance for a cross-product of observer capability and target difficulty. Best in test values are green, - denotes failure to produce a schedule.

		Easy Observer (GOLIAT)					Hard Observer (Hybrid)				
	Algorithm	**CPU**	**RAM**	$\|m\|$	$\|A\|$	%	**CPU**	**RAM**	$\|m\|$	$\|A\|$	%
Easy target	Online Frontier Repair	2s	0.04MB	1s	1	100	4s	3.14MB	87s	54	100
	Replanning Sidewinder	2s	0.08MB	2s	2	100	5s	2.11MB	89s	54	100
	Milling (Knight 2014)	11s	0.04MB	11s	8	100	3s	0.37MB	107s	64	100
	Sidewinder	2s	0.05MB	2s	2	100	2s	0.30MB	117s	63	100
	Grid Nibbler (distance)	4s	0.05MB	1s	1	100	8s	4.13MB	118s	72	100
	Grid Nibbler (area)	3s	0.05MB	1s	1	100	14s	3.71MB	109s	52	100
Hard target	Online Frontier Repair	7s	3.21MB	87s	48	100	80s	22.80MB	39429s	387	32
	Replanning Sidewinder	9s	2.19MB	81s	41	100	-	-	-	-	-
	Milling (Knight 2014)	6s	0.42MB	118s	68	100	24s	3.80MB	39430s	343	30
	Sidewinder	3s	0.22MB	74s	43	100	19s	2.30MB	39430s	389	18
	Grid Nibbler (distance)	19s	4.52MB	96s	52	100	56s	23.20MB	39428s	391	34
	Grid Nibbler (area)	20s	3.73MB	70s	39	100	146s	23.30MB	39429s	392	41

portfolio approach, where a higher level scheduler considers a possible start time, then chooses the best algorithm for each circumstance, by either executing each algorithm and comparing makespans, or by evaluating a heuristic estimate model of each algorithm's makespan (Lewellen et al. 2017b).

Discussion

Overall, fewer tour points $|A|$ means shorter makespans $|m|$. However, tour quality has a greater impact on the less-agile observer: an efficient plan with more points can outperform a bad plan with fewer points (compare the hard observer/easy target instance of Online Frontier Repair vs. Grid Nibbler (area) in table 3).

All algorithms have linear complexity in number of tiles except for Online Frontier and Replanning Sidewinder, which are quadratic. Algorithmic complexity had negligible impact on schedule makespan and CPU runtime in our tests because $|A|$ was small.

These algorithms are only feasible on the upper end of current CubeSat processor modules. The algorithms themselves consume between 0.3 and 6.15 MB RAM, but our hasty implementation adds an extra 470 MB of non-algorithm memory overhead. This is too much RAM for low end PIC-based CubeSat modules, but reasonable for the 800 MHz CPU, 512 MB RAM Raspberry Pi compute module in the AAReST MirrorSat CubeSat (Underwood and Bridges 2015). Linearly scaling to an 800 MHz flight processor, our algorithms would require an estimated 10-65 seconds runtime to produce 70-118 seconds of schedule, faster than real time.

Future Work

We constrained our experiments to targets that are entirely within the field of regard. Larger targets could be decomposed into neighborhoods associated with visibility windows to accommodate multiple overflights.

Grid Nibbler was comparable to Online Frontier Repair, but was susceptible to dead ends due to its greedy approach. If Grid Nibbler looked ahead, it could retain the advantages of late commitment while avoiding dead ends. Future work should examine local optimization and relationships between next nibble heuristics.

Crude slew models were used for these experiments because detailed performance models of imagery satellites are typically not available to the public. Future efforts should constrain maximum angular rates and accelerations for slews, considering different agility about different spacecraft-fixed axes.

Conclusion

The three axis steerable 2D framing instrument area coverage planning problem was proven to be NP-complete. Four approximation algorithms for an optimal framing instrument path were outlined and compared in a computational experiment.

Naive approaches, such as applying pushbroom algorithms to a framing instrument or choosing a fixed target decomposition at start time, performed poorly. Locally scoped or replanning algorithms produced the shortest makespan schedules. Replanning Sidewinder and Online Frontier performed the best across the widest range of observer agilities. An algorithm from this paper outperformed the existing 1D (Lemaître et al. 2002) and Milling/Subdividable framing instrument (Knight 2014) algorithms in all experiments.

If the spacecraft is extremely agile, or if the target area is small relative to the imager footprint, the more sophisticated algorithms offer few advantages over a naive plan. The choice of algorithm matters most when the observer has only marginally sufficient agility to attempt a target.

Acknowledgments

The research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

References

Acton, C.; Bachman, N.; Semenov, B.; and Wright, E. 2016. Spice tools supporting planetary remote sensing. *ISPRS-International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 357–359.

Analytical Graphics Inc. 2017. Stk help/training part 4: Compute access. Web.

Balan, M., and Piso, M. 2008. GOLIAT project overview. In *5th Annual CubeSat Developers' Summer Workshop at the 22nd Annual AIAA/USU Conference on Small Satellites, Utah State University, Logan, Utah.*

Choset, H., and Pignon, P. 1998. Coverage path planning: the boustrophedon cellular decomposition. In *Field and Service Robotics*, 203–209. Springer.

Cormen, T.; Leiserson, C.; Rivest, R.; and Stein, C. 2009. *Introduction to Algorithms.* MIT Press.

Dumitru, C. 2006. GOLIAT. In *Proceedings of the 2006 Spring CubeSat Developers' Workshop.* Cal Poly San Luis Obispo.

Ellison, J.; Massone, G.; Ela, N.; Goh, A.; Smith, L.; Sobtzak, J.; Muralidharan, V.; Hayden, I.; Spetzler, B.; Vente, G.; Lopez-Dayer, A.; Montoya, R.; McGehan, Q.; Jeffries, T.; Cook, C.; and Campuzano, B. 2013. ALL-STAR system integration review. Web.

Hutin, C. 2009. Pleaides meeting with ffg. Web.

Itai, A.; Papadimitriou, C. H.; and Szwarcfiter, J. L. 1982. Hamilton paths in grid graphs. *SIAM Journal on Computing* 11(4):676–11. Copyright - Copyright] 1982 Society for Industrial and Applied Mathematics; Last updated - 2012-02-06.

Knight, R. 2014. Area coverage planning for subdividable framing instruments. In *Proceedings of the 12th International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS 2014).*

Lee, E. T.; Pan, Y.; and Chu, P. 1987. An algorithm for region filling using two-dimensional grammars. *International journal of intelligent systems* 2(3):255–263.

Lemaître, M.; Verfaillie, G.; Jouhaud, F.; Lachiver, J.-M.; and Bataille, N. 2002. Selecting and scheduling observations of agile satellites. *Aerospace Science and Technology* 6:367–381.

Lewellen, G.; Davies, C.; Byon, A.; Knight, R.; Shao, E.; Tran, D.; and Trowbridge, M. 2017a. A Hybrid Traveling Salesman Problem - Squeaky Wheel Optimization Planner for Earth Observational Scheduling. In Chien, S., and Augenstein, S., eds., *Proceedings of the 10th International Workshop on Planning and Scheduling for Space (IWPSS)*, 62–72.

Lewellen, G.; Trowbridge, M.; Shao, E.; Davies, C.; and Knight, R. 2017b. Estimating Time to Image Areas with Steerable 2D Framing Sensors. In Chien, S., and Augenstein, S., eds., *Proceedings of the 10th International Workshop on Planning and Scheduling for Space (IWPSS)*, 73–83.

MDA DigitalGlobe. 2017. WorldView-4 Data Sheet. Web.

Romanian Space Agency (ROSA). 2012. GOLIAT TLE. Web. http://www.goliat.ro/.

Rosenkrantz, D. J.; Stearns, R. E.; and Lewis, II, P. M. 1977. An analysis of several heuristics for the traveling salesman problem. *SIAM journal on computing* 6(3):563–581.

Satellite Imaging Corporation. 2017. IKONO Satellite Sensor. Web.

Semiconductor Components Industries, LLC. 2017. *AR0331 1/3Inch 3.1 Mp/Full HD Digital Image Sensor*, rev. 14 edition.

Stodden, D. Y., and Galasso, G. D. 1995. Space system visualization and analysis using the satellite orbit analysis program (soap). In *1995 IEEE Aerospace Applications Conference. Proceedings*, number 0, 369–387 vol.1.

Underwood, C., and Bridges, C. 2015. AAReST Spacecraft Update: Spacecraft Bus, Propulsion, ADCS, SSTL-50 CoreSat, RDV/Docking, OBDH and Comms. Web.