12th International Scheduling and Planning Applications Workshop (SPARK'19)

Post-proceedings of SPARK 2019

Bernardini, Sara; Parkinson, Simon; Talamadupula, Kartik; Yorke-Smith, Neil

Publication date
2019

Document Version
Final published version

Citation (APA)
Bernardini, S., Parkinson, S., Talamadupula, K., & Yorke-Smith, N. (Eds.) (2019). *12th International Scheduling and Planning Applications Workshop (SPARK'19): Post-proceedings of SPARK 2019*.

Bernardini, Parkinson, Talamadupula, Yorke-Smith (Eds)

SPARK 2019

12th International Scheduling and Planning Applications Workshop

11 July 2019
Berkeley, CA, USA

Co-located with ICAPS 2019

Preface

These working notes contain the papers presented at SPARK 2019: the 12th International Scheduling and Planning Applications woRKshop held on 11th July in Berkeley, California, USA. 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. For the first time, SPARK 2019 used an open review process.

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 modelling to technological to institutional issues. New real-world domains and problems are becoming more and more frequently affordable challenges for AI. The SPARK workshop series was established to foster the practical application of advances made in the international AI P&S community. Building on antecedent events, SPARK 2019 is the twelfth edition of a 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 webpage of the workshop series is found at: http://decsai.ugr.es/~lcv/SPARK/.

SPARK 2019 received eleven submissions. Each submission was reviewed by at least three programme committee members. The committee decided to accept all papers. The programme also included an invited talk (joint with the KEPS workshop), and two papers originally submitted to the COPLAS workshop (Ferrer et al. and Levinson), which are also included in these SPARK post-proceedings. The thirteen papers were discussed during three technical sessions.

We thank the Programme Committee for their commitment in reviewing, and the team at OpenReview.net. We thank the ICAPS 2019 workshop and publication chairs for their support.

Sara Bernardini, Royal Holloway University of London, UK
Simon Parkinson, University of Huddersfield, UK
Kartik Talamadupula, IBM Research AI, USA
Neil Yorke-Smith, Delft University of Technology, Netherlands

SPARK 2019 Co-Chairs

Organising Committee

Sara Bernardini	Royal Holloway University of London, UK
Simon Parkinson	University of Huddersfield, UK
Kartik Talamadupula	IBM Research AI, USA
Neil Yorke-Smith	Delft University of Technology, Netherlands

Programme Committee

Laura Barbulescu	Carnegie Mellon University
Anthony Barrett	NASA Jet Propulsion Laboratory
Mark Boddy	Adventium Labs
Gabriella Cortellessa	ISTC-CNR, Italian National Research Council
Lukas Chrpa	Czech Technical University in Prague
Minh Do	NASA Ames
Andreas Ernst	Monash University
Kshitij P. Fadnis	IBM Research AI
Simone Fratini	European Space Agency – ESA/ESOC
Christophe Guettier	SAFRAN
Mark Giuliano	Space Telescope Science Institute
Patrik Haslum	Australian National University
Derek Long	Schlumberger
Karen L. Myers	SRI International
Angelo Oddi	ISTC-CNR, Italian National Research Council
Nicola Policella	European Space Agency – ESA/ESOC
Cédric Pralet	ONERA Toulouse
Riccardo Rasconi	ISTC-CNR, Italian National Research Council
Bram Ridder	King's College London
Mark Roberts	Naval Research Lab, USA
Mauro Vallati	University of Huddersfield
Michael A. Schaffner	Sandia National Laboratories
Tiago Stegun Vaquero	NASA Jet Propulsion Laboratory / Caltech
Terry Zimmerman	North Seattle College
Yingqian Zhang	TU Eindhoven

Workshop Series Steering Committee

Gabriella Cortellessa	ISTC-CNR, Italy
Riccardo Rasconi	ISTC-CNR, Italy
Steve Chien	NASA JPL, USA
Neil Yorke-Smith	TU Delft, Netherlands

Contents

Aerospace Applications

Planning & Scheduling with Preferences

Automated Reasoning in Real Domains

Aerospace Applications

Constraint Integer Program Formulations for NASA Planning, Scheduling, and Autonomy Problems

Rich Levinson
Planning and Scheduling Group, NASA Ames Research Center, Moffett Field, CA 94035
rich.levinson@nasa.gov

Abstract

We present constraint integer program (CIP) formulations for NASA planning, scheduling and autonomy problems along with a benchmark path planning application. CIP combines constraint satisfaction (CS) with mixed integer programming (MIP) methods. Our focus is primarily on exploring the use of CIP for planning problems, where the solver must generate a set of actions (in addition to scheduling them), particularly in the context of an autonomous system, where the solver is embedded in real-time sense/plan/act execution cycle. We describe challenging NASA constraint optimization problems and explore trades between model variations, in order to spur discussion and to further improve our formulations and performance. We present results from performance experiments showing high sensitivity to model and problem configuration changes.

Introduction

We present constraint integer program (CIP) formulations for NASA planning, scheduling, and autonomy problems along with a benchmark path planning domain. CIP combines constraint programming (CP), mixed integer programming (MIP) and linear programming (LP) methods.

We are particularly interested in exploring how CIP may be used for planning applications, where the planner must generate the set of actions to perform in addition to scheduling them. We are also interested in using CIP as the planning component which is embedded in a real-time sense/plan/act execution cycle.

This paper is organized as follows: We first introduce the Rover domain, and then present CIP formulations for three scenarios from a simulated autonomous space habitat with integrated power and life support systems, identifying the planning and execution context where appropriate.

Solving Constraint Integer Programs (SCIP). We implemented the models presented below using SCIP (Achterberg 2009, Heinz and Beck 2011). SCIP is a hybrid solver that combines LP, MIP and CP into a unified Constraint Integer Program (CIP) system (https://scip.zib.de). SCIP's "under-the-hood" behavior involves tight integration between LP, MIP, and CP methods. When SCIP solves a problem, it automatically combines methods from these paradigms which share data and search state. For example, variable domain constraints from MIP may be shared with CP. This integration is mostly behind the scenes. For users who want to get under the hood, SCIP provides heuristic and search control options to manage the solving process details, and even to build custom constraint handler plug-ins.

Rover: Path planning in grid with obstacles

		O_2		Start
O_1		O_5		O_3
Goal		O_4		

Figure 1. Rover grid with 5 obstacles

To facilitate discussion, we begin with a classical planning benchmark domain known as Tileworld (Pollock and Ringuette 1990, Levinson 1995) which involves path planning and execution in a grid world. We present a variant of Tileworld called Rover with moving obstacles. The Rover domain is a simple pedagogical scenario that has been useful to develop our initial CIP formulations for planning problems, which we then applied to the actual NASA problems described in this paper. We also use this domain for performance experiments to understand the effects of model changes and scaling complexity.

Problem: Find an optimal sequence of moves, N, S, E, W to go from start position to goal position without stepping into a cell blocked by obstacle O_n.

Inputs define the x and y dimensions of the grid, the starting and goal positions for the rover, and the maximum execution time (max # of moves). These inputs are:

- *xMax* = grid x-dimension size
- *yMax* = grid y-dimension size
- *tMax* = the maximum execution time. Assuming each move takes one time unit, tMax = the maximum number of rover moves.
- *Starting position* (s_x, s_y)
- *Goal position* (g_x, g_y).

Let $T = \{0, \ldots, tMax\}$ be the set of all execution times in the plan window.

Rover position variables and constraints:
x_t = Rover x-position at time t, $0 \leq x_t \leq xMax, \forall t \in T$
y_t = Rover y-position at time t, $0 \leq y_t \leq yMax, \forall t \in T$

Move constraints (1) define the move choices at each time step. They are disjunction constraints which encode the 5 choices for moving: West, East, South, North, or no move. They also enforce the constraint that rover can move only one step at a time (no diagonal steps). $\forall t \in T$:

$$\begin{aligned}&((g_x < x_t) \wedge (x_{t+1} = x_t - 1) \wedge (y_{t+1} = y_t)) \quad \vee \\ &((x_t < g_x) \wedge (x_{t+1} = x_t + 1) \wedge (y_{t+1} = y_t)) \quad \vee \\ &((g_y < y_t) \wedge (y_{t+1} = y_t - 1) \wedge (x_{t+1} = x_t)) \quad \vee \\ &((y_t < g_y) \wedge (y_{t+1} = y_t + 1) \wedge (x_{t+1} = x_t)) \quad \vee \\ &((x_t = g_x) \wedge (y_t = g_y) \wedge (y_{t+1} = y_t) \wedge (x_{t+1} = x_t))\end{aligned} \quad (1)$$

Constraints (1) say: If goal is to on left (west) of rover, then decrement x and no change to y-position. If goal is on right (east) of rover, then increment x and no change to y-position. If goal is below (south of) rover, then decrement y and no change to x-position. If goal is above (north of) rover, then increment y and no change to x position. If rover is at the goal then there is no move.

Move constraints (1) assume there are no cul-de-sacs or blind alleys because the rover will never step in the opposite direction from the goal. We can easily remove this assumption by removing all terms containing g_x or g_y (the terms in (1) that compare rover position to goal position).

Disjunction constraints: We implement (1) using SCIP's *disjunction* constraint handler which ensures at least one of the disjuncts must be true in any feasible solution. Modeling disjunction is a key benefit of CIP compared to MIP. We find it more natural to model planning choices and mutually exclusive state descriptions with disjunction compared to use of slack variables or related methods required for strict MIP. All of the models in this paper use SCIP's disjunction constraint in some way.

Goal distance variables and constraints:
Rover *x* and *y* goal distances at time *t*, $\forall t \in T$:
dx_t = x-distance from goal at time t, $0 \leq dx_t \leq xMax$
dy_t = y-distance from goal at time t, $0 \leq dy_t \leq yMax$

Constraints (2) and (3) are disjunction constraints which bind the dx_t & dy_t variables to the absolute value of the goal distance at each time point.

$$\forall t \in T: \ (x_t + dx_t = g_x) \vee (x_t - dx_t = g_x) \quad (2)$$
$$(y_t + dy_t = g_y) \vee (y_t - dy_t = g_y) \quad (3)$$

Moving obstacles: We now extend the model to include obstacles which must be avoided. Obstacles may be moving if we are given their trajectories. The trajectories are vectors of integers rather than decision variables, so there is no additional computational complexity for moving vs. stationary obstacles. Let O be a set of N moving obstacles with known trajectories. $\forall o_i \in O$: o_{it}^x = x position of o_i at time t and o_{it}^y = y position of o_i at time t.

Blocked position indicator b_{it} is a binary variable indicating a plan where the rover is in the same position as an obstacle, so those solutions can be rejected. $\forall t \in T, \forall o_{it}$: $b_{it} \in \{0,1\}, b_{it} = 1 \Leftrightarrow (x_t = o_{it}^x) \wedge (y_t = o_{it}^y)$
b_{it} is true if and only if the rover's x and y positions equal x and y positions of obstacle o_i at the same time *t*.

Blocked position tracking constraints enforce semantics for the blocked position variable b_{it} indicating when rover and obstacle o_i are in the same position at time t. Constraints (4) include 5 disjuncts: IF rover x & y equal object o_i x & y at time t, THEN b_{it} = 1, ELSE (in all other cases) b_{it}= 0. $\forall t \in T, \forall i \in \{1, \ldots, N\}$:

$$\begin{aligned}&((x_t = o_{it}^x) \wedge (y_t = o_{it}^y) \wedge (b_{it} = 1)) \vee \\ &((x_t < o_{it}^x) \wedge (b_{it} = 0)) \vee \\ &((o_{it}^x < x_t) \wedge (b_{it} = 0)) \vee \\ &((y_t < o_{it}^y) \wedge (b_{it} = 0)) \vee \\ &((o_{it}^y < y_t) \wedge (b_{it} = 0))\end{aligned} \quad (4)$$

No blocked positions constraints reject any plan where the rover steps into a position blocked by an obstacle.

$$\sum_i^N \sum_t^{tMax} b_{it} = 0 \quad (5)$$

Objective:
Minimize $\sum_t^{tMax} dx_t + dy_t$

The objective is to minimize the sum of the x and y distances from the goal (Manhattan distance) for all times.

This model is very minimal and does not even include decision variables representing each move or indicating when the goal is reached. The sequence of moves can be inferred from a solution's x_t and y_t assignments. It is an indirect encoding of the model since the rover moves are not explicitly modeled.

The model presented above is version 2. The first version was complex and much slower. For comparison, we describe key parts of version 1 below. Version 1 is a direct encoding where x and y positions and moves are modeled with separate constraints, and each move is explicitly modeled with decision variables. It also tracked and rewarded progress towards subgoals (being aligned with the goal in either the x or y axis). It handles x and y positions and moves independently with constraints (6, 7, 8) below:

$$\begin{aligned}&((g_x < x_t) \wedge (x_{t+1} = x_t - 1) \wedge (d_{xt} = 1) \wedge (m_{xt} = 1)) \vee \\ &((x_t < g_x) \wedge (x_{t+1} = x_t + 1) \wedge (d_{xt} = 1) \wedge (m_{xt} = 2)) \vee \\ &((x_t = g_x) \wedge (x_{t+1} = x_t) \wedge (d_{xt} = 0) \wedge (m_{xt} = 3))\end{aligned} \quad (6)$$

$$\begin{aligned}&((g_y < y_t) \wedge (y_{t+1} = y_t - 1) \wedge (d_{yt} = 1) \wedge (m_{yt} = 4)) \vee \\ &((y_t < g_y) \wedge (y_{t+1} = x_t + 1) \wedge (d_{yt} = 1) \wedge (m_{yt} = 5)) \vee \\ &((y_t = g_y) \wedge (y_{t+1} = y_t) \wedge (d_{yt} = 0) \wedge (m_{yt} = 6))\end{aligned} \quad (7)$$

where d_{xt} & d_{yt} are x & y distances moved at time t.
m_{xt} & m_{yt} are the x & y move directions at t (1=East, 2=West, 3= no x-move.

Constraints (6) say: If goal is on left of rover, then decre-

ment x, if goal is on right, then increment x, and if rover's x position is same as goal x-position, then no move in x direction. Constraints (7) are the same, for y-positions.

Constraints (8) ensure that at any time the rover moves only in the x or the y direction, but not both (no diagonals).

$$\forall t: d_{xt} + d_{yt} \leq 1 \quad (8)$$

Model version 1 scaled so poorly that we tried the minimal approach of version 2, resulting in major improvements shown in figure 2:

Rover experiments

Test	V	D	Size	T	O	1st Sol	Bst sol	Opt sol	Sol time
1	1	↗	6x6	13	0	45	526	---	---
2	2	↗	6x6	13	0	20	20	---	---
3	1	↙	6x6	13	0	0.1	0.1	0.1	180
4	2	↙	6x6	13	0	---	---	0.1	3.4
5	1	↗	6x6	10	2	---	---	---	---
6	1	↙	6x6	10	2	20	192	---	---
7	2	↗	6x6	13	5	562	562	---	---
8	2	↙	6x6	13	5	---	---	0.28	1.4
9	2	↘	6x6	13	5	21	21	---	---
10	2	↖	6x6	13	5	158	158	---	---

Figure 2. Subset of Rover Experiment Results

Performance experiments for the rover domain involve varying the initial and goal positions, varying the grid size (not shown), and varying the number and positions of obstacles, and max number of moves. Figure 2 shows a subset of our experiment results for the rover. We chose this subset to highlight the differences between model version 1 (V1) and version 2 (V2), and to demonstrate the directional asymmetries we observe. In model V1 the x and y move choices were modeled using separate constraints. Model V2 is the very minimal model with "unified" move choice constraints to handle x and y movements together.

The columns in Table 2 are: V = the model version. D = direction. In tests 1 and 2, the rover starts in the lower left corner (0,0) and the goal is the upper right corner indicated by the ↗ arrow. Tests 3 and 4 are the opposite, starting in the upper right and goal in the lower left, indicated by the ↙ arrow. We observed significant performance asymmetries based on which direction the rover goes. *Size* indicates the width and height of the grid (e.g., 6 x 6). *T* is the maximum number of moves in a solution (max execution time). *O* shows number of the obstacles (if any). *1st sol* shows the time when the first solution was found. *Bst sol* shows the time when the best solution (lowest objective) was found. *Opt sol* shows the time when the optimal solution was found. *Sol time* shows when the SCIP solver converged on a solution and could verify that a previously found solution was in fact optimal. All times are in seconds. All tests had a maximum time limit of 10 minutes, after which SCIP returned any solutions it found up to that point.

The solver struggled when rover had to move to upper right (the "hard" direction). It is unclear why these asymmetries exist but they are extremely reproducible even after changing from V1 to version V2.

Tests 1 and 2 compare V1 vs. V2 without any obstacles. V2 solves it in 20 seconds compared to 526 seconds for V1. However, neither problem converged because it was the "hard" direction. Tests 3 and 4 are the same except in the "easy" direction, where both version 1 and 2 solved the problem in 0.1 seconds, but it took 180 seconds for V1 to prove optimality compared to 3.4 seconds for V2. Tests 5 and 6 both use version 1 with 2 obstacles, but in opposite directions. Test 5, the hard direction, produced no solution. Test 6, the easy direction, found a first answer in 20 seconds and the best solution at 192 seconds before timing out without converging. V1 could not solve any problems with more than 2 obstacles. Tests 7-10 all use V2, with 5 obstacles, but the direction is varied to test all 4 diagonals.

Even with V2, we see performance asymmetries favoring the direction from upper right to lower left. Test 8 shows the best performance is when both x and y must decrease to reach goal. Increasing y appears costlier than increasing x (test 7 vs test 9). We also found high sensitivity to SCIP heuristics. SCIP includes 7 different node selector heuristics to control selection of the next search node. We tried every one of the options and found that only depth-first search (DFS) produced any solutions before timing out at 10 minutes with no solutions. By default, DFS is the last heuristic SCIP chooses, so we had to override the default settings to tell SCIP to prefer DFS.

Autonomous space habitat

NASA has demonstrated autonomy software to control a simulated space habitat, similar to the International Space Station (Aaseng et al. 2018). The demo involved management of the habitat's power and life-support systems while a power distribution system fault occurs, reducing available power and energy. The habitat includes various instruments (power loads) like heaters, fans, and oxygen, CO2, and methane processing. Each load has different power demands, and some may have multiple power modes (off, low, high) with different demands depending on the mode.

Operational constraints must be satisfied. For example, two loads may need to stay synchronized so they are either both on or both off, or possibly they cannot be on at the same time. For example, only one heater may be on at any time. There are also periodic duty cycle constraints requiring loads to remain on (or off) for a given period of time within a larger repeating period. For example, a load must be ON for 15 minutes then off for 5 minutes.

An autonomous power control (APC) system provides low-level reactive "autonomy" for the power distribution so that if a fault occurs it can immediately safe the system by shutting off the lowest priority loads. APC ensures only

that the power system, at the lowest level, will not exceed power or energy constraints. It does not understand operational constraints (duty cycles and coordinated load requirements), and does not understand how to balance spacecraft-wide mission priorities and constraints involving other systems such as life support and avionics.

The *Vehicle System Manager (VSM)* maintains a higher-level view compared to APC. VSM has the job of producing the mission-level plan which maintains that system level perspective by integrating life support systems, science experiments and power management. We have implemented the VSM planner using SCIP.

Every 5 minutes, APC tells VSM how much power is available for the next 2 hours, then VSM produces a "power plan" covering the next 2 hours. The plan specifies the priority for each load and when the load should be turned on or off based on these spacecraft-wide constraints. The planner considers power demand for each load at each time to ensure that power demand never exceeds capacity and that the cumulative energy consumed during the entire 2-hour plan window never exceeds the total available energy.

When a fault occurs, APC will immediately safe the system by shedding the low-priority loads (using the load priorities set by the VSM planner) and then report the new state to VSM, including the type of fault (which components failed), the new (reduced) power availability, and a list of loads which were shed while safing the system. VSM then generates a new power plan to rebalance the load priorities and schedule based on the new situation.

We approach this as a job scheduling problem. Each load is a job to be scheduled on a single machine with a given power capacity. Multiple jobs can be scheduled at the same time on the single machine but the total power and energy demands cannot exceed the machine capacity.

In the nominal case, the objective is for all loads to fulfill their duty cycles and meet operational constraints. After a fault occurs, the planner must decide which loads to shed so that the new (reduced) energy and power availability constraints are not violated. Fault recover may involve adding actions too. If a load is turned off too long for fault recovery, then an additional load may need to be turned on to compensate, which may require turning something else off. Parts of this model were informed by our rover experiments. For example, maintaining a temperature setpoint is similar to the minimizing the rover's distance to the goal. We've extracted and simplified 3 scenarios from the habitat model which are explained below in isolation, although they are parts of a larger system.

Scenario 1 - Surviving a temporary power loss

Given Inputs:

maxTime = plan horizon

$t \in \{0, \dots, maxTime\}$ = time index.

maxPriority = maximum (largest) load priority

minEnergy = minimum energy available limit (minimum battery charge level)

p_t = maximum power available (capacity) at time *t*

L = Set of all loads.

Each load $l_n \in L$ includes the following properties:

l_n^{pri} = load l_n priority, $1 \leq l_n^{pri} \leq maxPriority$

l_n^{durMax} = load l_n max duration (the nominal duration, unless load must be shed).

l_n^{dmd} = load l_n power demand

l_n^{minOff} = minimum time load l_n may be off between two iterations (default is 0).

l_n^{maxOff} = max time load l_n may be off (default *maxTime*)

$synchronized(l_n, l_m)$ means loads l_n *and* $l_m$must start and end at the same time

Load iteration notation: l_{n_i}= the i[th] iteration of l_n (e.g., the 3[rd] time heater-2 is turned on). l_{n_i} are "jobs" to be scheduled. We use the term "job" interchangeably with "load iteration" in this paper.

Start time and duration variables:

$l_{n_i}^{start}$ = Start time for *i[th]* iteration of load *n*

$l_{n_i}^{dur}$ = Duration for *i[th]* iteration of load *n:*

$0 \leq l_{n_i}^{dur} \leq l_n^{durMax}$

For VSM, maxTime = 24, representing 24 quanta, each of 5 minute duration. Each time t represents a 5 minute quantum of real-time. We have 15 loads with maximum priority (lowest priority) = 15. Highest priority = 1.

Synchronization constraints: The Sabatier (SAB) and the Plasma Pyrolysis Assembly (PPA) are two loads which must be synchronized so that they are either both on or both off at any time. SAB removes carbon dioxide from the air using hydrogen and a catalyst, and produces methane as a byproduct. The PPA is used to recover hydrogen from methane byproduct. We model the requirement that SAB and PPA either must both be on or must both be off at the same time with synchronization constraint (9):

$$synchronized(l_n, l_m) \Leftrightarrow \left(l_{n_i}^{start} = l_{m_i}^{start}\right) \wedge \left(l_{n_i}^{dur} = l_{m_i}^{dur}\right) \quad (9)$$

isActive binary variables indicate if a given job is active at time t: $l_{n_i}^{isActive_t} = 1 \Leftrightarrow l_{n_i}$ is on at time t

$$\forall l_{n_i}, \forall t: \quad (10)$$

$$\begin{aligned}(((0 \leq l_{n_i}^{start} \leq t) \wedge (t \leq l_{n_i}^{start} + l_{n_i}^{dur}) \wedge (l_{n_i}^{isActive_t} = 1)) \vee \\ ((0 \leq l_{n_i}^{start} + l_{n_i}^{dur} < t) \wedge (l_{n_i}^{isActive_t} = 0)) \vee \\ ((t < l_{n_i}^{start}) \wedge (l_{n_i}^{isActive_t} = 0)))\end{aligned}$$

Constraints (10) say: If job starts before or at t, and ends at or after t, then job is active, otherwise job is not active.

Power and energy variables track resource usage:

d_t = total power demand at time t.

$$d_t = \sum_{\forall l_{n_i}} l_n^{dmd} \; l_{n_i}^{isActive_t}, \; \forall t \quad (11)$$

$\forall t$: e_t = available energy at time t.

Initial energy $e_0 = \sum_{t=0}^{maxTime} p_t$

$\forall t:\ e_{t+1} = e_t - d_t$ (12)

Power demand never exceeds available power:

$\forall t: d_t \le p_t$ (13)

Available energy always exceeds minimum energy limit:

$\forall t:\ \text{minEnergy} < e_t$ (14)

isShed binary variables and constraints indicate if load iteration $l_{n_i}$was shed (truncated). If l_{n_i} duration is shorter than the load's maximum duration, then isShed is true.

$$l_{n_i}^{isShed} = 1 \iff l_{n_i}^{dur} < l_n^{durMax}$$

$$\forall l_{n_i}:\ (\left(\left(l_{n_i}^{dur} < l_n^{durMax}\right) \wedge \left(l_{n_i}^{isShed} = 1\right)\right) \vee \left(\left(l_{n_i}^{dur} = l_n^{durMax}\right) \wedge \left(l_{n_i}^{isShed} = 0\right)\right)) \quad (15)$$

Separation constraints for duty cycles and periodic loads specify the distance between successive load iterations. Periodic duty cycles require that a load must remain on for some duration, then off for some duration, within a larger repeating period. A load is periodic if $l_n^{minOff} = l_n^{maxOff}$. For example, the Potable Water Dispenser (PWD) must be on for 15 minutes then off for 5 minutes. Constraints (16) enforce this periodic separation:

$$l_{n_{i+1}}^{start} = l_{n_i}^{start} + l_n^{durMax} + l_n^{maxOff} \quad (16)$$

Separation for non-periodic loads

$$l_n^{minOff} + 1 \le l_{n_{i+1}}^{start} - l_{n_i}^{start} - l_{n_i}^{dur} \le l_n^{maxOff} \quad (17)$$

One is added to the lower bound because this constrains the $l_{n_i}^{start}$lower bound for the *next* time **after** the load's off period. This ensures successor start time is not the same as predecessor end time (iterations must start and end at different times). Note that (16) constrains the "start-to-start" distance between the predecessor start and the successor start. In contrast, (17) constrains the "end-to-start" distance between the predecessor end and the successor start.

We tried using (17) for both periodic and non-periodic constraints (not using 16 at all). This had the appeal of using a single constraint instead of two, but it turned out to be a performance killer, probably because (17) includes $l_{n_i}^{dur}$ decision variables whereas (16) doesn't.

Backup jobs held in reserve. The exact # of load iterations required for an optimal solution is not known at model creation time (when we generate the SCIP variables and constraints). Depending on how many loads are shed, more jobs may be required. In nominal cases only one iteration of EXP is required because it typically remains on. However, fault recovery may require that we shed the first EXP job and then we need a new EXP iteration to schedule after fault recovery. To address this, we create "benchwarmer" jobs which are only scheduled if necessary to restart a load after it's been shed.

These benchwarmer jobs introduce several complications to the model. In particular, we must ensure that the reserve jobs are "inert", meaning their assigned start and duration times don't affect the objective function unless they are called into action. The separation constraints ensure that the reserve jobs are sequenced after the nominal jobs. We also want backup jobs to start at the plan horizon and also have a duration of 0 (so backup job durations don't affect the objective), but unlike the "nominal" jobs, which are penalized for being shed, we don't want the penalize the backup jobs if their duration is 0. Another complication is preventing premature shedding (stopping a job early) and starting a benchwarmer immediately to follow it. For example, if *EXP* should remain on for 10 ticks, we prefer a plan where EXP_1remains on for the duration and EXP_2 never starts, compared to shedding EXP_1after 5 ticks and then starting $EXP_2$to complete the remaining 5 ticks. We are considering an alternative approach where benchwarmers are created on-demand, only after a prior job is shed, rather creating them in advance as part of the initial model.

Objective: We maximize the durations of higher priority jobs and minimize the # of higher priority loads which are shed. We prefer to complete earlier load iterations and shed later ones. This is because we want to avoid premature shedding and want to keep the benchwarmers out of action until required as described above. We prefer to complete the first iteration if possible, and shed the second iteration, rather than cutting the first iteration short then starting the second iteration earlier prematurely. Thus, we want to favor scheduling the earliest iterations of the highest priority loads.

We define a load's weight: $l_n^w = 10^{(maxPri+1)-l_n^{pri}}$. This is the weighting factor for all load iterations l_{n_i} of load l_n. We then define the job weight $l_{n_i}^w = l_n^w + 1000/i$.

This scheme produces weights for the objective function such that each higher priority load has weights that are an order of magnitude higher the next lowest priority load, and earlier jobs are weighted higher than later jobs. Sample job weights for our example are shown in figure 3:

```
sab0: 1000000000000
sab1:  500000000000
ppa0:  100000000000
ppa1:   50000000000
pwd0:     100000000
pwd1:      50000000
pwd2:      33333333
exp0:      10000000
exp1:       5000000
exp2:       3333333
exp3:       2500000
```

Figure 3. Job weights $l_{n_i}^w$ used in objective function

Objective Function:

Minimize: $\sum_{\forall l_{n_i}} -l_{n_i}^w\ l_{n_i}^{dur} + l_{n_i}^w\ l_{n_i}^{isShed}$

This objective includes a reward for longer job durations

and a penalty for shedding jobs. The rewards and penalties are proportional to the job weight.

t	SAB	PPA	PWD	EXP	avail	demand	energy
0:	100	100	22.75	200	500	422.75	10500.00
1:	100	100	22.75	200	500	422.75	10077.25
2:	100	100	22.75	200	500	422.75	9654.50
3:	100	100		200	500	400.00	9231.75
4:	100	100	22.75	200	500	422.75	8831.75
5:	100	100	22.75	200	500	422.75	8409.00
6:	100	100	22.75	200	500	422.75	7986.25
7:	100	100		200	400	400.00	7563.50
8:	100	100	22.75		400	222.75	7163.50
9:	100	100	22.75		400	222.75	6940.75
10:	100	100	22.75		400	222.75	6718.00
11:	100	100		200	400	400.00	6495.25
12:	100	100	22.75		400	222.75	6095.25
13:	100	100	22.75		400	222.75	5872.50
14:	100	100	22.75		400	222.75	5649.75
15:	100	100			400	200.00	5427.00
16:	100	100	22.75		400	222.75	5227.00
17:	100	100	22.75	200	500	422.75	5004.25
18:	100	100	22.75	200	500	422.75	4581.50
19:	100	100		200	500	400.00	4158.75
20:			22.75	200	500	222.75	3758.75
21:			22.75	200	500	222.75	3536.00
22:			22.75	200	500	222.75	3313.25

Figure 4. VSM planner solution

Figure 4 shows a sample VSM solution. Each row represents a time, t. The columns SAB, PPA, PWD, EXP represent the power demand (watts) from each load at time t if the load is scheduled to be on at t (entry is blank if load is off). The *avail* column is available power, *demand* is the total power demand from all loads, and *energy* is remaining energy. Loads are shown in decreasing priority from the left: SAB is highest priority and EXP is lowest (first to be shed). These priorities reflect overall system-wide priorities: First protect human life, then protect overall mission, then protect science, then protect individual subsystems. SAB, PPA and PWD are all life support systems which are higher priority than EXP, which is a freezer containing science specimens (to preserve the specimens, it shouldn't be off for more than 30 minutes). Notice PWD's duty cycle periodicity, which is on for 3 ticks then off for 1. Also note that SAB and PPA are synchronized in their duty cycles.

Figure 4 illustrates a reduced power scenario. Available power (avail) decreases from 500 to 400 watts, from t = 7 through t = 16 (highlighted by the box). This forces the planner to shed EXP which has a max-separation constraint that it may not be off for more than 6 time units (30 minutes). This forces the planner to turn EXP back on at t = 11, but then turn it off again for another 5 time units, so that EXP is never off too long.

Originally this solution took 1033 seconds (17.2 mins) to find. We then changed the start time and duration decision variables from integer to continuous and it took 1/3 of the time, solving this same problem in only 330 seconds (5.5 mins). SCIP's solution process involves first relaxing the integral constraints, then solving the LP, then reintroducing the integral constraints. Since our start times and durations are integral seconds, it seemed natural to model them as integers, but clearly SCIP incurs significant overhead in relaxing then reintroducing the integral constraints.

Continuous replanning: The plan window rolls forward. Every 5 minutes, the plan window's lower and upper bounds both increase by five minutes (a "quantum"). The plan is updated to reflect the new time bounds. Model variables and constraints from the past may be discarded and new ones for the future may be created to cover the new quantum extension. Load iterations are created as necessary on each quantum update.

Plan Execution causes decision variables to be fixed to their actual execution times. As the plan is executed, VMS sends start and stop commands to each load at the scheduled times. Past start and stop times are now known, so those start and end times are fixed to the actual time when those commands were sent.

From an execution perspective, if a fault forces VSM to stop a job earlier than planned, VSM simply sends commands to the loads to turn off. From the planning perspective, it's more indirect. We only model job start times and durations (not stop times), so we cannot set the stop time directly. Instead we shorten (and fix) the *duration* of the current SAB and PPA iterations as follows:

$$SAB_i^{dur} = faultTime - SAB_i^{start} \quad (18)$$

$$PPA_i^{dur} = faultTime - PPA_i^{start} \quad (19)$$

where *faultTime* is the time when the fault starts.

If a power fault causes us to lose a battery, APC informs VSM about the reduced energy capacity. VSM determines it must shut down the lowest priority load, EXP (a science experiment freezer), but not for more than 30 minutes. This is modeled by separation constraint (17) and can be rewritten as: $EXP_i^{start} + EXP_i^{dur} \leq EXP_{i+1}^{start} \leq 30$, where EXP_{i+1} is the next EXP iteration after *faultTime* and EXP_i is the iteration that was shut stopped at *faultTime*.

Scenario 2 - Contingent Action Planning: This second scenario involves planning (adding actions to the plan) rather than scheduling times for a given set of actions. In this scenario, fault recovery involves conditionally adding a new load to the plan, compared to prior scenario where we were strictly shedding loads. This is currently implemented as a standalone SCIP model but some version will eventually be integrated into the larger VSM application.

In this scenario, we have the SAB and PPA loads as before. The loads SAB and PPA should both remain on until a fault occurs. A fault occurs when the PPA collects too much residue to perform correctly. The only option is to turn off the PPA to perform a cleaning action which attempts to fix the problem. The duration of the cleaning action depends on how much residue has collected. Since SAB and PPA are synchronized, we must also turn off SAB while the PPA is off for cleaning. However, if SAB is turned off too long, then it will cool down so much that an extra action "reheat" must be added to the plan to reheat

the SAB after cleaning has resolved the problem and before turning both SAB and PPA back on.

In other words, depending on how long SAB remains off, we may have to add an additional recovery action to the plan (to reheat the SAB before turning it back on). Specifically, if SAB remains off for 4 time units or less, then we don't need the contingent reheat action, but if it remains off more than 4 time units (because the cleaning action is taking a long time), then we must add the reheat action to the plan. The model for this contingent action behavior is below. For brevity, we omit the PPA variables and constraints to illustrate the concept using SAB only. Since PPA and SAB are synchronized (constraint 9) they have nearly identical specifications.

In this simplified model we define 5 jobs:
SAB_1 = first iteration of SAB load (before fault).
SAB_2 = second iteration of SAB (after fault is resolved)
c = clean the PPA (fault recovery action)
r = reheat the SAB if necessary (contingent action)
f = fault "job" (exogenous activity triggered by sensors)

Variables:
$SAB_1^s,\ SAB_1^d, SAB_1^e$ = start, duration, end times for SAB_1
$SAB_2^s,\ SAB_2^d, SAB_2^e$ = start, duration, end times for SAB_2
$c^s,\ c^d,\ c^e$ = start, duration, end times for c
r^s, r^d, r^e = start, duration, end times for r
f^s, f^d, f^e = start, duration, end times for f

Duration constraints:

$$SAB_1^e = SAB_1^s + SAB_1^d \quad (21)$$
$$SAB_2^e = SAB_2^s + SAB_2^d \quad (22)$$
$$c^e = c^s + c^d \quad (23)$$
$$r^e = r^s + r^d \quad (24)$$
$$f^e = f^s + f^d \quad (25)$$

Sequence constraints:

SAB_2 starts after SAB_1 ends: $SAB_1^e \leq SAB_2^s$ (26)
SAB_1 ends when fault starts: $SAB_1^e = f^s$ (27)
$cleaning$ starts when fault starts: $c^s = f^s$ (28)
$cleaning$ ends when fault ends: $c^e = f^e$ (29)

The fault f is an exogenous activity, which is triggered by a PPA sensor. The fault is modeled as a "job" with start, duration and end times, just like other jobs, except the start time and duration are determined during execution by a sensor which measures the PPA residue buildup. When a sensor/state estimator tells VSM the fault has begun, then VSM fixes f^s to the current execution time, and when receives a message the fault has been repaired, then it fixes f^e to the time when fault is fixed. Before f^s is fixed to an actual value, the planner maximizes f^s (expressed in the objective function). If the fault never happens, this job should start at the end of (outside) the plan horizon.

Conditional temporal network constraint:

$$(((c^d \leq 4) \wedge (SAB_2^s = c^e)) \ \vee \ ((4 < c^d) \wedge (r^s = c^e) \wedge (SAB_2^s = r^e))) \quad (30)$$

Constraint (30) says: If the cleaning duration is less than or equal to 4 tine units, then SAB_2 starts when the cleaning ends, otherwise the contingent reheat action starts when cleaning ends, and SAB_2 starts after the reheat action ends. If cleaning takes too long, then the topology of the temporal constraint network is modified by splicing the contingent reheat action into place in between cleaning and restarting SAB. Constraints (30) define a conditional temporal network, where the network topology and distance constraints are conditional on the length of the cleaning operation. This approach is related work in constraint networks (Allen 1991) and constraint-based planning systems (Muscettola et al. 2002).

Objective: Minimize: $-SAB_1^d - SAB_2^d - c^s - r^s - f^s$

An optimal solution has the longest durations for SAB_1 and SAB_2, and the latest start times for SAB_2, c, r and f. If the fault never occurs then SAB_2, c, r and f never start (their start times are outside the plan horizon).

Scenario 3 - Thermostat with multi-mode heaters:

In this final scenario, we maintain a temperature setpoint using 2 heater loads. Like the scenario in the prior section, this has been implemented as a standalone problem but a version of it will be integrated into the larger VMS model.

This scenario was designed to model loads with variable power demands. Each heater may be in three different modes: Off, Low-power, or High-power. Only one heater may be on at any time. The objective is to minimize the difference between temperature and a setpoint, and to minimize the power consumption.

This model leverages methods developed for the Rover. Here the current temperature corresponds to the rover's current position, and the setpoint to the rover's goal position. We are minimizing the temperature difference between the current temperature and the setpoint, using similar variables and constraints as the Rover used to minimize distance from the goal.

There are two heaters, H1 and H2, represented by integer decision variables with range [0,2], representing 3 power levels (0 = off, 1 = low power, 2 = high power). High power demands more power and produces more heat output. The low and high power levels each have associated demand (power consumption) and output (representing temp increase, in this simplified example).

- H1 = 0: Heater1 is off (no power is consumed and no heat output is produced)
- H1 = 1: Heater1 is on low power: Demand = 1 unit of power consumed, and output = 2 temp units (increases temp by 2).
- H1 = 2: Heater1 is on high power: Demand = 2 units of power consumed, and output = 4 temp units (increases temp by 4).

The model includes *ambient cooling*. Temperature decreases by 1 unit each tick (if no heater is on, then the temp

decreases by 1 each tick. If heater is on, then its output is added to this ambient decrease).

Given Inputs:
maxTime = the plan length (each step takes one time unit)
maxTemp = maximum temperature
s_t = vector of set point temperatures for each time t
c_t = vector of maximum power capacity for each time t
a_t = vector of ambient cooling rate at each time t. Ambient conditions cause temperature to decrease by this amount on each time step.
tmp_0 = initial temperature
Variables:
$tmp_t =$ the temperature at time t, $\forall\, t < maxTime$
p_t = available power at time t, $\forall\, t < maxTime$
e_t = available energy at time t, $\forall\, t < maxTime$
d_t= absolute value of the difference between current temperature tmp_t and set point *s* at time t.
$d_t \in \{0, \dots, maxTemp)$, $\forall\, t < maxTime$.
$H1_t^m \in \{0,1,2\} =$ H1 power mode at time t
$H2_t^m \in \{0,1,2\} =$ H2 power mode at time t
$H1_t^d$ = H1 power demand at time t
$H2_t^d$ = H2 power demand at time t
$H1_t^o$ = heat output produced by H1 at time t
$H2_t^o$ = heat output produced by H2 at time t

We noticed performance differences between equivalent models, where both models produce feasible results using different constraints. We evolved three different model versions of heater constraints, with very different performance results (described at the end of this section).

Model Version 1: Each heater is modeled separately:
Heater state power demand and output constraints: $\forall t$:

$$\begin{aligned}&\big((H1_t^m = 0) \wedge (H1_t^d = 0) \wedge (H1_t^o = 0)\big) \vee \\ &\big((H1_t^m = 1) \wedge (H1_t^d = 1) \wedge (H1_t^o = 2)\big) \vee \\ &\big((H1_t^m = 2) \wedge (H1_t^d = 2) \wedge (H1_t^o = 4)\big)\big)\end{aligned} \quad (31)$$

$$\begin{aligned}&\big((H2_t^m = 0) \wedge (H2_t^d = 0) \wedge (H2_t^o = 0)\big) \vee \\ &\big((H2_t^m = 1) \wedge (H2_t^d = 1) \wedge (H2_t^o = 2)\big) \vee \\ &\big((H2_t^m = 2) \wedge (H2_t^d = 2) \wedge (H2_t^o = 4)\big)\big)\end{aligned} \quad (32)$$

Constraints (31,32) are disjunctions of each heater's three possible states: (mode is off, demand = 0, output = 0) or (mode is low, demand = 1, output = 2) or (mode is high, demand = 2, output = 4).

One-Heater constraints ensure that at most one heater is on at any time: $\big(H1_t^m = 0\big) \vee \big(H2_t^m = 0\big),\ \forall t$ (33)

Model Version 2: In this version we replace the previous one-heater constraint (33) with constraint (34) below, which ensures one heater at a time based on temperature and setpoint variables.

$$\begin{aligned}&\big((s_t \le tmp_t) \wedge (H1_t^m = 0) \wedge (H2_t^m = 0)\big) \vee \\ &\big((tmp_t < s_t) \wedge (1 \le H1_t^m \le 2) \wedge (H2_t^m = 0)\big) \vee \\ &\big((tmp_t < s_t) \wedge (1 \le H2_t^m \le 2) \wedge (H1_t^m = 0)\big)\end{aligned} \quad (34)$$

Constraints (34) describe three possible states: (setpoint ≤ temperature, and both heaters are off), or (temperature < setpoint, and H1 is on, and H2 is off), or (temperature < setpoint, and H2 is on, and H1 is off). This was an intermediate step towards model version 3.

Model Version 3: Unified constraints. This final version replaces all prior constraints with a single disjunction constraint describing 5 operating states. Each disjunct fully specifies the state vector for each heater including mode, power demand and output. This was motivated by performance improvements we saw after making similar changes to the Rover model. In this version, all previous thermostat constraints (31-34) are replaced with (35) shown below:

$$\begin{aligned}&\big((s_t \le tmp_t) \wedge (H1_t^m = 0) \wedge (H1_t^d = 0) \wedge (H1_t^o = 0)\\ &\quad \wedge (H2_t^m = 0) \wedge (H2_t^d = 0) \wedge (H2_t^o = 0)\big) \vee \\ &\big((tmp_t < s_t) \wedge (H1_t^m = 1) \wedge (H1_t^d = 1) \wedge (H1_t^o = 2)\\ &\quad \wedge (H2_t^m = 0) \wedge (H2_t^d = 0) \wedge (H2_t^o = 0)\big) \vee \\ &\big((tmp_t < s_t) \wedge (H1_t^m = 2) \wedge (H1_t^d = 2) \wedge (H1_t^o = 4)\\ &\quad \wedge (H2_t^m = 0) \wedge (H2_t^d = 0) \wedge (H2_t^o = 0)\big) \vee \\ &\big((tmp_t < s_t) \wedge (H2_t^m = 1) \wedge (H2_t^d = 1) \wedge (H2_t^o = 2)\\ &\quad \wedge (H1_t^m = 0) \wedge (H1_t^d = 0) \wedge (H1_t^o = 0)\big) \vee \\ &\big((tmp_t < s_t) \wedge (H2_t^m = 2) \wedge (H2_t^d = 2) \wedge (H2_t^o = 4)\\ &\quad \wedge (H1_t^m = 0) \wedge (H1_t^d = 0) \wedge (H1_t^o = 0)\big)\end{aligned} \quad (35)$$

Constraints (35) are a disjunction of these 5 possible states: (setpoint ≤ temperature, and H1 & H2 are both off) or (temperature < setpoint, and H1 is low and H2 is off) or (temperature < setpoint, and H1 is high and H2 is off) or (temperature < setpoint, and H2 is low and H1 is off) or (temperature < setpoint, and H2 is high and H1 is off).

Temperature difference constraints bind $d_t$to the absolute value of temp difference from setpoint at each time. These constraints are based on the rover goal distance constraints (2) and (3). $\forall t$:

$$(tmp_t + d_t = s_t) \vee (tmp_t - d_t = s_t), 0 \le d_t \quad (36)$$

Temperature change constraints:

$$\forall t: tmp_{t+1} = tmp_t + H1_t^o + H2_t^o - a_t \quad (37)$$

Available power constraints:

$$\forall t: p_{t+1} = c_t - H1_t^d - H2_t^d \quad (38)$$

Available energy constraints:
Initial energy $e_0 = \sum_{t=0}^{maxTime} a_t$.

$$\forall t:\ e_{t+1} = e_t - H1_t^d - H2_t^d \quad (39)$$

Objective: The objective is to minimize the sum of the temperature differences from setpoint for all times, similar to the rover minimizing goal distance, while also minimizing power demand: Min: $\sum_{\forall t} d_t + H1_t^d + H2_t^d$.

Reactive execution proceeds through a sense/plan/act cycle. At each execution time t, the actual current temperature is read from sensors and the variable tmp_t is fixed to

the sensed temperature reading for t = the current execution time step. All future temperatures ($tmp_{t+1\ldots}$) are predicted by the planner using the above constraints, but the first tmp_t in each execution cycle comes from the sensor. After fixing tmp_t to the sensed value, SCIP is called to resolve the problem.

Model Version	maxTime	1st sol	Opt sol	Solve time
1	6	4.05	6.92	32.07
2	6	---	11.99	39.48
3	6	0.86	1.46	4.56
1	8	60.57	100.6	632.37
2	8	---	140.33	491.5
3	8	---	1.4	30.85
1	10	411.49	---	---
2	10	897.92	---	---
3	10	---	12.23	222.72

Figure 5. Thermostat Results

Thermostat results are shown in Figure 5. For each model version, we compare the times required to find a first solution, the time until finding the optimal solution, and the "Solve time", which is time when the solver converged to prove optimality and returns before reaching the 30-minute solver time limit. All times are in seconds.

Note the differences between 1st sol, opt sol, and solve time. We compared three different problem sizes: 6, 8 and 10 (maxTimes), representing increasing difficulty. Most notable is how well version 3 performs, which mirrors the performance improvement we saw when we made similar model changes to the rover. Version 3 strongly outperforms the others in every metric. It's the only version to find an optimal solution for the largest problem (maxTime = 10). The other two versions timed out after 30 minutes on this largest problem without converging. Version 2 only outperforms version 1 in one case: a faster solve time for the middle-sized problem (maxTime = 8).

Conclusion and future work

We presented CIP formulations for three NASA scenarios from an autonomous space habitat project, focusing on the use of CIP for planning and execution scenarios where the set of actions to be scheduled is not known in advance and execution-time faults require reactive replanning.

Our overall conclusion is that it is possible to model dynamic planning and execution problems using SCIP. In particular the disjunction constraint is a natural way to model action choices and action outcomes as disjunctive states, which bind binary indicator variables to state descriptions. Such planner choices would be much harder to model with pure LP or MIP.

Performance within a real-time context is a key challenge. We demonstrated initial performance results showing solution times are very sensitive to model changes and problem configuration. Initial results show we can significantly improve performance by unifying some constraints (but not necessarily all of them). We also discovered significant performance improvement when we changed VSM decision variables from integer to continuous.

We will continue experiments to explore how formulation variations affect performance, and to better understand SCIP's search heuristics and parameters. We will try to identify the cause of the tenacious directional performance asymmetry we observed in the Rover experiments. We have not yet tested changing all integer variables to continuous with all models presented above, but intend to do so. We are updating and integrating the VSM subproblems described above into a single model, and extending the model to support a new set of loads and fault scenarios, and integration with new habitat subsystem simulators.

Acknowledgements

Thanks to the NASA Ames Autonomy Systems and Operations team for helpful discussions to clarify and spell out the problem scenarios which have been presented above. In particular, thanks to Jeremy Frank for fleshing out and articulating the autonomous habitat operating constraints and scenarios. This work was funded by the NASA Advanced Exploration Systems Program.

References

Aaseng, G.; Frank, J.; Iatauro, M.; Knight, C.; Levinson, R.; Ossenfort, J.; Scott, M.; Sweet, A.; Csank, J.; Soeder, J.; Carrejo, D.; Loveless, A.; Ngo, T.; and Greenwood, Z. 2018. Development and Testing of a Vehicle Management System for Autonomous Spacecraft Habitat Operations, In Proceedings of AIAA 2018, Orlando FL.

Achterberg, T.,2009. SCIP: solving constraint integer programs. Math.Prog.Comp.,Vol.1,2009,pp.1–41.

Allen, J. 1991. Temporal reasoning and planning. In Reasoning about plans, Ronald J. Brachman, James F. Allen, Henry A. Kautz, Richard N. Pelavin, and Josh D. Tenenberg (Eds.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA 1-67.

Heinz, S., Beck. J.C. 2011. Solving Resource Allocation/Scheduling Problems with Constraint Integer Programming. Proceedings of COPLAS 2011, pp 23-30.

Levinson, R. 1995. A General Programming Language for Unified Planning and Control. Artificial Intelligence, special issue on Planning and Scheduling. Vol. 76. Elsevier Press. July 1995.

Muscettola, N., Dorais, G., Fry, C., Levinson, R., Plaunt, C. 2002. IDEA: Planning at the Core of Autonomous Reactive Agents. Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space.

Pollack, M.E. and Ringuette, M. 1990 Introducing the Tileworld: Experimentally evaluating agent architectures. Proceedings of AAAI 90. Boston, MA

Automated Science Scheduling for the ECOSTRESS Mission

Amruta Yelamanchili, Steve Chien, Alan Moy, Elly Shao, Michael Trowbridge, Kerry Cawse-Nicholson, Jordan Padams, Dana Freeborn
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109
{firstname.lastname}@jpl.caltech.edu

Abstract

We describe the use of an automated scheduling system for observation policy design and to schedule operations of the NASA (National Aeronautics and Space Administration) ECOSystem Spaceborne Thermal Radiometer Experiment on Space Station (ECOSTRESS). We describe the adaptation of the Compressed Large-scale Activity Scheduler and Planner (CLASP) scheduling system to the ECOSTRESS scheduling problem, highlighting multiple use cases for automated scheduling and several challenges for the scheduling technology: handling long-term campaigns with changing information, Mass Storage Unit Ring Buffer operations challenges, and orbit uncertainty. The described scheduling system has been used for operations of the ECOSTRESS instrument since its nominal operations start July 2018 and is expected to operate until mission end in Summer 2019.

Introduction

NASA's ECOSTRESS mission (NASA 2019) seeks to better understand how much water plants need and how they respond to stress. Two processes show how plants use water: transpiration and evaporation. Transpiration is the process of plants losing water through tiny pores in their leaves. Evaporation of water from the soil surrounding plants affects how much water the plants can use. ECOSTRESS measures the temperature of plants to understand combined evaporation and transpiration, known as evapotranspiration.

ECOSTRESS launched on June 29, 2018 to the ISS (International Space Station) on a Space-X Falcon 9 rocket as part of a resupply mission. The instrument is attached to the Japanese Experiment Module Exposed Facility (JEM-EF) on the ISS and targets key biomes on the Earth's surface, as well as calibration/validation sites. Other science targets include cities and volcanoes. From the orbit of the Space Station (Figure 1), the instrument can see target regions at varying times throughout the day, rather than at a fixed time of day, allowing scientists to understand plant water use throughout the day.

The instrument used for ECOSTRESS is a thermal infrared radiometer. A double-sided scan mirror, rotating at a constant 25.4 rpm, allows the telescope to view a 53°-wide nadir cross-track swath with one scan per 1.18 seconds. The nominal observation unit is a scene, made up of 44 scans, and takes roughly 52 seconds to acquire. For simplification of operations, we consider that ECOSTRESS scenes are 52 seconds long. About 1000 scenes may be acquired in a given week. Figure 2 shows a set of planned observations over North America. Each square represents one 52-second long scene.

CLASP (Knight and Chien 2006) was initially used prelaunch as a tool to analyze the addition of a new science campaign. CLASP was then used for operations to generate command sequences for the instrument. The command sequences are translated from the observation schedule generated by CLASP, and include other time and location dependent instrument actions besides observations, such as hardware power cycles through high radiation environments.

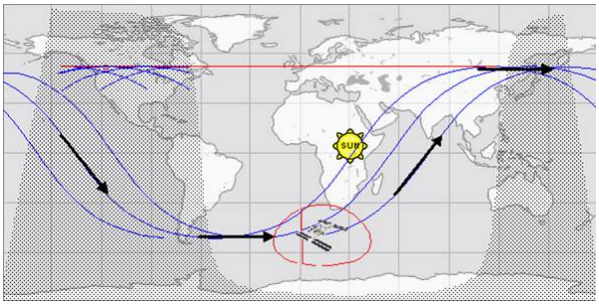


Figure 1: Three Orbital Tracks of the ISS (Robinson 2013)

Each mission comes with its own set of challenges, and there were three specifically that required adaptations to CLASP as follows.

- ECOSTRESS has a long-term science campaign that we need to satisfy. From week to week, the orbital ephemeris can change, and thus the schedule needs to be updated each week. We need to be able to account for previously executed observations when scheduling for the future.
- An issue with the instrument Mass Storage Unit (MSU) was discovered, and rather than performing an instrument firmware update, we proposed a ground-based solution that accounts for this additional complexity in the data modeling in the schedule.
- The uncertainty in the orbital ephemeris (predictions of

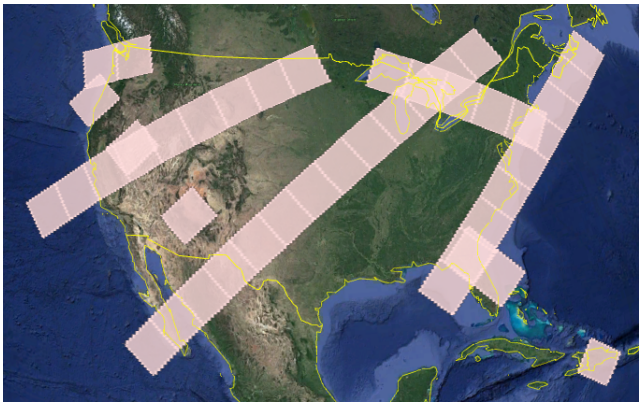


Figure 2: Observations Over North America

the spacecraft location) required scheduling additional observation time to ensure no targets are missed.

In the remainder of this paper, we describe these operational challenges and how we addressed them successfully. We also validate our methods used through computational analysis.

Initial Scheduling Problem

CLASP recieves as input (Figure 3) the ephemeris of the ISS (predicted time-tagged locations), instrument constraints, and a set of set of science campaigns, which are made up of:

- target regions of interest on the Earth's surface (Figure 4)
- illumination constraints
- a priority

We want to produce an observation schedule to view these regions as many times as possible while respecting constraints such as memory capacity, downlink rate, and keepout zones (e.g. high radiation environments) where we do not want to take any observations. Science campaigns can be target regions or single point locations. We generate a gridded approximation of target regions for faster computation, to get a set of target points.

CLASP uses the CSPICE Toolkit provided by the Navigation and Ancillary Facility (NAIF) (Acton 1996) at JPL to do geometric reasoning regarding the visibility swaths of instruments from the spacecraft they are attached to. The size, shape, and location of the swaths depend on the position and orientation of the spacecraft, and the field-of-view of the instrument. CLASP has the capability to schedule instruments that can point off-nadir, but ECOSTRESS specifically has no pointing capability, so each observation spans the whole range of its field-of-view. The planning horizon is broken up into fixed duration observations, and CLASP computes the intersection between the target grid points and the observations. We use a one-pass greedy scheduling algorithm to place observations according to the priority of the targets they cover.

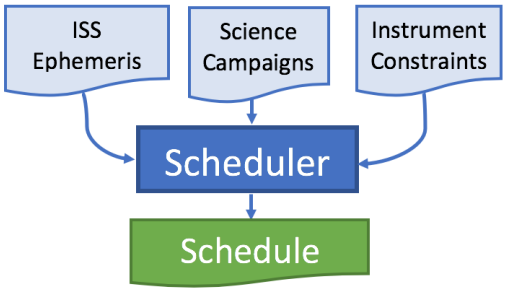


Figure 3: Scheduler inputs and outputs

Problem Statement

The CLASP problem statement (Knight and Chien 2006):
Given:

- a set of regions of interest $R = \{r_1, ..., r_n\}$
- a temporal knowledge horizon (hst, het) over which we know the vehicle's activities
- a set of observation opportunities $O = \{o_1, ..., o_n\}$ within the horizon (hst, het) where each $o_i \in O$ consists of a start $(o.start)$ and a duration $(o.duration)$
- a set of instrument swaths I $=\{i_1, ..., i_n\}$ where $\forall(o_i \in O)\exists(r_i, i_i) \mid (\text{grid}(o_i) \in \text{grid}(r_i)) \wedge (\text{grid}(o_i) \in \text{grid}(i_i))$
- a scoring function $U(r_i)$
- keepout zones where observations should not be taken
- a bound on memory M_{max}
- a rate at which memory is used while the instrument is on $\dot{M}_{fill}$
- a rate at which memory is recovered during downlink $\dot{M}_{drain}$, which occurs when the instrument is not observing and is not in a keepout zone

Our goal is to select $A \subseteq O$ to maximize U(r_i) $\forall r_i \in R$ subject to instrument constraints involving available memory and keepout zones.

We introduce new and modified ECOSTRESS-specific components to the problem statement in subsequent sections.

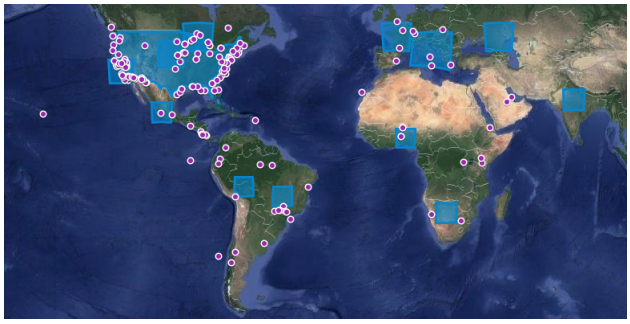


Figure 4: Science Campaigns can be made up of regions (blue boxes) or point targets (purple dots)

Sliding Window Scheduling and Changing Ephemeris

If we had perfect knowledge of the future and a perfect observer, we could simply schedule the entire mission at once and feed the instrument slices of the schedule to execute. Unfortunately, there are uncertainties that require the schedule to be updated:

- new or modified science campaigns
- special maneuvers for spacecraft docking that could put the instrument in an unsafe position
- thruster burns to counteract orbital decay that change the trajectory

The goal for most of the ECOSTRESS science campaigns is to observe them whenever possible in daylight, to see how their water use changes at different times throughout the day over an extended period of time. During pre-operations analysis, CLASP was used to understand how to effectively make use of unused data volume. A new campaign was thus inserted into the ECOSTRESS science operation goals - to construct daytime maps of the global landmass. We found that attempting to construct one global map per month would not violate any instrument memory constraints, and allow all of the primary science campaigns to still be fulfilled as much as possible.

The ECOSTRESS payload is commanded weekly after a new ISS ephemeris prediction is received, uploading two weeks worth of command sequences to the instrument. Operationally, only the first week of sequences will get executed before the next set of sequences is uploaded, with the second week only being executed if, for some reason, a new schedule is not able to be uploaded the next week. Since the global map takes four weeks to construct, but only two weeks worth of sequences are planned, we need to determine which parts of the global landmass have been previously observed, so in the next schedule we can attempt to observe currently unobserved regions. This required adapting CLASP to be able to receive a previous schedule as input, and account for those observations in scheduling for the current planning horizon.

Time is divided into three regions with varying certainty: past (high certainty), current (moderate certainty) and future (low certainty). All three regions contribute to the score of a schedule against the science observation campaigns. We force a boundary condition that the data recorder is empty at the end of each planning period to simplify operations. We add the following component to our problem statement:

- **a planning horizon $(\mathbf{phst}, \mathbf{phet}) \subseteq (\mathbf{hst}, \mathbf{het})$ over which the schedule may be modified**

The schedule is a living document that is updated weekly during operations. Figure 6 shows the inputs and outputs of the scheduler for each week's run.

Ring Buffer Scheduling Constraint

The MSU onboard ECOSTRESS is a ring buffer. Ring buffers consist of two pointers - a read pointer ($r(t)$) and a write pointer ($w(t)$) (Knuth 1997). During downlink, the

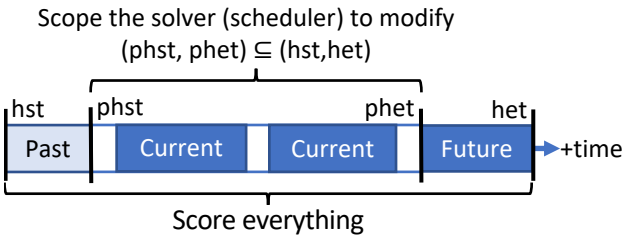


Figure 5: Planning horizon accounting for the past

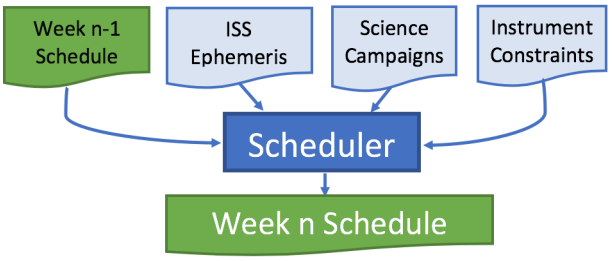


Figure 6: Scheduler now takes in previous week's schedule

data is read from the read pointer position, and the read pointer advances. New data is written to the write pointer position, and the write pointer advances. When functioning correctly, the read and write pointers move back to the start of memory when they reach the end of memory. An issue in the instrument firmware causes the read pointer to stay at the end of the memory rather than move to the start of memory as expected, continuously reading the same data from that position, even though new data may have been written at the start of memory.

The undesired condition resulting in data loss occurs when

$$w(t) < r(t) \tag{1}$$

indicating the write pointer has wrapped back around but the read pointer has not.

Rather than update the instrument firmware, which poses a higher risk, we opted to attempt a ground-based solution. A command can be issued that will reset the pointers back to the start. When scheduling the pointer reset times as well as the observations, we consider two constraints:

- **Constraint 1**: The amount of data acquired in between reset commands should not exceed the capacity of the buffer.
- **Constraint 2**: At the time of a reset command, the locations of the read pointer and the write pointer should be equal.

Constraint 1 prevents the write pointer from wrapping around the buffer, and Constraint 2 prevents any undownlinked data from being in the buffer at the time of a reset. If either constraint is not met, data will be lost.

Both constraints are specific to the ECOSTRESS mission and does not apply to the CLASP problem in general. Our scheduling goal then changes to:

- Our goal is to select $A \subseteq O$ to maximize U(r_i) $\forall r_i \in R$ subject to instrument constraints involving available memory, keepout zones, and Constraints 1 and 2.

We schedule in two passes, outlined in Algorithm 1. The first pass determines the ring buffer reset times, and the second pass returns the final schedule. In the initial pass, the scheduler is run with just the highest priority targets, with reset times at the end of each week. New schedules are uploaded weekly, so having the buffer empty at the end of each week allows for a more simple handover. When scheduling the observations, CLASP enforces the above constraints. We then examine the memory profile of the resulting schedule. We search forward through the memory profile until the point in time when the data has filled to some fraction of the buffer. This fraction is an estimate of the amount of memory going towards the high priority data, so there is enough memory still available to observe lower priority targets. Moving backwards from this point, we look for a time when the amount of memory onboard is lower than some threshold. If there is no memory onboard at a specific time, that means we are able to place a ring buffer reset there without sacrificing observing an high priority targets for that time period in the final schedule. The larger the amount of data scheduled to be in the buffer, the more observations will fail to be scheduled in the next run of CLASP. If a suitable point is not found, the threshold increases and the process repeats until a time for the reset is found. Then the search continues moving forward from the time chosen for the reset, and this repeats until we reach the end of the planning horizon. Figure 7 shows an example of reset times chosen after examining the memory profile.

Once all the reset times are found, the scheduler is run again with high and low priority targets to produce the final schedule, enforcing Constraints 1 and 2. Figure 8 shows the memory profile with data from high and low priority campaigns, and the buffer is empty at the reset times.

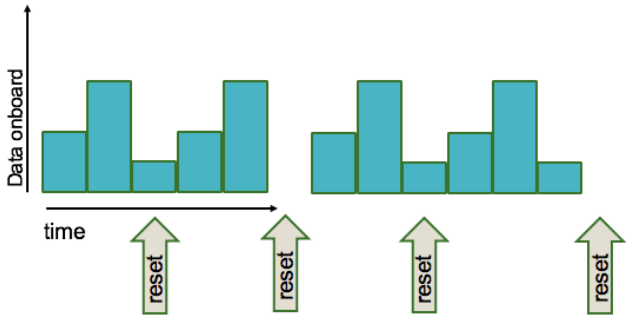


Figure 7: Scheduling resets after looking at memory profile with data from high priority campaigns (blue)

Uncertainty of Predicted Ephemeris

The ISS is in a region of orbit known as Low Earth Orbit (LEO). Objects in LEO experience drag from the atmosphere, which results in the ISS experiencing some drift

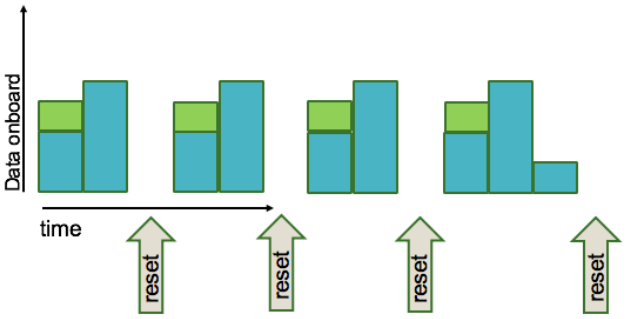


Figure 8: Memory profile of final schedule with memory from high priority campaigns (blue) and low priority campaigns (green), with no data in the buffer when resets occur

from its predicted location. This can cause an observation to be taken that misses the region it was intended to observe.

In the original version of CLASP, each observation has a start time ($o.start$), and a duration ($o.duration$). For ECOSTRESS, the duration is fixed at 52 seconds long. If a target is predicted to be viewed at any time between $o.start$ and $o.start$ + $o.duration$, that target is satisfied by that observation. If a target was predicted to be viewed near the start or the end or the observation window, that target may be missed operationally due to the uncertainty in the ISS position.

The initial solution to this problem was to spend extra time observing before and after each set of contiguous observations. Because the ECOSTRESS instrument takes data in scenes lasting 52 seconds, we add 26 seconds of observational time before and after, so each set of contiguous observations still has a duration that is a multiple of an ECOSTRESS scene. During scheduling, this extra time is accounted for when checking data volume constraints, but those times are not considered to satisfy any science targets. However, this extra time spent observing is wasteful and takes up data volume that could potentially be used by other productive observations.

A solution that could schedule observations such that no science targets would be missed due to drift, but would also allocate data volume efficiently, was warranted. Rather than choosing from 52 second observations to add to the schedule, we adapted CLASP to schedule from the second a target was predicted to be observed, and then build the observations from there accounting for some amount of uncertainty in the position and the fixed observation size.

The new method of scheduling observations is outlined in Algorithm 2. When a target is attempted to be scheduled that is visible at time t, we create an observation record that holds the start time (st), end time (et), as well as the latest start time (lst) and earliest end time (eet). These last two parameters are necessary when merging observations. We have two time-dependent functions p_b and p_a, which determine the amount of pad time necessary for a target visible at time t to ensure it is not missed. The latest start time and earliest end time will be $t - p_b(t)$ and $t + p_a(t)$ respectively. We consider three ways to determine st and et by shifting

```
procedure schedule()
    write resets at week ends
    run clasp with high priority campaigns
    last_reset_point = start_time
    while progress is made do
        last_reset_point =
          findNextReset(last_reset_point)
        write last_reset_point to file
    end
    run clasp with high and low priority campaigns
procedure findNextReset(lower_bound)
    upper_bound = find upper bound based on
      lower_bound
    while reset time not found do
        t = upper_bound
        while t > lower_bound do
            if memory at time t < threshold then
                return t
            else
                decrement t
            end
        end
        increment threshold
        t = upper_bound
    end
```

Algorithm 1: Algorithm for scheduling ring buffer resets

the observation forward or backwards. We choose the first shifting strategy, if any, that results in the observation being able to successfully be added to the schedule. We can center the observation, so that the amount of pad time on either side of lst and eet are equal. We can also make the observation as early or as late as possible, by adding all extra time before lst or after eet respectively.

Then we check to see if this observation is interfering with any previously scheduled observations. Interference could be a direct overlap in time spent observing, or it could violate the minimum length necessary between observations. If this observation does not interfere with any previously scheduled observations, and it does not violate any other constraints (memory, keepout times) it can be placed, and any targets observed during $(t, t+1)$ have one viewing requirement satisfied. If this observation does interfere with surrounding observations that, we merge this observation and the interfering one, and recursively merge until there are no interfering observations. In the merging algorithm, we check for interference between the newly created observation (x) and its immediate neighbors. For the preceding neighbor n_1, if there is interference, we create a new observation that has

$$lst \leftarrow min(x.lst, n_1.lst)$$

and

$$eet \leftarrow max(x.eet, n_1.eet)$$

This ensures that for any targets satisfied by O or N, the amount of pad time required on either side of them is maintained. Then we extend the observation to a multiple of a scene by setting st and et using the current shifting strategy. Figure 9 shows an example of created x' from merging x and n_1. We then recursively merge with this new observation (x'), and check for interference with the following neighbor n_2 and recursively merge once again if necessary. Once we obtain the newly merged observation, we can check if it violates any other constraints. If it does not violate any constraints, we can delete from the schedule any old observations that were merged to form this new one, and place the new one in the schedule. If it does, we consider the next shifting strategy for determining st and et, and return that the observation is not able to be placed once we consider all three strategies.

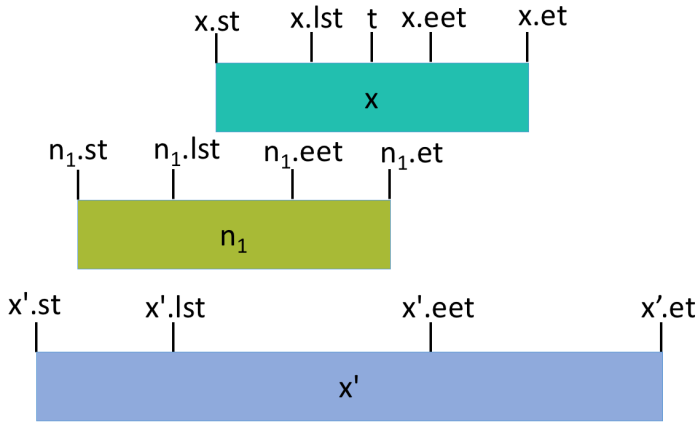


Figure 9: x' is the result of merging the newly created observation x with its preceding neighbor n_1

Validation

We validate the approaches previously presented with two experiments. The first experiment is an analysis of how well the algorithm for scheduling the ring buffer resets performs compared to a schedule produced discounting the issue. The second experiment is a comparison of the schedules produced by the two methods used to account for the uncertainty in the ephemeris.

The algorithm for scheduling the ring buffer resets is analyzed by comparing the schedule produced when accounting for the resets against a schedule produced when we only enforce the data recorder being empty at the end of each week. The goal with the algorithm is to avoid violating Constraints 1 and 2 while still taking as much high priority data as possible.

The change from adding one whole scene to each contiguous set of observations to building observations up from the second each target is observable will be validated by comparing the schedules produced by each method. We use a constant padding function that gives a pad time of 10 seconds on either side of each target. The schedules should have similar numbers of observations, since this value is limited by data constraints, but the resulting coverage should increase with the second method since the data availability is being used more effectively.

procedure `canPlaceObservation` (t)
  for *shift in shift strategies* **do**
    create observation x with $lst = t - p_b(t)$ and $eet = t + p_a(t)$, st and et according to $shift$
    x' = `mergeObservations` $(x,$ *shift*$)$
    temporarily delete any interfering observations merged to create x'
    if x' *does not violate any other constraints* **then**
      put back any deleted observations
      return True
    end
    put back any deleted observations
  end
  return False
procedure `mergeObservations` $(O,$ *shift*$)$
  if x *interferes with previous observation* n_1 **then**
    x' = merge x and n_1 together according to $shift$
    x = `mergeObservations` $(x', shift)$
  end
  if x *interferes with next observation* n_2 **then**
    x' = merge x and n_2 together according to $shift$
    x = `mergeObservations` $(x', shift)$
  end
  return x

Algorithm 2: Algorithm for checking if observations can be placed

Results

Ring Buffer Management

Figure 10 shows the coverage amount at each priority level for a schedule produced when considering the ring buffer constraint (orange), and one produced without considering the constraint (blue). When adding in this additional constraint, we achieve a level of coverage of high priority data that is very close to what we would achieve if this was not an issue.

The minimal impact on acquiring the high priority data is due to factors such as the locations of the high priority targets, the instrument data rate, and the downlink rate. There exists times in the schedule with only high priority targets when all of the data has been downlinked and the buffer is empty, allowing resets to be placed with no negative impacts. Had the downlink rate been slower, the instrument data rate been higher, or if there were more high priority targets, it is possible there would be no time when the buffer would be empty.

Uncertain Ephemeris

Figure 11 shows the difference in target coverage when using the method of building up observations from a smaller time, and Figure 12 shows the number of observations scheduled. For the six weeks tested, there was an average of 29.9% increase in coverage and a 3.75% decrease in observations scheduled when comparing the method of building

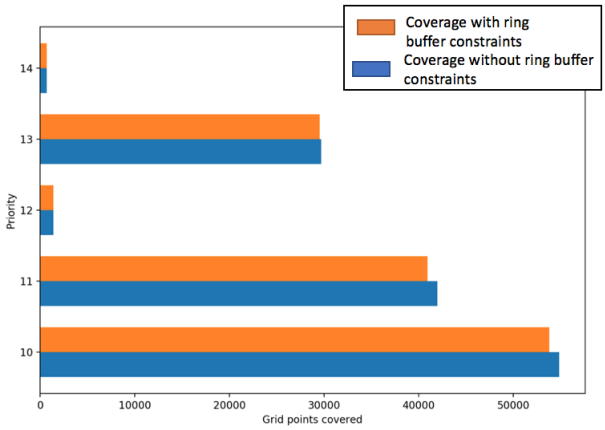


Figure 10: Plot showing gridpoints covered at each priority level by schedules produced when considering or not considering Constraints 1 and 2

scenes from a smaller time delta to adding a scene to each contiguous set of scenes. This shows there had been a significant amount of data volume being wasted with the original padding method. With the original method, the minimum acquisition length was two scenes. In the absolute worst case, if all targets were far enough apart that there were no contiguous scenes, the first method of padding would require double the scenes required by the second method of padding.

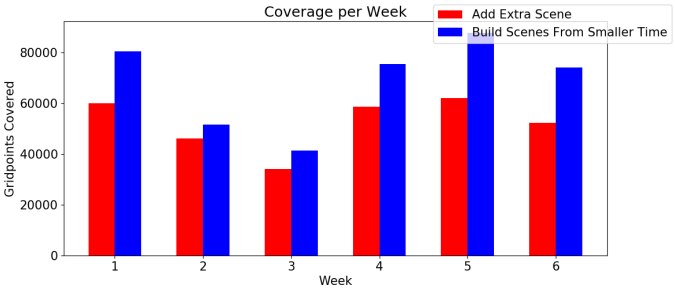


Figure 11: Plot showing coverage difference between the two padding methods

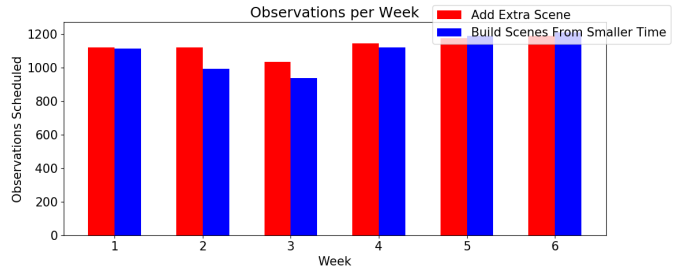


Figure 12: Plot showing number of observations scheduled with the two padding methods

Related Work

CLASP was previously used for on-orbit scheduling of the IPEX CubeSat (Chien et al. 2015), but IPEX did not require the scheduler to be aware of previously executed schedule or long-term observational campaigns. This paper describes extensions to CLASP that are aware of prior execution and long term observational campaigns.

CLASP has been used for long term mission studies for the upcoming Europa Clipper and JUICE missions (Troesch, Chien, and Ferguson 2017), as well as the NISAR mission (Doubleday and Knight 2014). The ARIEL mission study (Roussel et al. 2017) also focused on long term observation planning. The ARIEL and Europa Clipper studies assume perfect knowledge of future ephemeris and certain execution of scheduled observations, which is appropriate for early mission design analysis, but not mission operations. This paper focuses on the mission operations use case, where the schedule is continously updated to handle missed/unsatisfactory observations and changes in the observer's ephemeris.

CLASP was also used as a prototype for early stage mission planning of the THEMIS instrument on the Mars Odyssey spacecraft (Rabideau et al. 2010). The focus in the THEMIS study is performance of the squeaky wheel scheduling algorithm. This paper only considers a single pass of squeaky wheel when scheduling.

The receding horizon, sliding window scheduling approach has been implemented for Earth observational scheduling before (Lemaître et al. 2002; Aldinger et al. 2013; Lewellen et al. 2017). These three papers assumed perfect knowledge of vehicle state, perfect execution and focused on optimization and orientation path planning for agile spacecraft. ECOSTRESS is not agile and this paper does not explore optimization. This paper uses the sliding window scheme to address only imperfect state knowledge on longer timescales.

Future Work

Our decision to require the data recorder to be empty allowed for easier operations because it did not require an interface between the actual vehicle telemetry and the initial conditions of our data recorder fill state model. This simpler interface came at a cost – we prevent the scheduler from taking new science data near the end of each planning period so that the data recorder can drain. ECOSTRESS could produce more science data if we seeded the data recorder fill state with a predicted fill level based on the prior schedule or actual vehicle telemetry. Future missions should consider interfacing data recorder telemetry with the initial conditions of the scheduler's data recorder resource model.

A correctly scheduled and executed observation may be useless because of cloud cover at the time of observation. System malfunctions may also prevent the instrument from executing the scheduled observations. Both of these conditions require an observation to be rescheduled. Future work should handle the previous week's schedule carefully, removing activities that were not actually executed and preserving the resource consumption, but removing the the credit of unsatisfactory observations.

Currently a constant padding function is used when deciding the earliest end and latest start times. This value is an upper bound on the amount of drift there may be in a one week period. The drift may be time-dependent. The farther an observation from the creation of the ephemeris, the more likely the drift is larger. A better understanding of the drift may allow the padding functions to be truly time-dependent and allow for more observations to be scheduled.

Conclusion

This paper has described the use of an automated scheduling system in the analysis and operations for the ECOSTRESS mission. Changing orbital ephemeris and long-term campaign goals required adapting CLASP to consider past observations in scheduling for the future. The issue with the instrument ring buffer required scheduling with additional constraints, as well as scheduling another type of instrument activity. The uncertainty of the ISS orbital position required adapting how observations are scheduled. Through computational analysis we showed that our method for addressing the ring buffer approached the performance of schedules produced that did not have the added constraints, and that the second method of building observations up rather outperformed the method of adding a fixed amount of observational time to ensure no regions of interest were missed.

Acknowledgements

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

References

Acton, C. 1996. Ancillary data services of nasa's navigation and ancillary information facility. In *Planetary and Space Science*, volume 44, 65–70.

Aldinger, J.; Lohr, J.; Winker, S.; and Willich, G. 2013. Automated planning for earth observation spacecraft under attitude dynamical constrants. In *Jahrbuch der Deutschen Gesellschaft für Luft- und Raumfahrt*.

Chien, S.; Doubleday, J.; Thompson, D. R.; Wagstaff, K. L.; Bellardo, J.; Francis, C.; Eric Baumgarten, A. W.; Yee, E.; Stanton, E.; and Piug-Suar, J. 2015. Onboard autonomy on the intelligent payload experiment cubesat mission. volume 14, 307–315.

Doubleday, J., and Knight, R. 2014. Science mission planning for nisar (formerly desdyni) with clasp. In *SpaceOps 2014*.

Knight, R., and Chien, S. 2006. Producing large observation campaigns using compressed problem representations. In *International Workshop on Planning and Scheduling for Space (IWPSS-2006)*.

Knuth, D. E. 1997. The art of computer programming. volume 1, 244–245.

Lemaître, M.; Verfaillie, G.; Jouhaud, F.; Lachiver, J.-M.; and Bataille, N. 2002. Selecting and scheduling observa-

tions 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. 2017. A hybrid traveling salesman problem - squeaky wheel optimization planner for earth observational scheduling. In *International Workshop on Planning and Scheduling for Space (IWPSS 2017)*.

NASA. 2019. Ecostress https://ecostress.jpl.nasa.gov retrieved 2019-03-29.

Rabideau, G.; Chien, S.; Mclaren, D.; Knight, R.; Anwar, S.; Mehall, G.; and Christensen, P. 2010. A tool for scheduling themis observations. In *International Symposium on Space Artificial Intelligence, Robotics, and Automation for Space (ISAIRAS 2010)*.

Robinson, J. A. 2013. The sense in earth remote sensing from the international space station.

Roussel, S.; Pralet, C.; Jaubert, J.; Queyrel, J.; and Duong, B. 2017. Planning the observation of exoplanets: the ariel mission. In *International Workshop on Planning and Scheduling for Space (IWPSS 2017)*.

Troesch, M.; Chien, S.; and Ferguson, E. 2017. Using automated scheduling to assess coverage for europa clipper and jupiter icy moons explorer. In *International Workshop on Planning and Scheduling for Space (IWPSS 2017)*.

Autonomous Scheduling of Agile Spacecraft Constellations with Delay Tolerant Networking for Reactive Imaging

Sreeja Nag[1], Alan S. Li[1], Vinay Ravindra[1], Marc Sanchez Net[2], Kar-Ming Cheung[2], Rod Lammers[3], and **Brian Bledsoe[3]**

[1]NASA Ames Research Center, Bay Area Environmental Research Institute, CA, USA
[2]Jet Propulsion Laboratory, California Institute of Technology, CA, USA
[3]University of Georgia, Athens GA, USA
sreejanag@alum.mit.edu

Abstract

Small spacecraft now have precise attitude control systems available commercially, allowing them to slew in 3 degrees of freedom, and capture images within short notice. When combined with appropriate software, this agility can significantly increase response rate, revisit time and coverage. In prior work, we have demonstrated an algorithmic framework that combines orbital mechanics, attitude control and scheduling optimization to plan the time-varying, full-body orientation of agile, small spacecraft in a constellation. The proposed schedule optimization would run at the ground station autonomously, and the resultant schedules uplinked to the spacecraft for execution. The algorithm is generalizable over small steerable spacecraft, control capability, sensor specs, imaging requirements, and regions of interest. In this article, we modify the algorithm to run onboard small spacecraft, such that the constellation can make time-sensitive decisions to slew and capture images autonomously, without ground control. We have developed a communication module based on Delay/Disruption Tolerant Networking (DTN) for onboard data management and routing among the satellites, which will work in conjunction with the other modules to optimize the schedule of agile communication and steering. We then apply this preliminary framework on representative constellations to simulate targeted measurements of episodic precipitation events and subsequent urban floods. The command and control efficiency of our agile algorithm is compared to non-agile (11.3x improvement) and non-DTN (21% improvement) constellations.

Introduction

Response and revisit requirements for Earth Observation (EO) vary significantly by application, ranging from less than an hour to monitor disasters, to daily for meteorology, to weekly for land cover monitoring (Sandau, Roeser, and Valenzuela 2010). Geostationary satellites provide frequent revisits, but at the cost of coarse spatial resolution, extra launch costs and no polar access. Lower Earth Orbit satellites overcome these shortcomings, but need numbers and coordination to make up for response characteristics. Adding agility to satellites and autonomy to the constellation improves the revisit/response for the same number of satellites in given orbits. However, human operators are expected to scale linearly with constellation nodes (Eickhoff 2011) and operations staffing may be very costly.

Earth-Observing Constellation Autonomy

Scheduling algorithms for agile EO have been successfully developed for single large satellite missions, examples being ASPEN for EO-1, scheduling for the ASTER Radiometer on Terra, high resolution imagery from the IKONOS commercial satellite (Martin 2002), scheduling observations for the geostationary GEO-CAPE satellite (Frank, Do, and Tran 2016), scheduling image strips over Taiwan by ROCSAT-II (Lin et al. 2005), and step-and-stare approaches using matrix imagers (Shao et al. 2018). The Proba spacecraft demonstrated dynamic pointing for multi-angle imaging of specific ground spots that it is commanded to observe (Barnsley et al. 2004). Scheduling 3-DOF observations for large satellite constellations has been formulated for the PLEIADES project (Lemaître et al. 2002; Damiani, Verfaillie, and Charmeau 2005) and COSMO-SkyMed constellation of synthetic aperture radars (Bianchessi and Righini 2008). Scheduling simulations have demonstrated single Cubesat downlink to a network of ground stations within available storage, energy and time constraints. (Chien et al. 2019) has developed automated tasking for current sensors as a Sensor Web to monitor Thai floods.

Recent advances in small and agile satellite technology have spurred literature on scheduling fleet operations. Coordinated planners in simulation (Abramson et al. 2013; Robinson et al. 2017) can handle a continuous stream of image requests from users, by finding opportunities of collection and scheduling air or space assets. Cubesat constellation studies (Cahoy and Kennedy 2017) have successfully scheduled downlink for a fleet, aided by inter-sat communication. Evolutionary algorithms for single spacecraft (Xhafa et al. 2012), multiple payloads (Jian and Cheng 2008) and

satellite fleets (Globus et al. 2002) are very accurate but at large computational cost due to their sensitivity to initial condition dependence (genetic algorithms), exponential time to converge (simulated annealing) or large training sets (neural nets). Agile constellation scheduling with slew-time variations have shown reasonable convergence in the recent past using hierarchical division of assignment (He et al. 2019). However, algorithms have not been developed for onboard execution on real-time, fast-response EO applications and do not consider inter-sat comm scheduling in conjunction with imaging operations.

We have recently demonstrated (Nag, Li, and Merrick 2018) a ground-based, autonomous scheduling algorithm that optimizes spacecraft attitude control systems (ACS) to maximize collected images and/or imaging time, given a known constellation. The algorithm is now broadened in application scope by leveraging inter-satellite links and onboard processing of images for intelligent decision-making. Improving coordination among multiple spacecraft allows for faster response to changing environments, at the cost of increased scheduling complexity.

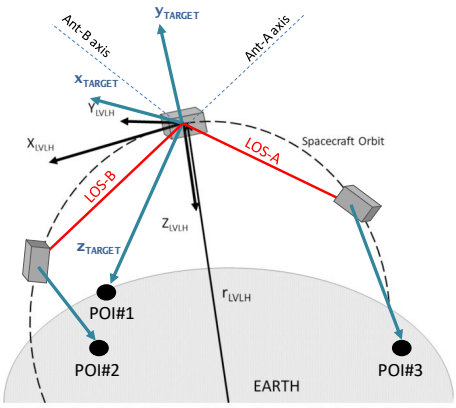


Figure 1—A constellation of satellites observing three points of interest (POI) by agile steering of their body frames, based on information shared when they have line of sight (LOS).

Networked Constellations and Reactive Science

DARPA's delay/disruption tolerant network or DTN paradigm (Cerf et al. 2007) is an emerging protocol standard for routing in the dynamic and intermittent operation environment. DTN makes it possible to minimize replication and improve the delivery probability within available resources, but has never been applied to EO inter-sat data exchange. We show that DTN enabled, agile constellations can respond to transient, episodic and/or extreme events using an autonomous scheduling algorithm that is executable onboard. The target scenarios are simulated by a simplified Observing System Simulation Experiment (OSSE) to evaluate the benefit of our proposed algorithm.

In the traditional sense, an OSSE is a data analysis experiment used to evaluate the impact of new observing systems, e.g. satellite instruments, on operational forecasts when actual observational data are not fully available (Arnold and Dey, 1986). An OSSE comprises of a free-running model used as the ground truth ('nature run'), used to compute the 'synthetic observations' for any observing systems, with added random errors representative of measurement uncertainty. Synthetic observations represent a small, noisy subset of the ground truth. They are used to forecast the full ground truth, then compared with the nature run. The disparity between the nature run of a chosen scenario, and the instrument-derived forecasts is then used to inform better instrument or mission design (Feldman et al. 2011). OSSEs can be used to train heuristics for mission planning, because different operational options can be assessed for different relevancy scenarios by changing the observing system characteristics and nature run appropriately (Nag, Gatebe, and Weck 2015; Nag et al. 2016).

Methodology

We propose a novel algorithmic framework that combines physical models of orbital mechanics (OM), attitude control systems (ACS), and inter-satellite links (ISL) and optimizes the schedule for any satellite in a constellation to observe a known set of ground regions with rapidly changing parameters and observation requirements. The proposed algorithm can run on the satellites, so that each can make observation decisions based on information available from all other satellites, with as low a latency as ISL allows. Satellites generate data bundles after executing scheduled observations to be broadcast by ISLs. Bundles contain information about the ground points observed and meta-data parameters pre-determined by the OSSE. Considering networking delays in a temporally varying disjoint graph (e.g. results in Figure 5) and diminishing returns for observing fast-changing environments, satellites are not expected to iterate on acknowledgments to establish explicit consensus. Instead, the more a satellite knows about a region before its observation opportunity, better its scheduler performance. The algorithm may also run on the ground, i.e. the satellites can downlink their observed data, the ground will run the proposed algorithms, and uplink the resultant schedule to the satellite. Since the ground stations are expected to be inter-connected on Earth and in sync with each other at all times, the optimization is centralized and the resultant schedule avoids potentially redundant observations due to lack of consensus among the satellites. This approach also reduces the onboard processing requirements on the satellites. However, since information relay occurs at only sat-ground contacts (function of orbits, ground network), the scheduler would use significantly information compared to the distributed, onboard bundles. The transiency of the environment being observed and its robustness to latency in exchanging inferences determines effectiveness of the onboard, decentralized vs.

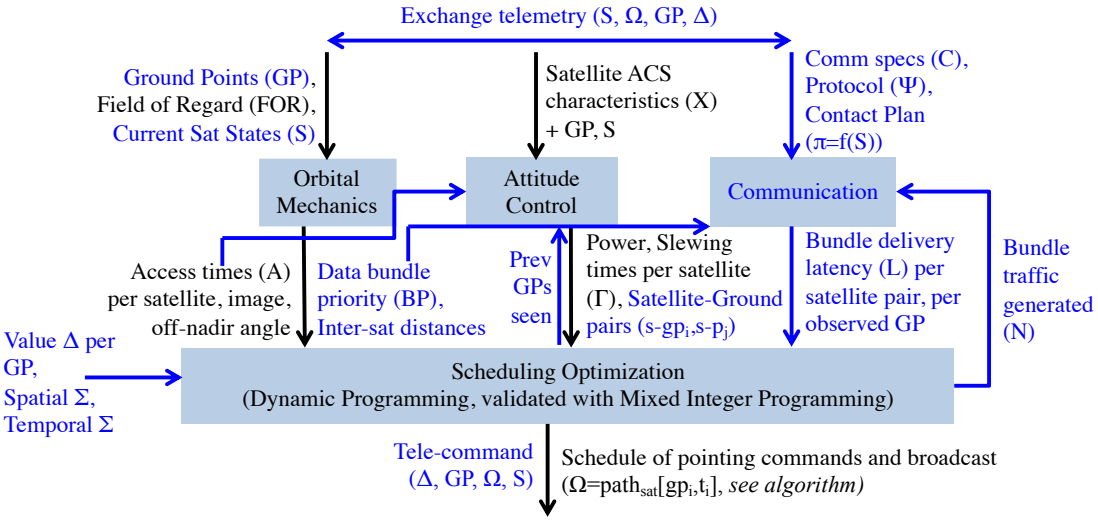


Figure 2 – Major information flows between the modules in the proposed agile EO scheduler, expected to run onboard every satellite in a given constellation, applied to global urban flooding in this paper. This framework can exchange information (as identified at the top) between the satellites via peer-to-peer communication or via the ground (reverse bent pipe architecture). The blue arrows/text represent newly added components to the previous version of the algorithmic framework (Nag, Li, and Merrick 2018).

ground, centralized implementation of our proposed algorithm (e.g. scenario results in Table 2). The algorithmic framework and information flow summarized in Figure 2.

The OM module propagates orbits in a known constellation and computes possible coverage of known regions of interest (appropriately discretized into ground points – GP) within a field of regard (FOR). It provides the propagated states and possible access times per satellite, per GP to the ACS. The ACS uses this information, with known satellite and subsystem characteristics to compute: time required by any satellite at a given time to slew from one GP to another (including satellite movement), resultant power, momentum and stabilization profiles. The OM also provides available line-of-sight (LOS) availability, corresponding inter-sat distances at any time, as well as the priority of bundle delivery to the Comm/ISL module so that it knows which satellites need to receive the data sooner. The comm. module computes the link budget for a known set of specifications and protocols, and uses the resultant data rate to simulate DTN and compute bundle drop rates and latency to deliver any known bundle between any given pair of satellites. Bundles exchanged between the satellites are modeled to contain seen GPs time series (Ω) or new GPs of interest, and their re-computed value (Δ), either in full, an update to the original, or as parametric meta-data. The mechanism of re-computing value at a GP (input on the left of Figure 1) is described on pg.4.

The optimization module ingests the outputs of the OM, ACS and Comm modules to compute the schedule of when each satellite should capture any GP. The executed schedule dictates the number of observations a satellite will make over any region, which dictates the number, size and timing of bundles generated for broadcast, therefore, we include a feedback loop between the optimizer and the comm. module. Slew characteristics depend on the previous GPs the satellite was observing and intended next, thus a feedback loop between the ACS and optimizer. If the constellation specifications, e.g. number of satellites, their arrangement, instruments, FOR, are expected to change over operational lifetime, a feedback between the OM and the optimizer may also be added. In the current implementation, we assume that the proposed real-time scheduling will take a fixed time interval and that other operations, e.g. downlink, calibration, maintenance, etc., will be scheduled separately.

Orbital Mechanics and Attitude Control

The main revision to the OM and ACS modules comprehensively described in (Nag, Li, and Merrick 2018) is that we now compute slewing time as a function of a pair of $sat_{t,s}$-gp_i , each representing a vector from satellite s at time t to ground point i. The dynamic programming (DP) algorithm in the optimizer now uses observable GPs as potential states, instead of discrete pointing options. Since the DP scheduler iteratively calls ACS for any pair of vectors, the ACS slew time cannot be pulled from a static table using the starting and ending satellite pointing direction as before, and are computed real-time. Onboard processing constraints limit our use of the full physics-based ACS simulator (developed on MATLAB Simulink), therefore we developed weighted least squares on a third order polynomial whose coefficients are a function of the satellite mass, ACS and other specifications that served as knobs in the original model. The polynomial provides a very efficient implementation of the

ACS-DP feedback loop in Figure 2, by allowing fast computation of slew time as a function of α, the angle between any pair of vectors $sat_{t,s}$-gp_i. This process is adaptable to non-planar angle dependencies as well, in case a full body re-orientation of the satellite is necessary, not just re-pointing an instrument.

The OM module has been developed in house, leveraging NASA GSFC's open-source General Mission Analysis Tool (Hughes 2007). The OM also generates data bundle priority, to be ingested by the comm. module as follows: The data-bundle is tagged with the corresponding region or point of observation (where the data is generated). Priority is given to the next satellite to be able to access the same region or point, after the bundle generation, so that when it reaches the said region, it is up-to-date about it, as inferred by the last observing satellite. For example, if Sat11 generates data over "Dallas", and Sat12 is the next satellite on scene, Sat12 is given highest priority for the data-bundle to be delivered. If some satellites do not ever visit "Dallas", they are removed from the recipient queue.

Delay/Disruption Tolerant Networking

Delay/Disruption Tolerant Networking (DTN) is a new set of communication protocols developed with the intent of extending the Internet to users that experience long delays and/or unexpected disruptions in their link service. At its core, DTN defines an end-to-end overlay layer, termed "bundle layer", that sits on top of the transport layer (i.e., TCP or UDP in the Internet, CCSDS link layer standards in space) and efficiently bridges networks that may experience different types of operational environments. To achieve that, it encapsulates underlying data units (e.g., TCP/UDP datagrams) in bundles that are then transmitted from bundle agent to bundle agent, who store them persistently for safekeeping until the next network hop can be provisioned. This hop-to-hop philosophy is at the core of DTN and differentiates it from the Internet, where transactions typically occur by establishing end-to-end sessions (between the data originator and data sink).

At present, DTN is comprised of a large suite of specifications that encompass all aspects of network engineering, including its core protocols (e.g., the Bundle Protocol (Scott and Burleigh 2007), the Licklider Transmission Protocol (Farrell, Ramadas, and Burleigh 2008), or the Schedule Aware Bundle Routing (Standard and Book 2018)), adapters for bridging different types of underlying networks, network security protocols and network management functionality (Asynchronous Management Protocol). For the purposes of this paper, however, only the parts of the Bundle Protocol and Schedule Aware Bundle Routing Protocol were implemented. Together, they provide a medium to high fidelity estimate of how bundles would move in a DTN consisting of near-Earth orbiting spacecraft and allow us to quantify network figures of merit such as bundle loss or average bundle latency. Our DTN model is implemented in Python using Simpy (Matloff 2008), a discrete-event engine built upon the concept of coroutines (or asynchronouos functions in the latest Python versions).

Quantifying Value: Observing System Simulations

We apply our proposed framework to episodic precipitation and resultant urban floods, to demonstrate its utility and scalability. We used the Dartmouth Flood Observatory (Brakenridge 2012) to study the frequency of global floods in 1985 – 2010, and identified 42 large cities that are within floodprone areas and marked a 100 km radius buffer around them to define the watersheds. We assume a 6 hour planning horizon, during which 5 of the 42 cities (Dhaka, Sydney, Dallas, London, Rio de Janeiro) flood to varying degrees as modeled by an OSSE nature run. For example, London tends to get slower, longer rains that might cause the Thames to flood, Dallas is more concerned with short, intense thunderstorms causing flooding on smaller creeks. The OSSE developed for this paper uses an area of 80km x 80km, and is currently agnostic to flood-type disparities between cities.

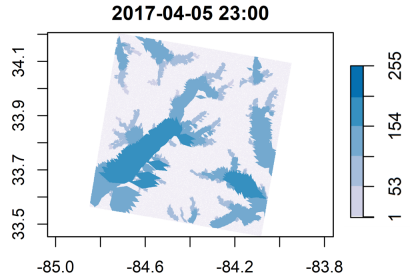


Figure 3 – Example of the spatial distribution of value (8-bit scale) of an ~80 km square region around Atlanta, GA (X/Y axis in degrees), for a single snapshot in time.

To quantify value for the ***optimizer's objective function***, we modeled riverine flooding in the Atlanta metropolitan area for a single storm event from April 5-6, 2017, using the WRF-Hydro hydrologic model version 5 (Gochis et al. 2018). The model was run with a grid resolution of 900 m, and was calibrated by adjusting parameters until modeled streamflow matched measured flow at nine U.S. Geologic Survey gages in the Atlanta metro area. The modeled channel flow rates were then normalized by the 2-year recurrence interval flow rate (Q2), as estimated from the USGS regional regression equations for urban streams in that region (Feaster, Gotvald, and Weaver 2014). Q2 is the flow that has a 50% chance of happening in any year, and is an estimate of what constitutes a "flood" at any location. Finally, these normalized flood rates were then transformed on a log-scale to integer values between 1 and 256. We estimated the

watershed land area draining to each channel point and the [1,256] value of that point was assigned to the entire watershed area. Areas with high value correspond to watersheds with active flooding. In these watersheds, it is important to obtain satellite-derived estimates of precipitation to determine if this flooding will worsen (with more rainfall) or abate. This process provides an expected value of observing every point in a region of interest at 15 min resolutions, to be used by the satellite scheduler, *absval*($[gp_x,t_y]$) in the next section. One snapshot is shown in Figure 3.

This paper uses the following statistical model for value re-computation. Since the OSSE time resolution is 15min, the cumulative value of observing any GP within 15min should be constant, i.e. if it has been seen once, subsequent observations within 15min are of zero value. Since value re-computation based on collected data is not physically simulated, we estimate it from OSSE output (*absval*) in the following ways: The value of observing any GP after 15min is considered a fraction (=*1/number_of_times_seen*) of its OSSE-provided value with some random noise added. This re-computation ensures that a diversity of GPs is observed over time. Similarly, the value of observing a GP can be inversely proportionate to its distance to already observed GPs, to maximize the spatial spread of information collected and characterize the region better. The time-stepping nature of our algorithm causes it to be agnostic to the future value of any GP, therefore it can ingest changing values as they come along and compute schedules accordingly. We applied a standard normal distribution with a 2%-8% (uniformly random) standard deviation to the OSSE-provided values, to generate slightly different value functions to be used by each satellite's optimizer. This was to simulate different inferences by satellites after onboard processing their different observations, owing to different schedules. If they observed the same GPs at the same time, they would have the same inference, but that is obviously not possible. We are developing a high fidelity, value re-computation model to replace this statistical model, whose onboard processing algorithms and predictive technology will be described in a future publication. It will simulate processing data collected from executed observing schedules, updating future value, re-computing schedule and passing along insights the other sats.

Dynamic Programming based Scheduler

Our proposed optimization algorithm (Table 1) uses dynamic programming (DP) to greedily optimize the scheduler. Each satellite is theorized to possess its own DP scheduler on board (or its own thread on ground) - a cartoon version is in Figure 4, where the gradient of the nodes represents varying absval($[gp_x,t_y]$), $\forall x \in [1,numGP],\ y \in [1,horiz\_tSteps]$, as obtained from the OSSE. The scheduler outputs a vector of tuples $[gp_i,t_i]\ \forall i \in [1,pathLength]$, which is the schedule for sat to observe gp_i at t_i. Compared to our previous implementation, the state space that the optimized path has to trace is now a graph of time steps and GPs, instead of time steps and satellite pointing directions. At any time *tPlan* during mission operations, a schedule can be computed for a future *planning_horizon*. The scheduler (line 4) processes bundles received until *tPlan* through the DTN and updates its knowledge of all other sats *c* as broadcasted at $tSrc_c$, i.e. $path_c[gp_i,t_i \leq tSrc_c]$, and its insights of the regions, i.e. $modelParams(t_i \leq tSrc_c)$. For every time step *tNow*, it then steps through the GPs *gpNow* within the sat's FOR, and computes the cumulative value *val([gpNow,tNow])* of each path ending at *[gpNow,tNow]*, e.g. in Figure 4. Via DP, line 11's computation entails adding *val([gpNow,tNow])* to *val([gpBef,tBef])* for all possible nodes *[gpBef,tBef]* $\forall$ *gpBef* $\in$*[1,numGP], tBef* $\in$*[tNow-max(slewTime), tNow-min(slewTime)]*. *slewTime* is the full y-space of Eq.(2) for representative set of reorientations in the current mission scenario. The nodes between the red horizontal lines in Figure 4 are examples of *[gpBef,tBef];* searching only a practical portion of the space (e.g., *t-2* through *t-9*) mitigates some of the computational load that has been added due to the dynamic slew computation, instead of the previous static, slew time table. Note that *val* is re-computed using *absval* and the satellite's knowledge of the executed observations by the rest of the constellation and their insights. Cumulative value is computed statistically as:

$$computeValue() = \sum_{x=1}^{numGP} \sum_{y=1}^{horiz\_tSteps} val([gp_x, t_y]) \quad (1)$$

A future scheduler will implement a higher fidelity cross-correlation function which extends the current OSSE. If the scheduling sat's FOR at *tNow* overlaps with any other's FOR (line 9), it must *computeValue()* for all possible paths by

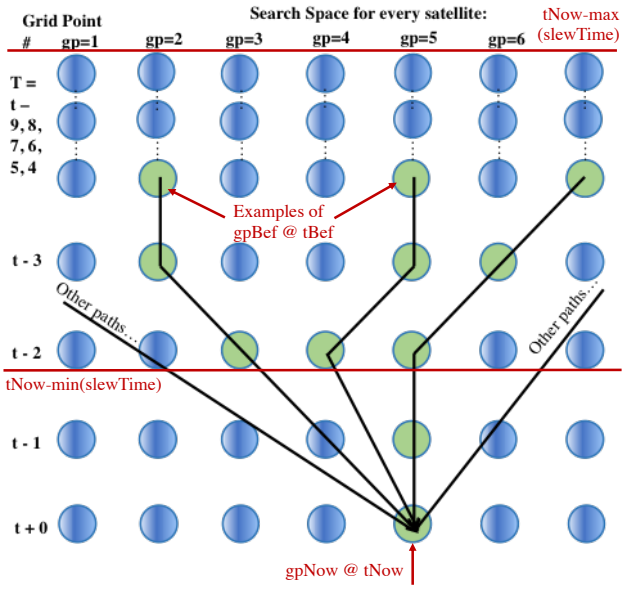


Figure 4—State space searched by the scheduler to compute the optimum path ending at *[gpNow,tNow]*, with no FOR overlaps

the other *s,* and maintain cumulative value numbers for possible paths by every permutation of sats in set *satsWoverlappingFOR* (line 13). Starting with the paths with maximum cumulative value, slew time of the last leg *[gpBef,tBef]→[gpNow,tNow]* (or combination of legs for overlapping sats) is dynamically computed. For the first instance where the time required is shorter than allowed between the *tBef→tNow* gap, the path is stored as the optimum path $path_{sat}[gpNow,tNow]$, and all other paths ending at *[gpNow,tNow]* discarded (to prevent memory becoming astronomical). Future implementations may explore ways to preserve some dominated paths since potentially optimum solutions, although tied with sub-optimum ones at *tNow*, are lost in the process.

Table 1—Summarized Scheduling Algorithm

```
1: Inputs – sat, absval([gp_x,t_y])
2: Output – path_sat[gp_i,t_i]
3: For c in Constellation-{sat} do
4:    modelParams(t_i≤tSrc_c), path_c[gp_i,t_i≤tSrc_c] ←
        DTN(c,tSrc_c,sat,tPlan)
5: End For
6: For tNow in planning_horizon do
7:    For gpNow in GroundPntsInFOR(sat, tNow) do
8:       GroundPntsInBND=[tNow-max(slewTime):
                          tNow-min(slewTime),1:numGP]
9:       For s in satsWoverlappingFOR(sat,gpNow) do
10:         For [gpBef,tBef] in GroundPntsInBND do
11:            v[s]=computeValue(path_s[gpBef_s,tBef_s]+
                  [gpNow,tNow],absval, path_c∈Const
                  [gp_i,t_i≤tSrc_c], modelParams(t_i≤tSrc_c))
12:         End For
13:         v_combi = v[permute(s in satsWoverlappingFOR)]
14:         For vn in reverse_sort(v_combi)
15:            tslew=computeManueverTimes(s_combi,
                  [gpBef_s_combi,tBef_s_combi], [gpNow,tNow])
16:            If tslew≤[tNow-tBef_s_combi] then
17:               path_s[gpNow,tNow]←
                        path_s[gpBef_s,tBef_s]+[gpNow,tNow]
18:               break // forLoop for vn
19:            End If
20:         End For
21:      End For
22:   End for
23: End for
```

The advantages of this algorithm are:

1. Runtime is linearly proportionate to the number of time steps in planning horizon n(T). The scheduler may be run for only the duration of FOR access over a region. It needs to be rerun only if value of the GPs (expected to be accessed) changes, as inferred or informed.
2. Since the scheduler steps through the planning horizon, it can be stopped at any given point in the runtime, and the resultant schedule is complete until that time step. Schedules can thus be executed as future schedules are being computed by the onboard processor.
3. A complex value function with non-linear dependencies (e.g. on viewing geometry or solar time) or multi-system interactions (e.g. Simulink or proprietary software calls) are easy to incorporate.
4. Algorithmic complexity per satellite per region to be scheduled is $O(n(T)\times n(GP)^{2\times n(S)})$, where n(GP) is the number of ground points within FOR and n(S) is the number of satellites *that can access the same GPs at the same time*, i.e. GPs within FOR overlaps. Since GPs are typically designed to Nyquist sample the footprint, runtimes are instrument dependent. If satellite FORs are non-overlapping, runtime or space complexity does not depend on the size of the constellation. For well-spread constellations observing non-polar targets, n(S) = 1, or a couple.

Integer programming (IP) was able to verify that optimality of the above algorithm for single satellites was within 10%, and find up to 5% more optimal solutions (Nag, Li, and Merrick 2018). The DP solution was 22% lower than the IP optimality bounds for constellations, which is well within the optimality bounds of greedy scheduling for unconstrained submodular functions (Piacentini, Bernardini, and Beck 2019). The DP schedules were found at nearly four orders of magnitude faster than IP, therefore far more suited for real time implementations. Currently, the scheduler is (re)run at the same frequency as DTN-informed value (re)processing, however future implementations will explore methods to decouple them, because rapider value updates but longer planning horizon are better for solution quality.

Results

We simulate a case study of 24 (20 kg cubic) satellites in a 3-plane Walker constellation observing floods in 5 global regions over a 6-hour planning horizon. All satellites are simulated at a 710 km altitude, 98.5 deg inclination, circular orbits similar to Landsat. The constellation is a homogeneous Walker Star-type, with 3 orbital planes of 8 satellites each. While the gap between satellite accesses to a region is ~10 mins when there is an orbital plane overhead, a minimum of 3 planes is needed for the maximum gap to be within 4.5 hours (median gap ~ 1hr). Two planes would not be able to appropriately respond to a 6-hr flood phenomenon even with agile pointing, crosslinks onboard autonomy. For the chosen altitude, at least 8 satellites per plane ensures consistent in-plane LOS (cross-plane LOS in polar regions only), therefore the 24-sat topology is the minimum nodes for continuous DTN to enable <6-hr urban flood monitoring.

Instruments potentially used for precipitation and soil moisture sensing are narrow field of view (FOV) radars, which justify the need to continuously re-orient the <10 km footprint to cover a large flooding area. Examples are the Ka-band radar on a cubesat called RainCube (Peral et al. 2015) for precipitation, L-band bistatic radar on CYGNSS, and a Cubesat P-band radar (Vega Cartagena et al. 2018) for soil moisture. This paper presents results for an 8km footprint

instrument. The field of regard (FOR) which limits the maximum off-nadir angle of the payload/instrument is set to 55 deg, because it corresponds to 5x distortion of the nadir ground resolution, which is the OSSE's limit to allow combining observations in a given region. Spatial resolution dependence of value can also be included in the objective function. The presented scenario will be varied in terms of the mission epoch and regions of interest since it affects access intervals, observations and bundle traffic, and performance sensitivity reported in a future publication.

The ACS model, characterized with the satellite specs from (Nag, Li, and Merrick 2018), is fitted by the following polynomial, where t is time for maneuver, and α is angle to span. The standard deviation is around 0.2116, so add 0.4232 to get ~95% percentile.

$$t = 6.1974 \times 10^{-6} \times \alpha^3 + 1.3904 \times 10^{-3} \times \alpha^2 + 1.4165 \times 10^{-1} \times \alpha + 4.6231 \quad (2)$$

While we present results based on full body re-orientations of a small satellite, our proposed algorithm can support constraints from the gimbaled re-orientation of payloads for fixed, larger satellites, by replacing the *tslew* computation model in Table 1 line 15.

Performance of Inter-Satellite Networking

To estimate the performance of the DTN protocol stack, we first evaluated the supportable data rate in the inter-satellite links between spacecraft in the constellation. We make the following assumptions: All spacecraft transmit at S-band within the 6MHz typically available to class A missions; the link distance is set to 6000km (from the OM module; we use the worst case for the inter-plane links since their distances are variable); they are equipped with an SSPA that can deliver up to 5W of RF power, and a dipole placed parallel to the nadir/zenith direction (typical for small sats). This design ensures minimal complexity since the SSPA can be directly connected to the antenna without needing splitters. Since the orientation of the spacecraft at any point in time is highly variable, we close the link budget assuming that both the transmitting and receiving antennas operate at the edge of the -3dB beamwidth. We consider that no atmospheric effects impair the links, and we select a ½ LDPC coding scheme together with a BPSK modulation, SRRC pulse shaping and NRZ baseband encoding. Using these inputs, we pessimistically estimate the link performance at 1kbps. Since multiple spacecraft can be in view of each other at any point in time (especially over the poles), and they carry omnidirectional antennas, there is potential for interference. For the physical layer, we assume that signal interference is mitigated using some form of multiple access scheme (e.g. Frequency or Code Division Multiplexing) – the reported 1kbps data rate must be interpreted as that presented by the multiple access scheme to the upper layers of the protocol stack. Interference can also affect DTN's routing layer.

To route data through the time-varying topology of the 24 satellite constellation, we simulate the system assuming that each of them is a DTN-enabled node with a simplified version of the Bundle Protocol and the Schedule Aware Bundle Routing Protocol. The DTN simulation uses the following inputs: The OM-provided contact plan (opportunities between any satellite pair in the network) is the basis for all routing decisions, and is specified as a six element tuple: Start time, end time, origin, destination, average data rate, range in light seconds. Second, the traffic generated in the constellation, provided by the optimizer as a function of average collections, indicating when bundles are created, who sources them, who they are destined for, and the OM-provided relative priority flag with 14 levels. Bundles of size of 2000 bits (1645 bits of observational inference data plus 20% of overhead due to the protocol stack) are generated and broadcasted for every GP observation, and communications are assumed error-less at 1kbps. The priority levels are also used to set the Time-To-Live (TTL) property of all bundles such that: Priority 1 has a 15min TTL, priorities 2 and 3 have a 30min TTL, and priorities 4 to 15 have a 50min TTL. These rules let the network automatically discard stale information and minimize traffic congestion.

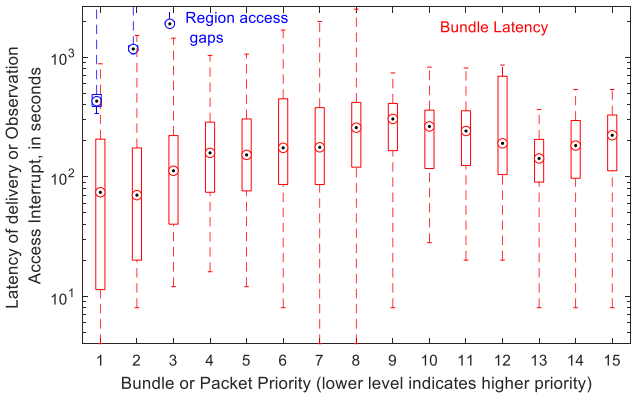


Figure 5 – Latency of data bundle delivery over all satellite pairs compared to the gaps between satellite FOR access to any region. For any satellite pair of given priority of DTN comm, if longest latency is less than shortest gap, each satellite can be considered fully updated with information the other, i.e. perfect consensus in spite of distributed scheduling on a disjoint graph. Each box represents 25%-75% quartiles, circle is median, whiskers show max/min

End-to-end latency experienced by 8341 bundles generated and sent over a 6 hour simulation (<1min DTN runtime) is shown in Figure 5. This latency is computed on a bundle-per-bundle basis, and measures the absolute time difference between the instant a bundle is delivered to the destination's endpoint (akin to TCP port), and the time it was originally created. Assuming a perfect multiple access scheme, any

spacecraft might receive a copy of a bundle that was not originally intended for it, causing the problem of packet duplication in the system due to physical interference. If not dealt with, these extra copies would be re-routed and create exponential replication problem that would overwhelm the entire system. To mitigate this, we take advantage of the extension blocks defined in the Bundle Protocol (Scott and Burleigh 2007). Particularly, every time router decides the next hop for a bundle, it appends an extension block with the identifier of the intended next hop. If another spacecraft receives a copy inadvertently, the router simply discards it.

Results indicate that latency is indeed affected by the bundle prioritization, however the effect is not monotonic because prioritization only happens at the bundle layer (e.g., radios have queues of frames, but they do not know about priorities in upper layers). Bundles with priorities 1-6 typically experience latencies of ~10s, with few outliers up to 15 minutes. This is quick enough for *most* of any satellite's knowledge to be transferred to the *next two* approaching any region (Figure 5). Also, no high priority bundles were dropped due to TTL expiration. Bundle with lower priorities experience larger latencies of ~2 minutes on average. The time to reach a region by satellites with priority≥3 is long enough for all bundles to be delivered, therefore all generated schedules by individual satellites have implicit consensus (because they use the same inputs). Access gaps for satellites with priority≥4 is out of Figure 5's Y-axis range. Latency was found to deteriorate non-linearly with increasing number satellites, bundle size due to more model parameters, and bundle traffic due to more observations. Future implementations will model bundle interference, trade-offs between omni vs. directional antennas, variations in bundle size and broadcast frequency, and their impact on latency.

Performance of the Imaging Scheduler

The DP-based optimizer ingests outputs from all modules to find the best observational path that maximizes cumulative value till any given time, per satellite. We compare results from the use case in running the proposed algorithmic framework in 2 scenarios. *One,* the scheduler runs on onboard and uses collected information from other satellites as they come through the DTN every 10 minutes. Lowering (increasing) this re-scheduling frequency based on onboard power or processing constraints will improve (lower) the quality of results. *Two,* the scheduler runs on the ground and uses collected information from other satellites as they downlink. The ground stations are placed near both poles to emulate an optimistic scenario of ground contacts (thus, value update and rescheduling) twice an orbit i.e. ~30 per day. Lowering the contact frequency will lower the quality of results, e.g. current Cubesat missions commit to 2 contacts per day at NASA and 4-5 per day commercially.

The onboard run uses updated value of any GP, based on all bundles about that GP that have arrived (executed schedule and inference data from others ingested in Table 1 line 5). Since the scheduler is distributed and runs per satellite, it risks knowing everything or nothing about any GP, based on DTN's relay from other satellites. Our implementation shows that the constellation predicts GP value at an average of 4% different from their actual value, due to bundles about GPs arriving later than the satellite already observes them. This happens only for some outliers in the one or two hop connections (Figure 5), thus >95% of the GPs in all 5 regions are observed. Longer the DTN latency, more the difference between the assumed ("what it thinks it's seeing") and recorded value ("what it's actually seeing") of fast-changing phenomena, lower the cumulative value.

Table 2—Comparison of optimizers run Onboard vs. Ground. A constellation with no agility sees 8.4% of the GPs, in either case)

	Scenario#1 (Distributed)	**Scenario#2 (Centralized)**
Cumulative Value (6h)	26347	21820
% of all GP observed	95.2%	99.2%

The centralized run has no risk of overlapping observations because all sats "know" every other's schedule, allowing for >99% of GPs seen. However, value functions are based on information obtained approximately an orbit earlier, due to collection-uplink-reschedule-downlink latency between any satellite pair. Our implementation shows that the constellation assumes GP value at an average of 70% different from recorded value, due to lack to timely communication of value updates. While the exact difference is a function of sensitivity of value updates to schedules executed by different sats (currently fractional decay with observation, 2%-8% variation in inference), it shows that in fast changing environments, a responsive constellation's performance is better captured by OSSE-driven metrics beyond simple coverage.

In the presented case study, the DTN-enabled decentralized solution provides 21% more value over 6 hours than the centralized implementation of the same algorithm. If we lower the transiency of the phenomena to an hour (currently 15 mins for precipitation) i.e. time resolution of OSSE-outputs; or if we focus on the poles (currently mid-latitude floods) where there is more FOR overlap i.e. increased processing complexity, and ISL interference i.e. more DTN latency, the centralized solution may provide more value. The proposed scheduler may be evaluated for a given user scenario, and run either way or as a combination.

The time taken to run the algorithm per sat was 1% of the planning horizon, evaluated on MATLAB installed in a Mac OS X v10.13.6 with a 2.6 GHz processor and 16 GB of 2400 MHz memory.

Acknowledgements

Funded by the NASA New Investigator Program, Earth Science Technology Office, and the Interplanetary Network Directorate at the Jet Propulsion Laboratory.

References

Abramson, Mark R., Stephan Kolitz, Eric Robinson, and Dorri Poppe. 2013. "Earth Phenomena Observation System (EPOS) for Coordination of Asynchronous Sensor Webs." In *AIAA Infotech@ Aerospace (I@ A) Conference*, 4813. https://arc.aiaa.org/doi/abs/10.2514/6.2013-4813.

Arnold Jr, Charles P., and Clifford H. Dey. 1986. "Observing-Systems Simulation Experiments: Past, Present, and Future." *Bulletin of the American Meteorological Society* 67 (6): 687–695.

Barnsley, M. J., J. J. Settle, M. A. Cutter, D. R. Lobb, and F. Teston. 2004. "The PROBA/CHRIS Mission: A Low-Cost Smallsat for Hyperspectral Multiangle Observations of the Earth Surface and Atmosphere." *Geoscience and Remote Sensing, IEEE Transactions On* 42 (7): 1512–1520.

Bianchessi, Nicola, and Giovanni Righini. 2008. "Planning and Scheduling Algorithms for the COSMO-SkyMed Constellation." *Aerospace Science and Technology* 12 (7): 535–544.

Brakenridge, G. R. 2012. *Global Active Archive of Large Flood Events, Dartmouth Flood Observatory, University of Colorado*.

Cahoy, Kerri, and Andrew K. Kennedy. 2017. "Initial Results from ACCESS: An Autonomous CubeSat Constellation Scheduling System for Earth Observation." In . Logan, Utah. http://digitalcommons.usu.edu/smallsat/2017/all2017/98/.

Cerf, Vinton, Scott Burleigh, Adrian Hooke, Leigh Torgerson, Robert Durst, Keith Scott, Kevin Fall, and Howard Weiss. 2007. "Delay-Tolerant Networking Architecture."

Chien, Steve, David Mclaren, Joshua Doubleday, Daniel Tran, Veerachai Tanpipat, and Royol Chitradon. 2019. "Using Taskable Remote Sensing in a Sensor Web for Thailand Flood Monitoring." *Journal of Aerospace Information Systems* 16 (3): 107–119.

Damiani, Sylvain, Gérard Verfaillie, and Marie-Claire Charmeau. 2005. "An Earth Watching Satellite Constellation: How to Manage a Team of Watching Agents with Limited Communications." In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, 455–462. ACM. http://dl.acm.org/citation.cfm?id=1082543.

Eickhoff, Jens. 2011. *Onboard Computers, Onboard Software and Satellite Operations: An Introduction*. Springer Science & Business Media. https://books.google.com/books?hl=en&lr=&id=JUYo2E9UB18C&oi=fnd&pg=PP2&dq=jens+eickhoff+onboard&ots=pzlK4F7Qt1&sig=wlngMSJIL3uI_tEeCjc3ndNHzmQ.

Farrell, Stephen, Manikantan Ramadas, and Scott Burleigh. 2008. "Licklider Transmission Protocol-Security Extensions."

Feaster, Toby D., Anthony J. Gotvald, and J. Curtis Weaver. 2014. "Methods for Estimating the Magnitude and Frequency of Floods for Urban and Small, Rural Streams in Georgia, South Carolina, and North Carolina, 2011." *US Geological Survey, SIR* 5030.

Feldman, Daniel R., Chris A. Algieri, Jonathan R. Ong, and William D. Collins. 2011. "CLARREO Shortwave Observing System Simulation Experiments of the Twenty-First Century: Simulator Design and Implementation." *Journal of Geophysical Research: Atmospheres (1984–2012)* 116 (D10). http://onlinelibrary.wiley.com/doi/10.1029/2010JD015350/full.

Frank, Jeremy, Minh Do, and Tony T. Tran. 2016. "Scheduling Ocean Color Observations for a GEO-Stationary Satellite." In *Proceedings of the Twenty-Sixth International Conference on International Conference on Automated Planning and Scheduling*, 376–384. AAAI Press. http://dl.acm.org/citation.cfm?id=3038642.

Globus, Al, James Crawford, Jason Lohn, and Robert Morris. 2002. "Scheduling Earth Observing Fleets Using Evolutionary Algorithms: Problem Description and Approach." https://ntrs.nasa.gov/search.jsp?R=20020091594.

Gochis, D.J., Michael Barlage, A. Dugger, K. Fitzgerald, L. Karston, M. McAllister, J. McCreight, et al. 2018. "The WRF-Hydro Modeling System Technical Description, (Version 5.0)." NCAR Technical Note DOI:10.5065/D6J38RBJ.

He, Lei, Xiao-Lu Liu, Ying-Wu Chen, Li-Ning Xing, and Ke Liu. 2019. "Hierarchical Scheduling for Real-Time Agile Satellite Task Scheduling in a Dynamic Environment." *Advances in Space Research* 63 (2): 897–912. https://doi.org/10.1016/j.asr.2018.10.007.

Hughes, Steven P. 2007. "General Mission Analysis Tool (GMAT)." http://ntrs.nasa.gov/search.jsp?R=20080045879.

Jian, Li, and Wang Cheng. 2008. "Resource Planning and Scheduling of Payload for Satellite with Genetic Particles Swarm Optimization." In *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress On*, 199–203. IEEE. http://ieeexplore.ieee.org/abstract/document/4630799/.

Lemaître, Michel, Gérard Verfaillie, Frank Jouhaud, Jean-Michel Lachiver, and Nicolas Bataille. 2002. "Selecting and Scheduling Observations of Agile Satellites." *Aerospace Science and Technology* 6 (5): 367–381.

Lin, Wei-Cheng, Da-Yin Liao, Chung-Yang Liu, and Yong-Yao Lee. 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.

Martin, William. 2002. "Satellite Image Collection Optimization." *Optical Engineering* 41 (9): 2083–2087.

Matloff, Norm. 2008. "Introduction to Discrete-Event Simulation and the Simpy Language." *Davis, CA. Dept of Computer Science. University of California at Davis. Retrieved on August* 2 (2009): 1–33.

Nag, Sreeja, Charles K. Gatebe, and Olivier de Weck. 2015. "Observing System Simulations for Small Satellite Formations Estimating Bidirectional Reflectance." *International Journal of Applied Earth Observation and Geoinformation* 43: 102–18. https://doi.org/10.1016/j.jag.2015.04.022.

Nag, Sreeja, Charles Gatebe, D. W. Miller, and O. L. De Weck. 2016. "Effect of Satellite Formation Architectures and Imaging Modes on Global Albedo Estimation." *Acta Astronautica* 126 (April): 77–97. https://doi.org/10.1016/j.actaastro.2016.04.004.

Nag, Sreeja, Alan Li, and James Merrick. 2018. "Scheduling Algorithms for Rapid Imaging Using Agile Cubesat Constellations." *COSPAR Advances in Space Research* 61 (3): 891–913.

Peral, Eva, Simone Tanelli, Ziad Haddad, Ousmane Sy, Graeme Stephens, and Eastwood Im. 2015. "Raincube: A Proposed Constellation of Precipitation Profiling Radars in CubeSat." In *Geoscience and Remote Sensing Symposium (IGARSS), 2015 IEEE International*, 1261–1264. IEEE. http://ieeexplore.ieee.org/abstract/document/7326003/.

Piacentini, C., S. Bernardini, and C. Beck. 2019. "Autonomous Target Search with Multiple Coordinated UAVs - Research - Royal Holloway, University of London." *Journal of Artificial Intelligence Research*. https://pure.royalholloway.ac.uk/portal/en/publications/autonomous-target-search-with-multiple-coordinated-uavs(6b804079-8369-43f8-bb68-2f3f981dc0de).html.

Robinson, Eric, Hamsa Balakrishnan, Mark Abramson, and Stephan Kolitz. 2017. "Optimized Stochastic Coordinated Planning of Asynchronous Air and Space Assets." *Journal of Aerospace Information Systems*. https://arc.aiaa.org/doi/full/10.2514/1.I010415.

Sandau, R., H. P. Roeser, and A. Valenzuela. 2010. *Small Satellite Missions for Earth Observation: New Developments and Trends*. Springer Verlag. http://books.google.com/books?hl=en&lr=&id=D6LaoU-VsxQC&oi=fnd&pg=PR6&dq=sandau+small+satellite+missions+for+earth+observation&ots=VjuMKOSfDa&sig=reMkSbdiHAfAuUWUpH8X5D8FupQ.

Scott, K., and S. Burleigh. 2007. "RFC 5050: Bundle Protocol Specification." *IRTF DTN Research Group*.

Shao, Elly, Amos Byon, Chris Davies, Evan Davis, Russell Knight, Garett Lewellen, Michael Trowbridge, and Steve Chien. 2018. "Area Coverage Planning with 3-Axis Steerable, 2D Framing Sensors." In *Scheduling and Planning Applications Workshop , International Conference on Automated Planning and Scheduling*. Delft, The Netherlands.

Standard, Proposed Draft Recommended, and Proposed Red Book. 2018. "Schedule-Aware Bundle Routing."

Vega Cartagena, Manuel A., Jeffrey R. Piepmeier, James Garrison, Joseph J. Knuble, Cornelis F. Du Toit, and Matthew A. Fritts. 2018. "Signals of Opportunity-Airborne Demonstrator (SoOP-AD): Instrument Overview, Performance during First Flights and Future Instrument Concept [STUB]."

Xhafa, Fatos, Junzi Sun, Admir Barolli, Alexander Biberaj, and Leonard Barolli. 2012. "Genetic Algorithms for Satellite Scheduling Problems." *Mobile Information Systems* 8 (4): 351–377.

Scheduling with Complex Consumptive Resources 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

Generating and scheduling activities is particularly challenging when considering both consumptive resources and complex resource interactions such as time-dependent resource usage. We present three methods of determining valid temporal placement intervals for an activity in a temporally grounded plan in the presence of such constraints. We introduce the *Max Duration* and *Probe* algorithms which are sound, but incomplete, and the *Linear* algorithm which is sound and complete for linear rate resource consumption. We apply these techniques to the problem of scheduling awfor a planetary rover where the awake durations are affected by existing activities. We demonstrate how the *Probe* algorithm performs competitively with the *Linear* algorithm given an advantageous problem space and well-defined heuristics. We show that the *Probe* and *Linear* algorithms outperform the *Max Duration* algorithm empirically. We then empirically present the runtime differences between the three algorithms. The *Probe* algorithm is currently baselined for use in the onboard scheduler for NASA's next planetary rover, the Mars 2020 rover.

Introduction

In many space missions, consumptive resources such as energy or data volume limit the number of activities that can be scheduled. These consumptive resources are oftentimes replenished periodically or gradually over time. For example, data is downlinked—replenishing data capacity—or energy is generated by solar panels or radioisotope thermoelectric generator (RTG) power supplies. The scheduler must therefore schedule activities while staying aware of resource replenishment in order to ensure that the resource state does not violate constraints (e.g. energy below a specified level or data buffers overflow). We focus on awake and asleep scheduling for a planetary rover, but our techniques generalize scheduling in the presence of complex consumptive resource activities.

We focus on the onboard scheduler for NASA's next planetary rover, the Mars 2020 (M2020) rover (Jet Propulsion Laboratory 2018a). Since the heart of our paper is awake and asleep scheduling, we concentrate on energy as the limiting consumptive resource. The M2020 rover's power source is a Multi-Mission Radioisotope Thermoelectric Generator (MMRTG) (Jet Propulsion Laboratory 2018b). The MMRTG constantly generates energy for the rover's battery, but the CPU's awake and "idle" state (i.e. no other tasks) consumes more energy than the MMRTG provides. Therefore, the rover can only increase its energy, measured as battery state of charge (SOC), when the rover is asleep. The rover, however, must stay awake to not only execute activities, but also (re)-invoke the scheduler to generate a schedule. The M2020 onboard scheduler is responsible for generating and scheduling these awake periods.

In order to generate and schedule awakes, the scheduler must compute valid start times for awakes and activities jointly to ensure that there is sufficient energy for both the awake and the activities. Each activity, however, requires varying awake sizes depending on existing awake periods and the activity's scheduled start time. If the activity is close to an existing awake, it may be necessary to extend an existing awake rather than generating a new awake as this would require the rover to shutdown and wakeup in quick succession (Figure 1) which may lead to issues if the shutdown runs longer than nominally expected. Due to its varying duration, an awake's energy consumption and valid start times are challenging to determine.

The remainder of the paper is organized as follows. First, we describe the timeline representation, which is also used by the M2020 onboard scheduler. We discuss calculating valid start time intervals—intervals in which starting the activity would not violate any constraints—and define the problem in relation to the timeline framework. Second, we discuss a general case-by-case approach to handling automatically generated awakes and the challenges specific cases pose. Third, we present three specific approaches to handling these challenges when generating and scheduling awakes: a) an over-conservative approach that always uses the maximum awake period potentially required by the activity when calculating valid intervals; b) a "probing" approach that only considers a single point in time rather than the entire interval; and c) a linear algebra approach that calculates exact valid intervals given the linear rate of energy replenishment and consumption. The "probing" approach is currently base-lined for the M2020 onboard scheduler. Fourth, we present empirical analysis to compare their de-

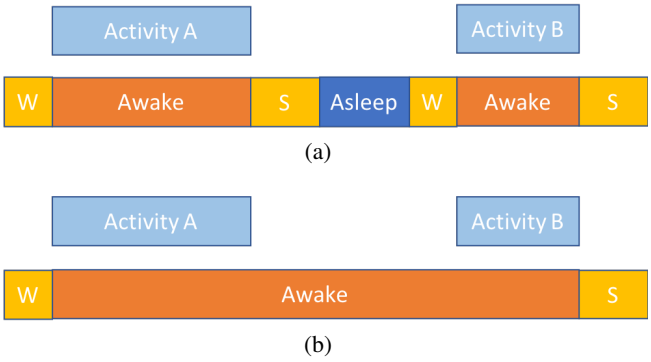


Figure 1: When scheduling activity B, the scheduler should extend the existing awake rather than creating a new one to account for the possibility that the shutdown runs longer than nominally expected. W is a wakeup and S is a shutdown.

grees of completeness and runtime performance. Lastly, we reference related works, describe future works, and discuss conclusions.

Timeline Representation

Timelines are commonly used to model resource utilization and temporal constraints (Chien et al. 2012) and are used for the M2020 onboard scheduler as well. The timeline framework used for the M2020 onboard scheduler projects the impact of activities on shared states and resources (Rabideau and Benowitz 2017). The following is a summary of the timeline library; more detail can be found in Rabideau and Benowitz 2017.

A timeline, $T_1\langle N_1, C_1, B_1\rangle \dots T_l\langle N_l, C_l, B_l\rangle$, is a collection of:

- Timeline impacts, $n \in N_i$, which are a change in the timeline at a specific time, $t(n)$, such as a value assignment or a change in incremental rate.
- Timeline constraints, $c \in C_i$ which are maximum or minimum limits for timeline values over a period of time; if a value exceeds the limit then there is a conflict.
- Timeline bounds, $b \in B_i$, which are maximum or minimum range for values. Values that fall outside the range are truncated, but not conflicts. This is useful to portray the maximum battery capacity, for example.
- Timeline values—where $v_i(t(u))$ is the value (e.g. SOC, current state) at time $t(u)$ for timeline T_i—may be calculated from the timeline impacts, N_i.

Each timeline also has the ability to find a) any conflicts that currently exist on the timeline and b) valid intervals for a new set of impacts, N'. Activities scheduled in their valid intervals create no new conflicts on the timeline. Methods (a) and (b) can also be limited to certain intervals.

There are multiple types of timelines based on practical and specific use cases (Chien et al. 2012; Rabideau et al. 1999; Knight, Rabideau, and Chien 2000), but we focus in particular on the Cumulative Rate timeline. Cumulative Rate timelines allow changes in incremental rate in addition to changes in value. This allows us to represent the incremental SOC drained and gained from being awake and asleep.

Valid Intervals Typically, valid intervals for a Cumulative Rate timeline are calculated by taking in activities as a set of impacts, N', and temporarily placing N' at the time of each existing impact, $n \in N$. Activities are each represented as a set of start and end impacts separated by the duration of the activity. The offset between each impact in N' is fixed in relation to the earliest impact, $n'_{earliest} \in N'$. N' is temporarily placed at the time of each impact, $n \in N$, such that $t(n'_{earliest}) = t(n)$. This determines if any conflicts are generated and, thus, determines N''s valid intervals. The impacts in N' are applied based on their offset from $n'_{earliest}$. There are potentially $\mathcal{O}(N)$ impacts making this an $\mathcal{O}(N' \cdot N)$ operation. When an impact is applied, the effects to the timeline must be propagated into the future of which there are $\mathcal{O}(N)$ potential impacts. Thus, the overall runtime is $\mathcal{O}(N' \cdot N^2)$. $N >> N'$ resulting in an effective runtime of $\mathcal{O}(N^2)$. Constant time slope and intercept calculations compute any values between impacts.

These calculations are dependent on fixed impact offsets in N'. When impact offsets are not fixed, as is the case with non-constant awake durations, valid interval calculations are more complex. At each impact, the offsets in N' are derived from a function rather than a constant. Intercept calculations further exacerbate this complication as values can change between impacts. Therefore, we must either heuristically determine fixed offsets or calculate valid intervals using a different algorithm. We present methods for both approaches.

Problem Definition

Assume that the scheduler is given:

- a list of activities
  $A_1\langle w_1, d_1, e_1, r_1, Z_1, S_1\rangle \dots$
  $A_\tau\langle w_\tau, d_\tau, e_\tau, r_\tau, Z_\tau, S_\tau\rangle$,
- where w_i is the scheduling priority of A_i,
- d_i is the nominal, or predicted, duration of A_i,
- e_i is the rate at which the consumable resource energy is consumed by A_i,
- r_i is the preferred start time for A_i,
- Z_i is the set of physical rover zones or instruments (e.g. arm, mastcam) $z_{i_1} \dots z_{i_k}$ that A_i requires to be heated before and during use,
- S_i is the set of start time windows $s_{i_1} \dots s_{i_q}$ that A_i must respect.

The scheduler is also given a global minimum SOC constraint, C_{soc}^{min}. Each activity may also require the automatic generation of: 1) a set of preheat activities, $P_i = \{p_{i_1} \dots p_{i_k}\}$, 2) a set of maintenance heating activities, $M_i = \{m_{i_1} \dots m_{i_k}\}$, or 3) an awake activity, a_i. Preheats are setup activities (i.e. they occur before the activity), while maintenance heating and awakes are companion activities (i.e. they occur during or with the activity).

Our goal is to calculate valid intervals for activity A_i with a focus on its required awake activity, a_i. For this paper, we

Algorithm 1 General Scheduling Algorithm

Input:
$A\langle w, d, e, r, Z, S\rangle$: List of activities and their attributes and resources
C: Constraints for the whole plan (e.g. available cumulative resources)
E: Current state of the spacecraft (state of charge, data volume, activity status)
Output:
U: Resulting schedule

$U \leftarrow \emptyset$
Sort(A) ▷ By highest to lowest priority
for each $A_i \in A$ **do**
  $P_i \leftarrow \emptyset$
  $M_i \leftarrow \emptyset$
  if $Z_i \neq \emptyset$ **then**
    $P_i \leftarrow generate\_preheats(A_i\langle Z_i\rangle)$
    $M_i \leftarrow generate\_maintenances(A_i\langle Z_i, d_i\rangle)$
  end if
  // Consider S_i as a set of disjoint valid intervals
  $I \leftarrow \emptyset$
  for each $S_{i_j} \in S_i$ **do**
    $I \leftarrow I \bigcup Split(s_{i_j})$ ▷ Based on four cases
  end for
  Sort(I) ▷ By proximity to t
  for each $I_j \in I$ **do**
    // Calculate I_j using either
    // Max Duration, Probe, or Linear
    $I_j, a_i \leftarrow valid\_intervals\_with\_awake($
      $A_i\langle d, e\rangle, I_j, P_i, M_i)$
    if $I_j \neq \emptyset$ **then**
      // start is the scheduled start time for A_i
      $start \leftarrow schedule\_activity(A_i, I_j, U)$
      $schedule\_activity(a_i, start, I_j, U)$
      for each $p_{i_k} \in P_i$ **do**
        $schedule\_activity(p_{i_k}, start, I_j, U)$
      end for
      for each $m_{i_k} \in M_i$ **do**
        $schedule\_activity(m_{i_k}, start, I_j, U)$
      end for
    end if
  end for
end for

refer to valid intervals as valid *start time* intervals for activity A_i. An awake activity is always composed of a wakeup and shutdown. When valid interval calculations involve extending an existing awake rather than creating a new one, an existing wakeup or shutdown may be shifted to match the extended awake. Wakeups are all the same duration, as are shutdowns. If x is the duration:

- The MMRTG generates $g(x)$ SOC consistently.
- The rover consumes $f(x)$ SOC when it is awake and "idle".
- Thus, when the rover is awake and "idle" the net change in SOC is $h(x) = g(x) - f(x)$.
- $g(x) \propto x$ and $f(x) \propto$ x.
- $g(x) \geq 0$ and $f(x) \leq 0$.
- $|f(x)| > |g(x)|$ as more energy is consumed when awake and idle than can be generated by the MMRTG.
- $h(x)$ is negative since $|f(x)| > |g(x)|$.

The overall scheduling algorithm is described in Algorithm 1. Scheduling an awake activity mainly involves SOC, which is represented as a Cumulative Rate Timeline. Recall that we can limit the interval considered for valid interval calculations to improve runtime; we consider S_i as such limiting intervals. We assume that S_i is computed or given before the problem begins. In the Mars 2020 use case, S_i is actually the set of intervals after all other resources (e.g. state, dependencies) are considered. These are computed before SOC is considered due to their less significant runtime. As such, they can be generalized as S_i, and used to improve runtime by limiting valid interval calculation ranges.

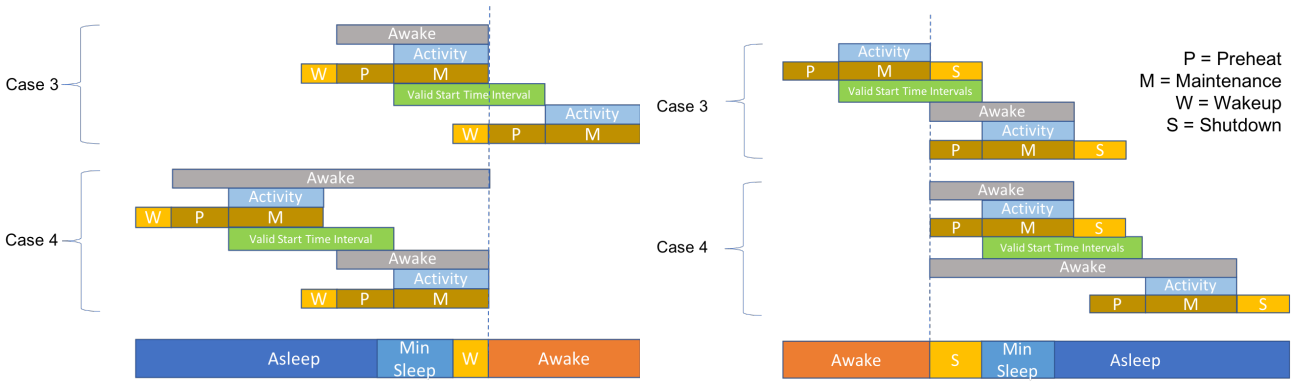
After valid intervals are calculated, the scheduler will place the activity according to its preferred time. Each activity's preferred time, r_i is a soft constraint for activity, A_i. The scheduler will prefer to schedule the start of the activity as close to its preferred time as possible, but is not required to schedule it at that time. Although the actual M2020 scheduler allows multiple preferred times (one for each start time window), we will assume without a loss in generality that there is only one preferred time per activity.

Interval Cases

Valid interval calculations for non-constant duration awakes are complicated for two reasons. a) Standard valid interval calculations assume that the relative time between impacts is constant. This allows the same set of input impacts to be easily and repeatedly applied at different points on the timeline. b) Knowledge about each activity's duration is usually prior knowledge and independent from where the activity will be scheduled; this allows valid interval calculations to focus on one variable (e.g. SOC) as a function of time. Determining valid intervals when duration is dependent on scheduled time is challenging because the calculation must account for multiple variables as a function of time.

In order to schedule awakes, an activity's input intervals, s_i, are split into smaller intervals (I in Algorithm 1). Each smaller interval matches one of the four types dependent on the activities' proximity to existing awakes and constraints. These cases are:

1. *Fully Encompassed by an Existing Awake*. If the set of activities can be scheduled entirely within an existing awake, then there is no need for a new awake activity to be generated.
2. *Disjoint from Existing Awakes*. If the set of activities can be scheduled such that any new awake is completely disjoint from an existing awake, then a new awake that encompasses all the activities must be generated and scheduled.
3. *Overlap with an Existing Awake (Straddle)*. If the set of activities overlaps with an existing awake, but is not fully



(a) Intervals leading an awake.

(b) Intervals trailing an awake

Figure 2: Intervals for each case. The awakes required at the earliest and latest times are shown. Note that these are start time intervals for activity A_i given the known offset of activities in the set of activities A_i, P_i, M_i, a_i

encompassed by the awake, then the overlapped existing awake must be extended to encompass the set of activities.

4. *Overlap with a Minimum Asleep Constraint (Stretch).* To prevent degradation from excessive rover on-off throttling, after each shutdown the rover must stay asleep for a minimum amount of time before waking up again; therefore, there is a minimum asleep constraint both after a shutdown and before a wakeup. In addition, activities requiring an awake cannot be scheduled during a wakeup or shutdown. If the set of activities overlaps with a wakeup, shutdown, or minimum asleep constraint, then the existing awake nearest to that constraint must be extended to encompass the set of activities.

Awake duration is independent of an activity's scheduled start time for intervals matching cases 1 and 2. Case 1 requires no additional awake since it is fully encompassed by an existing awake. Case 2 requires an awake that is equal in duration to the makespan of the set of activities since there are no nearby awakes to potentially extend off of. Thus, these intervals can be handled through previously described valid interval calculations.

For intervals matching cases 3 and 4, the duration of the awake is dependent on where the activities will be scheduled. Case 3 is the *straddle* case as the activities straddle an existing awake. Case 4 is the *stretch* case as the existing awake must stretch to encompass the activities. Both the straddle and stretch cases are similar in that they require the extension of an existing awake, of which the duration (and therefore energy consumption) will vary depending on the placement of the activity. In addition, these intervals can be further categorized depending on if the extension *leads* (Figure 2a) or if it *trails* (Figure 2b) the existing awake. The scheduling algorithm splits the timeline into intervals each matching one of the above cases (Line 13 in Algorithm 1) and calculates valid intervals depending on each case (*valid_intervals_with_awake* in Algorithm 1).

In the following sections we discuss algorithms specifically designed to handle the straddle and stretch cases. Each method describes a way to determine the awake duration. The first assumes an overestimation, the second determines exact durations, but only for a certain times, and the third computes a range of valid durations. The algorithms discussed are all sound, but some are incomplete. Violating mission constraints such as minimum battery SOC would be a significant problem (soundness), and we show in our empirical results that, for both the mission and in general, the incomplete solutions perform acceptably.

Max Duration Algorithm

The Max Duration algorithm assumes the maximum awake duration required to schedule a set of activities. This is a simple, but over-conservative approach to handling non-constant awake duration. Let $start(I_j)$ and $end(I_j)$ be the start and end of the start time interval considered, I_j, for activity A_i. Also let $start(awake)$ and $end(awake)$ be the start and end of the nearest existing awake. The maximum awake duration is $start(awake) - start(I_j) - max(duration(p_{ik}) \in P_i)$ for leading interval cases and is $end(I_j) + d_i - end(awake)$ for trailing interval cases. Figure 2 showcases examples of the maximum awake required for both leading and trailing interval cases.

The benefit of assuming the maximum awake duration is that it allows for simpler valid interval calculations. Constant awake duration leads to a constant relative offset between impacts allowing for previously described valid interval calculations. The downside is that this approach is over-conservative. Depending on where the activities are to be scheduled, a portion of the new awake may overlap with an existing awake resulting in a "double-dipping" of resources. As the approach is over-conservative, it is sound, but incomplete; sometimes it will not find a valid interval to schedule the activities when such an interval exists.

Probe Algorithm

The Probe approach determines the exact duration of the awake, but only for specific points of time in the input interval. Instead of computing valid intervals throughout the

entire input interval, the Probe algorithm checks for conflicts at specific points in time. At each specific point in time, the exact awake duration needed is known, thus avoiding the complications of having a non-constant awake duration.

The algorithm's simplicity is both its strength and weakness. First, the overall runtime is drastically reduced. If k points in time are checked for conflicts instead of at each existing impact, then the runtime for valid intervals is $\mathcal{O}(kN)$ rather than $\mathcal{O}(N^2)$. Usually, only a few specific points are checked (e.g. earliest, latest, midpoint); hence, $k < N$. In our specific approach, we only check at the point nearest to the activity's preferred time; thus, the runtime is $\mathcal{O}(N)$. Due to this runtime improvement, the Probe algorithm is also applied to intervals of cases 1 and 2. Second, while the Max Duration algorithm is over-conservative in terms of awake duration, the Probe algorithm is exact. The downside is that the search does not span the entire interval, only "probing" certain predetermined points of time; therefore, in a sense the Probe algorithm is under-conservative in terms of the interval search space. The Probe algorithm is also sound, but incomplete. While its calculations will be accurate given its knowledge of the exact awake duration, the Probe algorithm will miss valid solutions if the probe locations are not well-defined or unlucky.

Linear

While the other two algorithms are simple or fast, the Linear algorithm uses the linear increase in energy cost and awake duration to calculate exact valid intervals. There are two distinctions to the Linear algorithm. First, the straddle and stretch cases can be regarded as one singular *extension* case because the linear rate of energy does not change between the stretch and straddle cases. Second, the specific steps of this algorithm vary slightly depending on whether the extension *leads* the existing awake (Figure 2a) or if it *trails* the existing awake (Figure 2b); we will discuss the trailing case first.

For activity A_i and input interval I_j the algorithm is as follows:

1. Temporarily apply the activities to the start of the interval, $start(I_j)$, and determine if any conflicts are generated. If conflicts are generated, then there is no valid solution in I_j. If no conflicts are generated, then $start(I_j)$ is the start of the valid interval.
2. Temporarily apply the activities to the end of the interval, $end(I_j)$, and determine if any conflicts are generated. If no conflicts are generated, then all of I_j is a valid interval. If a conflict is generated at $end(I_j)$, then a valid interval exists between $[start(I_j), end(I_j))$.
3. Recall that N_i is the set of timeline impacts currently existing on the timeline T_i, $t(n)$ is the time of impact n, and $v_i(t)$ is the value at time t for timeline T_i. Let l be the point where the asleep begins between $[start(I_j), end(I_j))$. Calculate the valid interval between $[start(I_j), end(I_j))$.
   (a) Determine the point in time, $t(u)$ such that the SOC at that time, $v_{soc}(t(u))$, satisfies both a) $\min_{\forall n \in N_{soc}} v_{soc}(t(n))$ and b) $t(u) > l$). In other words, $t(u)$ is the lowest point on the energy timeline after l.
   (b) Recall that $f(x)$ is the energy consumed while staying awake and that C_{soc}^{min} is the global minimum SOC constraint. Calculate $x \ni f(x) = v_{soc}(t) - C_{soc}^{min} - energy\_cost(A_i, P_i, M_i)$.
   (c) Determine the point in time, $t(u')$ such that its SOC, $v_{soc}(t(u'))$, satisfies both a) $\min_{\forall n \in N_{soc}} v_{soc}(t(n))$ and b) $start(I_j) \leq t(u') \leq l$.
   (d) Recall that $h(x) = g(x) - f(x)$ is the net change in SOC while the rover is awake. Calculate $x' \ni h(x') = v_{soc}(t(u')) - C_{soc}^{min} - energy\_cost(A_i, P_i, M_i)$.
   (e) $[start(I_j), start(I_j) + min(x, x')]$ is the valid interval for the activity.

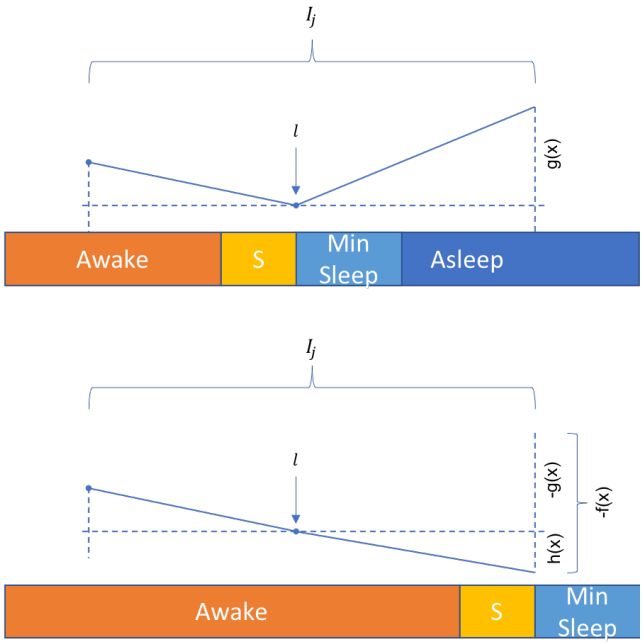


Figure 3: Energy changes from extending an awake. l is the point where the asleep begins without the awake extension.

Since steps 1 and 2 are evaluated at a single point in time—similar to the Probe algorithm—the exact duration of the awake is known. For step 1, it is the minimum awake and, thus, if a conflict is generated there is no valid solution in I_j. For step 2, it is the maximum awake and, thus, if there is no conflict the entire range must be valid. In step 3a, we determine the maximum amount of energy that can be used without violating the minimum soc constraint as $v_{soc}(t) - C_{soc}^{min}$. In step 3b, we use this and the energy cost function to determine exactly how long the awake can potentially be. We can subtract the energy cost of the other activities (A_i, P_i, M_i) because they are not affected by their placement. Steps 3c and 3d differ only slightly from steps 3a and 3b. The difference occurs because of how the awake affects the SOC timeline. In steps 3a and 3b, any

point (including the minimum point $t(u)$) after $end(I_j)$ decreases by $f(x)$ as seen in Figure 3. Recall that energy is only gained when the rover is asleep. Staying asleep between $(l, end(I_j)]$ generates $g(end(I_j) - l)$. For simplicity, let us temporarily assume $end(I_j) - l$ as x. Any point after $end(I_j)$ must deduct $g(x)$ since that energy will not be generated if the awake is extended. $-f(x) = h(x) - g(x)$ is the resulting energy lost from any point after $end(I_j)$. Between $[start(I_j), l]$, however, the energy has not been gained yet. As a result, $g(x)$ does not need to be deducted from any point in $[start(I_j), l]$. Using the energy function, we can calculate how long the maximum awake can be without violating the minimum SOC constraint. Step 3e simply translates the now known awake duration into valid intervals.

The main difference when applying this algorithm to the trailing case is that the existing awake extends in the opposite direction. This results in several changes. First, at $end(I_j)$ the awake duration is the minimum, and at $start(I_j)$ the awake duration is the maximum; therefore, do steps 2 and then 1. Secondly, steps 3c and 3d can be eliminated. Extending the awake from $end(I_j)$ makes it impossible for any point between $[start(I_j), end(I_j)]$ to violate the minimum SOC constraint without a point after $end(I_j)$ violating the constraint first. Lastly, in step 3e $[end(I_j) - min(x, x'), end(I_j)]$ is the valid interval.

This algorithm is both sound and complete. We can show that it is complete through a proof by contradiction. Let us assume we are scheduling in a trailing extension interval and a point, $t(u'')$, exists between $(start(I_j) + min(x, x'), end(I_j)]$ such that A_i can be scheduled without violating C_{soc}^{min}. The awake extension duration needed is $x'' = t(u'') - start(I_j)$. We know that $x'' > min(x, x')$ otherwise it would have been in our solution. For any point after $end(I_j)$, the total energy consumed is $f(x'') + energy\_cost(A_i, P_i, M_i)$. Recall that $v_{soc}(t(u))$ is the SOC of the lowest point after $end(I_j)$. Therefore, a) $v_{soc}(t(u)) - (f(x'') + energy\_cost(A_i, P_i, M_i)) \geq C_{soc}^{min}$ otherwise the minimum SOC constraint would be violated. For any point after $end(I_j)$, the total energy consumed is $h(x'') + energy\_cost(A_i, P_i, M_i)$. Recall that $v_{soc}(t(u'))$ is the SOC of the lowest point between $[start(I_j), l]$. Therefore, b) $v_{soc}(t(u')) - (h(x'') + energy\_cost(A_i, P_i, M_i)) \geq C_{soc}^{min}$ otherwise the minimum SOC constraint would be violated. However, both statements (a) and (b) cannot be true as $f(x'') > f(min(x, x')$. Therefore, a point between $(start(I_j) + min(x, x'), end(I_j)]$ that does not violate the minimum SOC constraint cannot exist. This proof can be similarly done for leading extension intervals.

The trade off for completeness, however, is that each valid interval requires more calculations. As a result, it increases both code complexity and runtime costs. The question is: Is the increased completeness worth the costs?

Empirical Results

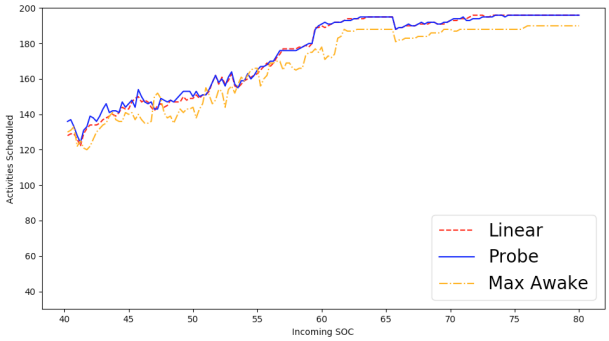
To evaluate the performance of each method, we apply each algorithm to various sets of inputs comprised of activities and their constraints. The inputs are derived from *sol types* which are currently the best available data on expected Mars 2020 rover operations (Jet Propulsion Laboratory 2018a). Each input file contains between 20 and 40 activities, and our goal is to schedule as many activities as possible. We apply our algorithms on top of the *M2020 surrogate scheduler* - a Linux workstation implementation of the same algorithm as the M2020 onboard scheduler (Rabideau and Benowitz 2017) - to construct a schedule and simulate plan execution. The M2020 surrogate scheduler is expected to produce the same schedules as the operational scheduler, but lends itself to more rapid research and development on a linux workstation environment. While the baseline scheduler utilizes the Probe algorithm, we interchange with the Linear and Max Duration algorithms; the remainder of the algorithm is identical to the operational scheduler.

We compare each method against varying incoming SOC levels to vary the level of difficulty for scheduling. The incoming SOC is the SOC remaining after the previous schedule; therefore, it is the SOC that the current schedule begins with. Since energy consumption is the main constraint and focus of sleep scheduling, varying the incoming SOC will vary the set of correct valid intervals and, by extension, the difficulty of the problem. If the incoming SOC is sufficiently high, the sleep scheduling problem is easy as there is sufficient energy for the rover to constantly stay awake and activities can be scheduled more freely. This is because each sol type was specifically constructed so that all activities would be schedulable given a reasonable (e.g. 75 percent) incoming SOC. As the incoming SOC decreases, the sleep scheduling problem gets harder as the algorithms must determine if there is sufficient energy to schedule a new awake or extend an existing awake when considering activity placement. We first analyze which algorithm performs the best as the problem difficulty increases. Secondly, we evaluate the runtimes of each algorithm.

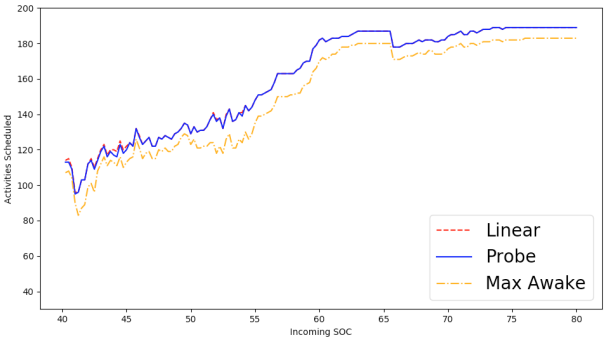
Results

Completeness Figure 4a showcases how the scheduler performs with each respective sleep scheduling algorithm. As the incoming SOC decreases, the problem difficulty increases and fewer activities are scheduled. While the conservative Max Duration approach clearly performs worse than the others, the distinction between the Probe and Linear approaches seems unclear. This may seem surprising given that the Linear algorithm is more complete than the Probe algorithm; there are, however, multiple reasons as to why the algorithms perform similarly despite their difference in completeness.

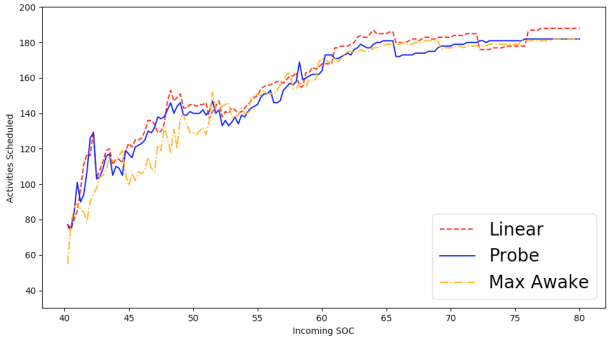
First, the scheduler focuses on finding the locally optimal state—where the local optimal is scheduling the current considered activity as close to its preferred time as possible—but does not focus on global optimality. Due to the rover's computational limitations, the M2020 onboard scheduler is a one-shot, non-backtracking scheduler. When considering each activity, the scheduler attempts to schedule it as close to the activity's preferred time as possible, but does not consider the impact it may have to any future activities. Since activities are not moved or removed after they have been considered for the schedule, the current scheduler cannot guarantee global optimality, regardless of the valid



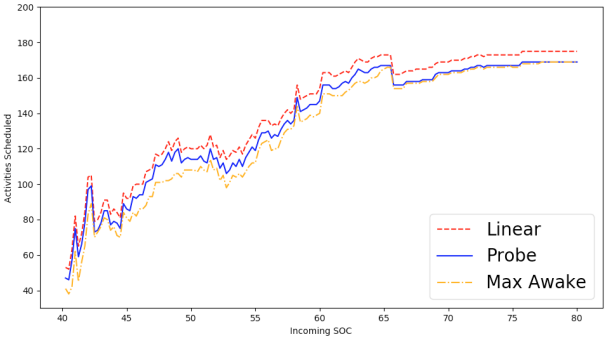
(a) Full schedules generated. Baseline wakeup (5 minutes) and shutdown (10 minutes) durations.



(b) Partial schedules generated. Baseline wakeup (5 minutes) and shutdown (10 minutes) durations.



(c) Full schedules generated. Extended wakeup (30 minutes) and shutdown (60 minutes) durations.



(d) Partial schedules generated. Extended wakeup (30 minutes) and shutdown (60 minutes) durations.

Figure 4: As the Incoming SOC Increases, resources are less constrained and more activities are able to be scheduled. When generating a schedule for all activities from an empty schedule, the differentiation between the Probe and Linear algorithms is unclear. When using the baseline wakeup and shutdown durations, it is clear that the Max Duration algorithm performs the worst; if the wakeup and shutdown durations are extended, however, the Max Duration algorithm starts to perform better. When scheduling only one activity at a time from a partial schedule, the Linear algorithm strictly outperforms the other two algorithms. When the wakeup and shutdown durations are extended, the difference becomes even more clear.

interval algorithm chosen. To account for this, the M2020 team has developed Copilot, a ground based tool that uses Monte Carlo and Squeaky Wheel Optimization (Joslin and Clements 1999) to adjust scheduling priorities before the plan is uplinked to the rover. To read more about how activity priorities are set, see (Chi et al. 2019).

Take for instance the following example: the Linear algorithm successfully finds valid intervals for a longer activity for which the Probe algorithm cannot. This may, however, result in the Linear algorithm being unable to schedule either multiple shorter activities or an activity that multiple future activities depend on, while the Probe algorithm (due to having excess energy from not scheduling the activity) may successfully schedule multiple future activities.

To balance the scheduler's focus on local optimality, we considered scenarios where only the next activity in the scheduling algorithm is considered. To do this, we generated partial schedules where the first i activities are scheduled by the same algorithm, but the $i+1$ activity is scheduled with the different algorithms and compared. This is repeated for every $i \in m$ so that every activity is scheduled with the same algorithm for the first i activities. We used the Probe algorithm to schedule the first i activities as it is the current baseline for the M2020 scheduler. In Figure 4b, we see that the Linear algorithm strictly outperforms (although the outperformance is minimal) the Probe method while the Max Duration method strictly under-performs. These results reaffirm our initial belief that the Linear algorithm is complete and more locally optimal than the other methods. The fix to the local vs global optimality issue thus lies more with the overall scheduling algorithm than with the valid interval calculations.

Second, the stretch and straddle regions are not large enough to cause a drastic difference between algorithms. Currently, wakeups are 5 minutes, shutdowns are 10 minutes, and the minimum sleep duration is 20 minutes. The makespan of meaningful activities in a sol is around 8 to 10 hours of which there are usually only a few wakeups and shutdowns. Since the algorithms only differ in how the stretch and straddle regions—regions encompassing wakeup and shutdowns—are handled, the differences in the algorithms are minute. By increasing the length of the stretch and straddle regions, the differences between the algorithms become more clear. In Figures 4c (full schedule) and 4d (partial

schedule), we increased the wakeup duration to 30 minutes and shutdown durations to 60 minutes. Indeed, the Linear algorithm starts to perform better in comparison to the Probe algorithm with the differences especially evident in Figure 4d. The Max Duration algorithm also performs better than before as its advantage over the Probe algorithm (it searches the entire range rather than just one point) is able to be more utilized.

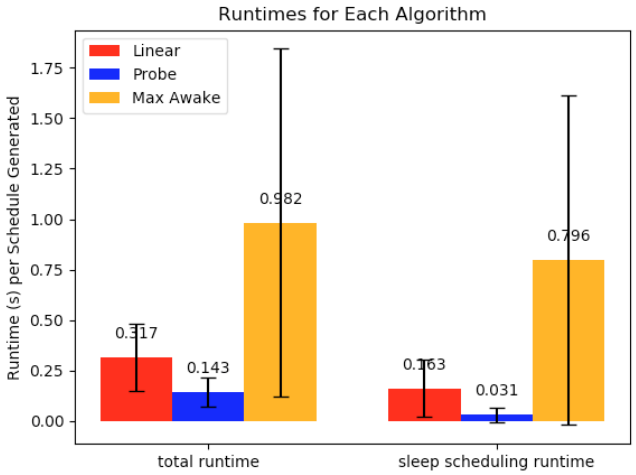


Figure 5: Left is the runtime for the overall scheduling algorithm. Right is the runtime sleep scheduling part of the algorithm only (i.e. *valid_intervals_with_awake*). The Probe algorithm greatly outperforms the other algorithms while the Max Duration algorithm greatly underperforms.

Runtime While completeness is important, the runtime performance of each algorithm is equally critical to determining which algorithm to use. To accurately compare the algorithms, we compare both the total scheduling runtime (left) and the runtime of the algorithms after prior computations and constant factors are deducted (right). In figure 5, the Probe algorithm significantly outperforms the other two algorithms as expected. Recall that the Probe algorithm is an $\mathcal{O}(N)$ algorithm while both Linear and Max Duration are $\mathcal{O}(N^2)$. Not only that, but the Probe algorithm can be applied to non stretch and straddle intervals as well, further speeding up runtime. Despite its simple nature, the Max Duration algorithm is the slowest algorithm by a wide margin. Since the Max Duration algorithm is often unable to find valid intervals due to its conservatism, it must often search through and calculate valid intervals for multiple stretch and straddle regions. As a result, its simple valid interval calculations is offset by its need to do this calculation multiple times for each activity. On the other hand, the Linear algorithm may take longer per region, but needs to explore less regions overall. This is further evidenced by the large standard deviation seen in the Max Duration algorithm's runtime; if it can find a valid interval within the first few regions it considers it is fast, but if it cannot then it is incredibly slow. Although the difference between tenths of a second may seem small, these computations are done on a linux workstation instead of onboard the rover; when eventually run onboard, a single scheduler run can take up to 1 minute. As a result, the two factor increase in total runtime between the Probe and Linear algorithms is substantial.

Future Work

There are a few topics to be considered for future research. First, preheat and maintenance heating pose a similar energy management challenge to the scheduler. While they have been intentionally glossed over for this paper, preheat and maintenance heating are also dependent on existing preheats and maintenances; an existing maintenance may be extended instead of requiring a new preheat. Preheats differ in that an activity may require preheats for multiple regions on the rover and preheat durations may vary depending on thermal conditions (Rabideau and Benowitz 2017). We would like to analyze the different algorithms used to schedule preheats and maintenances as well. Second, we would like to analyze runtimes onboard the rover. In this paper, we analyzed the runtimes on a linux workstation and compared it against an estimate of how long a scheduler run takes on a flight-like processor. The onboard scheduler runtime estimate is, however, only run with the baseline algorithm (i.e. Probe); our analysis would be further substantiated if we were able to run each algorithm onboard a flight-like processor.

Related Work

Schedulers have a long history of handling consumptive resources.

ASPEN-EO-1 not only took into account onboard data storage (among other constraints) when scheduling plan observations, but summarized or deleted data depending on onboard data analysis (Chien et al. 2005a; 2005b; 2010).

MEXAR2 addressed Mars Express's Spacecraft Memory Dumping Problem (MEX-MDP) and synthesized data downlink plans that took into account data storage capacity (Cesta et al. 2007).

MAPGEN was used to plan operations for Mars Exploration Rovers (MER) Spirit and Opportunity(Bresina et al. 2005). MAPGEN similarly managed battery SOC as a consumptive resource, but MER rovers relied on solar power rather than a MMRTG. In addition, this system addresses ground-based rather than onboard planning.

Remote Agent is an onboard autonomous agent architecture that takes into account consumptive resources such as energy and data volume (Muscettola et al. 1998). It mainly utilizes constraint posting and propagation rather than "fixed" start times and is a more general architecture compared to our specific solution to scheduling consumptive resources.

Conclusion

Generating and scheduling activities in the presence of consumptive regenerative resources is especially challenging when a driving factor of feasibility of placement is dependent on interactions with the existing schedule. Scheduling activities and their awake periods is particularly challenging in the context of M2020 because the awake's

duration is dependent on existing awakes. We presented three algorithms—Max Duration, Probe, and Linear—for scheduling awakes and analyzed their completeness and runtime. Despite being a locally sound and complete algorithm, the Linear algorithm was not always able to outperform in the global problem space. We demonstrated how a simple and incomplete algorithm can perform both suboptimally, as seen with the Max Duration algorithm, and also close to optimal, as seen with the Probe algorithm, dependent on the heuristic and input parameters. We showed that the Probe algorithm is a fair alternative to a more complete algorithm, especially considering its ease of implementation and runtime improvement.

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

Bresina, J. L.; Jónsson, A. K.; Morris, P. H.; and Rajan, K. 2005. Activity planning for the mars exploration rovers. In *International Conference on Automated Planning and Scheduling (ICAPS 2005)*, 40–49.

Cesta, A.; Cortellessa, G.; Fratini, S.; Oddi, A.; and Policella, N. 2007. An innovative product for space mission planning: An a posteriori evaluation. In *International Conference on Automated Planning and Scheduling (ICAPS 2007)*, 57–64.

Chi, W.; Chien, S.; Agrawal, J.; Fosse, E.; and Guduri, U. 2019. Optimizing parameters for uncertain execution and rescheduling robustness. In *International Conference on Automated Planning and Scheduling (ICAPS 2019)*.

Chien, S.; Cichy, B.; Davies, A.; Tran, D.; Rabideau, G.; Castano, R.; Sherwood, R.; Mandl, D.; Frye, S.; Shulman, S.; et al. 2005a. An autonomous earth-observing sensorweb. *IEEE Intelligent Systems* 20(3):16–24.

Chien, S.; Sherwood, R.; Tran, D.; Cichy, B.; Rabideau, G.; Castano, R.; Davis, A.; Mandl, D.; Trout, B.; Shulman, S.; et al. 2005b. 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. A.; Tran, D.; Rabideau, G.; Schaffer, S. R.; Mandl, D.; and Frye, S. 2010. Timeline-based space operations scheduling with external constraints. In *International Conference on Automated Planning and Scheduling (ICAPS 2010)*, 34–41.

Chien, S.; Johnston, M.; Policella, N.; Frank, J.; Lenzen, C.; Giuliano, M.; and Kavelaars, A. 2012. A generalized timeline representation, services, and interface for automating space mission operations. In *International Conference On Space Operations (SpaceOps 2012)*.

Jet Propulsion Laboratory. 2018a. Mars 2020 rover mission https://mars.nasa.gov/mars2020/ retrieved 2018-12-17.

Jet Propulsion Laboratory. 2018b. Mars 2020 rover mission https://mars.nasa.gov/mars2020/mission/rover/electrical-power/ retrieved 2018-12-17.

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

Knight, R.; Rabideau, G.; and Chien, S. A. 2000. Computing valid intervals for collections of activities with shared states and resources. In *AIPS*, 339–346.

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.

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

Rabideau, G.; Knight, R.; Chien, S.; Fukunaga, A.; and Govindjee, A. 1999. Iterative repair planning for spacecraft operations using the aspen system. In *Artificial Intelligence, Robotics and Automation in Space*, volume 440, 99.

Planning & Scheduling with Preferences

Privacy-aware Adaptive Scheduling for Coalition Operations

Karen L. Myers,[1] Thomas Lee,[1] Laura Tam,[1] José Manuel Calderón Trilla,[2] Benjamin Davis,[2] Stephen Magill[2]
Artificial Intelligence Center, SRI International[1]
333 Ravenswood Ave., Menlo Park, CA 94025
firstname.lastname@sri.com
Galois, Inc.[2]
421 SW 6th Avenue, Suite 300
Portland, Oregon 97204
firstname.lastname@galois.com

Abstract

Coalition operations are essential for responding to the increasing number of world-wide incidents that require large-scale humanitarian assistance. Many nations and non-governmental organizations regularly coordinate to address such problems but their cooperation is often impeded by limits on what information they are able to share. In this paper, we consider the use of an advanced cryptographic technique called secure multi-party computation to enable coalition members to achieve joint objectives while still meeting privacy requirements. Our particular focus is on a multi-nation aid delivery scheduling task that involves coordinating when and where various aid provider nations will deliver relief materials after the occurrence of a natural disaster. Even with the use of secure multi-party computation technology, information about private data can leak. We describe how the emerging field of quantitative information flow can be used to help data owners understand the extent to which private data might become vulnerable as the result of possible or actual scheduling operations, and to enable automated adjustments of the scheduling process to ensure privacy requirements.

Introduction

Coalition operations are becoming an increasing focus for many nations. The need for collaboration derives from increasing awareness of mutual interests among both allies and nations that traditionally have not worked closely together. For example, coalition operations for Humanitarian Assistance and Disaster Relief (HADR) have increased substantially in recent years in both numbers and scope. With the impact of global warming, it is anticipated that there will be even more large-scale humanitarian crises resulting from adverse weather-related events and sea-level rises. These coalitions often involve not just government and military organizations but also non-governmental organizations (NGOs) and commercial entities.

A key challenge facing coalitions is how to collaborate without releasing information that could jeopardize national (or organizational) interests. Information security mechanisms for the military to date have focused on safeguarding information by limiting its access and use. This approach has lead to a significant *undersharing* problem, which impedes effective joint operations. Recent work on defining access control mechanisms is useful for enabling selective sharing of information that is safe to release (Phillips, Ting, and Demurjian 2002). However, many coalition tasks require information that participants do not wish to make available to others.

For example, consider the problem of scheduling the delivery of aid (food, water, medicine, fuel) to impacted nations after a natural disaster. Historically, international response has involved dozens of countries and NGOs, each with the desire to contribute in interconnected and potentially conflicting ways. Ideally coordination would be achieved by directly sharing information about the amount of aid each contributor has available or can produce, where that aid is situated, the position and storage capacity of ships for delivering the aid, harbor facilities where aid ships could dock, etc. But the owners of this data may be unwilling to directly share it with coalition partners, for fear of revealing information that impacts national security (e.g., ship locations) or competitive advantages (e.g., a company's backlog of inventory).

To address this problem of coalition collaboration without revealing private information, we exploit a cryptographic technology called *secure multi-party computation (MPC)* (Yao 1982). MPC protocols enable mutually distrusting parties to perform joint computations on private information while it remains encrypted. In our work, we use MPC to enable privacy-preserving computations over several types of coordination tasks related to scheduling aid delivery.

While participants cannot directly learn others' private inputs from the process/protocol of a secure multi-party computation, they may be able to infer something about private data based on the results of the computation. For this reason, our privacy-aware scheduling solution makes use of a complementary technology called *quantitative information flow (QIF)* to measure the degree to which private data used in the scheduling task might leak to other participants. Insights gained from this analysis are folded back into the scheduling process via an adaptive workflow capability, to ensure that the vulnerability of private data stays within acceptable thresholds.

The paper is organized as follows. We begin by summarizing our aid distribution task, including a description of the core scheduling problem and the data (private and

non-private) belonging to the various coalition members. We then provide a short overview of secure multi-party computation followed by a description of how we employ it within a broader adaptive workflow framework to address the aid delivery scheduling task. Next, we describe our use of quantitative information flow to assess the vulnerability of private information as the scheduling progresses and to adapt the scheduling workflow in light of those assessments to ensure adherence to predefined privacy requirements. We conclude by identifying directions for future work and summarizing our contributions.

HADR Aid Delivery Problem

The aid delivery problem that we consider involves two categories of participants:

- N *Aid Provider* nations, each of which has some number S_n of ships with aid (e.g., food, medicine) to be delivered
- a single *Aid Recipient* nation that has P ports to which aid can be delivered

Collectively, these participants need to formulate a schedule for delivering aid on board the Aid Provider ships to ports belonging to the Aid Recipient, ensuring delivery prior to a specified deadline. As summarized in Figure 1a , a solution involves assigning to each Aid Provider ship: a port to which its aid should be delivered, a berth at the port, and a docking time. These assignments are subject to various constraints on ship and port characteristics to ensure physical compatibility between the ship and the port/berth, schedule availability of the berth, and the ability for the ship to completing the docking before the assigned deadline. We further seek an assignment that optimizes according to the following load-balancing criteria.

Optimization Criteria: Load-balancing across ports. Let $\mathit{Assigned}(\text{Port}_i)$ designate the number of aid provider ships assigned to aid recipient port Port_i. An optimal solution is a set of assignments that minimizes

$$\text{MAX}_{j,k}(\mid \mathit{Assigned}(\text{Port}_j) - \mathit{Assigned}(\text{Port}_k) \mid)$$

Generating a solution to this scheduling problem requires information from the various parties about their assets (ships, ports), some of which they would prefer not to share with other coalition members. Figure 1b summarizes this private data. In both Figure 1b and Figure 1a the problem data is color-coded, with blue used for private data belonging to an Aid Provider nation and green for private data belonging to the Aid Recipient nation. As the coloring clearly shows, determining a solution requires combining private information from multiple parties, which motivates our use of secure multi-party within our scheduling algorithm.

Secure Multi-Party Computation

To address privacy concerns in our aid delivery use case, we leverage MPC. MPC protocols compute a function and reveal the output of the computation without revealing private inputs to any other participant. Early MPC work was based on two-party problems (Yao 1982), and subsequent work produced general approaches for any number of participants to participate in shared computation without learning private information (Goldreich, Micali, and Wigderson 1987).

Most MPC approaches involve modeling the desired algorithm as a boolean circuit (fixed-time algorithms expressed as a combination of `AND`, `OR`, and `NOT` logic gates). However, circuit-based approaches may require unrolling loops, introducing potential performance implications. Alternate approaches based on Oblivious RAM (Goldreich and Ostrovsky 1996) do not require this full unrolling though have other performance trade-offs. We use a circuit-based MPC approach in our scenario, though our privacy analysis and adaptive scheduling do not depend on the specifics of the underlying MPC approach.

MPC has proven to be a powerful tool to enable a wide range of use cases. For example, private keys can be split among several hosts in order to reduce the number of hosts that must be compromised in order to obtain the key. A computation that requires the decryption operation can then be done under MPC between the hosts, ensuring that the whole key is never revealed in the clear (Archer et al. 2018). MPC can also be used between companies in collaborative supply-chain management, where the reluctance to share sensitive data (such as costing and capacities) can lead to sub-optimal production plans. The use of MPC allows for collaborative supply-chain planning without compromising sensitive data (Kerschbaum et al. 2011).

In our setting, each data owner provides their private input to a MPC circuit designed such that their inputs are not revealed to other participants. Then participants follow the protocol for the multi-party computation, which allows them to collectively compute the solution to the planning and scheduling optimization problem. In the end, only the final result (schedule) is revealed to the relevant parties.

Scheduling Workflow

Creating a single MPC circuit to solve the full optimized scheduling problem in a general way is not practical at this point in time. For this reason, our solution approach consists of a workflow that decomposes the scheduling task into the following sequence of steps.

Step A. Collect relevant inputs:

- Aid Provider: determine ports that can be reached by each ship before the deadline D
- Aid Recipient: select ports to which aid will be accepted

Step B. Determine ports that satisfy joint ship/harbor physical compatibility constraints:

- Aid Provider ship draft is compatible with the harbor depth in the port
- Aid Recipient port has sufficient offload capacity

Step C. Determine all berths within each feasible port from Step B that satisfy joint ship/berth compatibility and berth availability constraints:

- Aid Provider ship fits within the berth

For a **Ship** to be scheduled to dock at a given **Docking Time** at a **Berth** belonging to a **Port**, the following must hold:

Port

- **ship-draft** $\leq$ **port-harbor-depth**
- ship-cargo-amount $\leq$ **port-cargo-offload-capacity**
- **ship-earliest-arrival** $= \frac{\text{EarthDistance}(\textbf{ship-location}, \text{port})}{\textbf{ship-maxspeed}}$
- **ship-earliest-arrival** $\leq$ deadline

Berth in Port

- **ship-length** $\leq$ **berth-length**

Docking Time at Berth

- **ship-earliest-arrival** $\leq$ docking-time $\leq$ deadline
- **berth-availability** (pairs of **berth-occupied-start** and **berth-occupied-end**) for berth at docking-time

(a) Constraints in the Scheduling Problem

Aid Provider	Aid Recipient	
Ship Info	Port Info	Berth Info
ship-location	**port-available**	**berth-length**
ship-length	**port-cargo-offload-capacity**	**berth-occupied-start**
ship-draft	**port-harbor-depth**	**berth-occupied-end**
ship-maxspeed		

(b) Summary of Private Data by Data Owner

Figure 1: Constraints over the private data and the relationship of private data to the data owners

- Aid Recipient berth is available at planned ship arrival time

Step D. Schedule the aid ships across possible ship-port-berth-arrival-time options (from Step C) in accordance with the optimization goal of load-balancing across ports.

Step A is a simple information gathering/filtering task performed locally by individual participants. Step B requires computation over private inputs from both the Aid Provider and the Aid Recipient. Steps C and D combine private inputs with intermediate results from earlier steps. For these reasons, we use three secure multi-party circuits within our workflow:

- A two-party circuit for the physical compatibility test in Step B (Aid Provider and Aid Recipient)
- A two-party circuit for the viability test in Step C (Aid Provider and Aid Recipient)
- An N-party circuit for optimizing berth allocation in Step D (all N Aid Providers)

The circuits for Steps B and C are straightforward but must be run for each possible ship/port (Step B) and ship/berth (Step C) combination. The circuit for Step D is more complex. Because it is not possible to implement a truly optimized solution in the MPC circuit model, we instead opted for the following greedy approach to load balancing across ports.

1. Initialization: set the list of ship-port-berth solutions to be empty and the working set of options to be the set of ship-port-berth-unload time entries from Step C
2. Select a port P with the fewest number of assignments and for which there remain options
3. Select earliest ship-port-berth-unload time entry for P and add to the solutions list
4. Remove from the set of options all entries for the ship and berth selected in Step 3
5. Repeat steps 2-4 until no more ship-port-berth solutions remain
6. Output the solution list

We use the Lumen agent technology (described in (Myers et al. 2011)) to provide an adaptive workflow capability for executing this scheduling process. Although we depict only one workflow here, more generally our adaptive workflow capability draws from a library of alternative approaches to a range of aid distribution scheduling problems, enabling solution approaches to be matched to the specifics of a given situation. For example, we have defined alternative workflows that embed different scheduling and optimization strategies and that make use of secure multi-party computation in different ways as a means of investigating tradeoffs between efficiency and privacy.

Quantifying Information Vulnerability

In this section we describe what is involved in using QIF to reason about questions of how private data can be inferred from the results of a computation.

Limitations of Multi-Party Computation

The "Millionaires' Problem" (Yao 1982) is an early use-case for multi-party computation in which two people, Alice and Bob, use MPC to securely determine which of them is richer without revealing their actual wealth to each other. But while they don't learn the other's precise values, they do learn *something* about each other.

The nature of the relationship between the revealed output and the private inputs determines how much one may infer about the inputs from the result. For example, if Alice and Bob use MPC to instead compute the arithmetic mean of their net worths, Alice could use the input she provided and the resulting mean from the computation to solve for Bob's exact net worth. Alice may not be able to extract Bob's input via examination of the MPC circuit itself, but MPC cannot change the relationship between inputs and output that exists outside the circuit.

In our Aid Delivery scenario, private "inputs" may involve sensitive capability and operational details. As the scenarios increase in scope and complexity, it becomes difficult to reason informally about what an adversary may be able to infer.[1] We use Quantitative Information Flow (QIF) to address these concerns by characterizing an adversary's ability to make these inferences based on information-theoretic bounds on the relationship between results and private inputs.

Modeling Workflows

A QIF analysis begins by transforming a program (such as our aid distribution workflow) into a model that represents the relationship between the (private) inputs and outputs. Specifically, these models are based on information-theoretic *channels*, which we use to construct a mapping from prior[2] distributions on private data to distributions on posterior distributions (Smith 2011; Smith 2009).

We use these models to support "predictive-mode" adaptive workflows in which we reason about what an adversary could learn from any possible set of private inputs, as well as "posterior-mode" in which we determine how much an adversary may learn from some specific concrete result.

Quantifying Inference Capabilities

We can construct games in which we quantify the adversary's inference capability by measuring how their chances of "winning" (i.e., guessing a piece of private data) improve given additional information. We call the probability that an adversary can with one chance correctly guess a piece of private data the *vulnerability* of that variable.

In the "prior game," the defender picks a private input value by sampling from the prior distribution of possible input values. The adversary makes their best guess of the defender's input based only on their knowledge of the prior distribution of possible input values. The *prior vulnerability* is the probability the adversary will correctly guess the defender's private input (without any additional information).

In the "posterior game," the defender again picks a private input, but then performs the computation using that input and shares (only) the result with the adversary. We also assume the adversary has full knowledge of how the inputs relate to outputs in the computation (e.g., could construct an identical channel-based model of the computation). Here the chance the adversary correctly guesses the defender's private input is called the *posterior vulnerability*.

To summarize, the prior game is how likely an adversary is to guess the private input *before* seeing the result of the computation, the posterior game is how likely the adversary is to guess the private input *after* seeing the *specific output* of the computation.

The Use of Vulnerability as a Metric

Vulnerability as a metric of risk is appealing because of its relatively intuitive nature. The caveat to vulnerability as a metric is that it is extremely sensitive to the chosen prior belief. Therefore, care must be taken in deciding upon a prior in order to assure that the metric reflects reality as much as possible.

Supporting Workflow Adaptation

Our predictive-mode adaptation strategy is based on QIF *predictive leakage*, which attempts to predict how much will be leaked about the private data before the computation is run on the concrete private values. We can also approximate predictive leakage using Monte Carlo simulation, permitting the use of these metrics even in scenarios where operational requirements make it infeasible to complete the precise predictive leakage within a desired time frame. Incorporating predictive leakage metrics into our workflows enables identification of potentially higher-risk situations where it may be preferable to adapt or halt the workflow (based on some policy) rather than participating in a computation that may reveal too much.

Our posterior-mode adaptation strategy is based on QIF *dynamic leakage*, which takes into account the actual results of a computation. As opposed to the predictive leakage's assessments, that average all possible private input values, dynamic leakage enables our workflows to incorporate more accurate assessments of what was actually revealed in the specific ongoing workflow.

Sample Vulnerability Analysis

As discussed previously, the HADR aid delivery problem provides a strong foundation for analyzing adaptive workflows in a privacy-aware setting. In this section we describe how the notion of *vulnerability* can be used to inform the cooperating parties of risk to their private data.

A Simplifying Example

To begin our discussion we first describe a simplified version of the HADR aid delivery problem involving a single ship, S, and a single port, P. In this simplified scenario, the only private data is S's location (`ship_loc`). We assume that

[1] In this work, an 'adversary' is a so-called *honest-but-curious* actor who attempts to infer the values of private data by observing the public results of each step in the workflow. Adversaries could be other coalition partners or an unrelated third party that has access to the computational framework.

[2] A 'prior' can be thought of as the initial set of beliefs that an adversary has about a system. An adversary may have a prior over the lengths of naval ships (e.g., between 10 and 1,500 feet long), or the possible locations of the ships. If an adversary has no reason to believe one value is more likely than any other then the prior is a uniform distribution over the space of secrets, hence it is known as a *uniform* prior.

other relevant data (such as ship maximum speed) is known publicly.

```
def reachable(deadline):
  dist = distance(ship_loc, port_loc)
  return (dist <= (max_speed * deadline))
```

Figure 2: The pseudo-code for a simple query

The channel is "is S able to reach port P within d hours?". As pseudo-code we write this channel as a function of d, over some global set of private (`ship_loc`) and public (`port_loc`, `max_speed`) variables.

Prior Belief In our simplified scenario an adversary who wants to determine S's private data will have some *prior* belief about that data. In this case, the prior belief will be some area of the ocean where S could be located. This may or may not be informed by other information available to the adversary. While QIF can be calculated over non-uniform priors, in this exposition we assume uniform priors for the sake of simplicity. Figure 3a shows a graphical representation of a possible prior for the simplified scenario.

Because our prior is uniform, the adversary's chance of guessing the ship's position (the *prior vulnerability*, V_{prior}), is simply

$$V_{\text{prior}} = \frac{1}{\text{number of possible locations in the prior}}$$

This makes intuitive sense: under a uniform prior, the adversary's likelihood of winning the prior game is equivalent to choosing a point out of the prior at random.

Posterior Belief When ship S, cooperating in the simplified scenario, responds to the `reachable` query in Figure 2, the adversary is able to observe the *output* of the channel. This observation allows the adversary to rule out entire sections of the space of possible locations.

If the result of the query is `True`, then the adversary can infer that the location of S is within a circle of radius r, where r is the distance that the ship can travel at `max_speed` with `deadline` amount of time. Inversely, if the result of the query is `False`, then the adversary could infer that the ship must be *outside* of the same circle. Observing `False` is the circumstance illustrated in Figure 3b.

The important point is to see that *regardless of the result* of the query, the adversary may be able to infer rule out subsets of possible values for the private data.

The probability the adversary has of guessing the ship's position after seeing the channel output (the *posterior vulnerability*, V_{post}) is simply

$$V_{\text{post}} = \frac{1}{\text{number of possible locations in the posterior}}$$

Further Queries Because cooperation is the goal, it is likely that the coordinator of our simple scenario will need to query S multiple times. This is particularly true if the initial query's result was `False`. The coordinator may query S with `reachable(d2)`, where $d2$ is a new, longer, deadline. If the result is then `True` then the adversary is able to infer that S resides within the ring formed by two overlapping circles: an inner circle with radius r, described above, and a wider circle with radius $r2$, where $r2$ is the distance that S is able to travel at max speed in time $d2$. This circumstance is illustrated in Figure 3c.

Worst-Case Simple Scenario If we add another port, $P2$, to the simple scenario above, but keep all other details the same, it may be possible for S's location to be determined within a very small tolerance. If the result of `reachable` is `True` for both P and $P2$, then (using the same process as above) the adversary would intersect the appropriate circles. The smaller the intersection, the more the adversary knows about the ship's position.

Analysis of the Aid Distribution Scenario

The simplified scenario above only has one piece of private data: the ship's position. As a reminder, Figure 1b shows the private data in the full HADR scenario. In this section we present some results from an analysis over a model of the full scenario, using a single ship (Ship #9) as a running example.

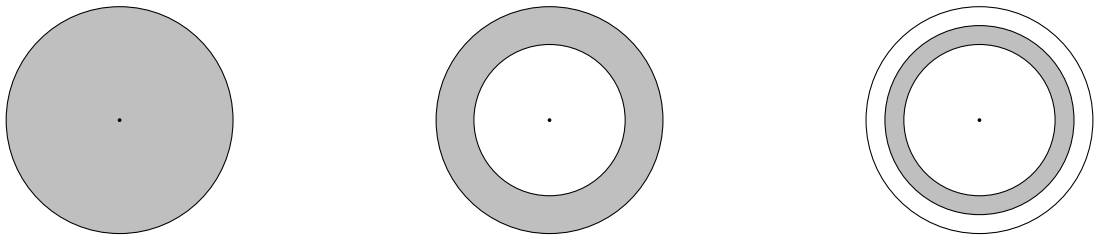
Prior Vulnerability As discussed above, an adversary has *some* notion of the possible values that a secret can take on even before running any computations over that secret data. This is referred to as the *Prior*. When analyzing a system, the analyst must *choose* appropriate ranges for the private data in question. For some variables this may be straightforward (e.g., the ocean sector in question when deciding on a prior for a ship's location), for others it may depend on domain knowledge (e.g., the appropriate range for naval ship drafts).

We can visualize the prior vulnerability over location easily. Figure 5 shows several aspects of the HADR scenario. The boundaries of the map in the image are the boundaries of the prior belief over ship position. It is important to remind ourselves that the choice of prior (part of which is determined by the geographic area under consideration) affects the vulnerability metric significantly. Increasing the area *decreases* the prior vulnerability, while decreasing the area *increases* the prior vulnerability. Because the prior models the *adversary's* view of the world, it should be constructed carefully.

Predictive Vulnerability for Steps A+B In resource-constrained systems, it is useful to be able to predict how much of a given resource would be used if a certain action were to be executed. Private data can similarly be viewed as a form of resource for which predictive analysis can be applied to assess the impact of planned or possible actions.

The ability to predict the future vulnerability can be implemented in several ways, the practicality and efficiency of different methods depends heavily on the constraints of the system, imposed by the state-space and the adversarial model.[3]

[3]In the QIF literature, this ability is knowns as *Static Leakage Analysis* (Smith 2009; Alvim et al. 2012).



(a) The initial prior belief: no reason to believe the ship is in one location over any other.

(b) Posterior belief after observing a `False`: The ship must be *outside* the distance of the `reachable` query.

(c) If a subsequent query for a further distance returns `True` the ship must be within the newly formed ring.

Figure 3: The effects of observing query results on a prior belief over the simplified scenario.

In our system we have chosen an approximate method that enables an analyst to calculate a histogram over the possible vulnerability outcomes without any access to the private data. This method uses Monte Carlo simulation to run our analysis over randomly sampled points from the space of possible private data values.

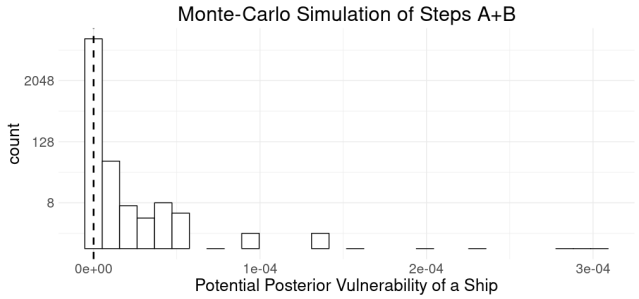


Figure 4: Predictive Vulnerability of Steps A+B

Figure 4 shows the results of a Monte Carlo simulation of the predictive vulnerability of running Steps A+B (using 13788 samples[4]). The vertical dotted line shows the median value ($6.847603\text{e}{-6}\%$) of all the samples. This method of approximation provides analysts with various options. In a scenario where the preservation of privacy is paramount, the analyst may focus on the right-hand side of the figure, where the potential vulnerability is higher, 0.03039514%, though still unlikely.

This method also adapts well to use cases where an analyst may have access to *some* of the private data. For example, an analyst for one of the coalition partner nations might have access to that nation's private data, but not the private data of the ports. Using this same method of sampling from the space of (unknown) private data, such an analyst would be able to approximate the future vulnerability of their data if they were to respond to a query.

Posterior Vulnerability for Steps A+B Once a step is taken, we can calculate the *posterior vulnerability* of the private data. Unlike the predictive analysis in the previous section, this analysis is 'exact' in that the real vulnerability cannot be more than what the analysis reports.

Ship #9 Position $6.847603\text{e}{-6}\%$

In this case, the posterior vulnerability coincides with the median of the predictive vulnerability. This is not too surprising as the median was also the most likely value by a significant margin.

Predictive Vulnerability for Step C Figure 6 shows the Monte Carlo simulation for vulnerability of a ship after Step C is completed.[5] The median predicted value in this instance, $6.644298\text{e}{-4}\%$, is two orders of magnitude higher than the vulnerability after Steps A+B alone. This makes intuitive sense as much more information is revealed after Step C that can be used to infer a ship's private data. Unsurprisingly, the maximum sampled predictive vulnerability is also substantially higher: 0.2012072%.

Posterior Vulnerability for Steps C As with the posterior vulnerability for Steps A+B, the posterior vulnerability for Step C is based on a sound analysis using the real results of the workflow step, and are not simulated as in the predictive vulnerability.

Ship #9 Position $2.049806\text{e}{-4}\%$

In the case of Step C, the posterior vulnerability for Ship #9 is *lower* than the median of the predictive result ($6.644298\text{e}{-4}\%$). From an analyst's perspective, this could mean that Ship #9 has revealed even less about its private data than the 'average' ship would in this scenario.

Why Step D doesn't leak Step D has no meaningful consequences on the vulnerability of the private data for any stakeholder in the HADR scenario *if the result of Step C has been observed by the adversary*. The reason for this is that Step D's algorithm can be computed completely from the results of previous steps in the workflow, i.e. it does not require the values of the private data directly. Interestingly, this point reinforces an important aspect of QIF analysis: even though Step C's result was computed with private data, the vulnerability metrics from the QIF analysis of Step C take

[4]The sampling procedure is time based, hence the seemingly arbitrary number of samples.

[5]Note that the range of the x-axis has changed in order to better display the data.

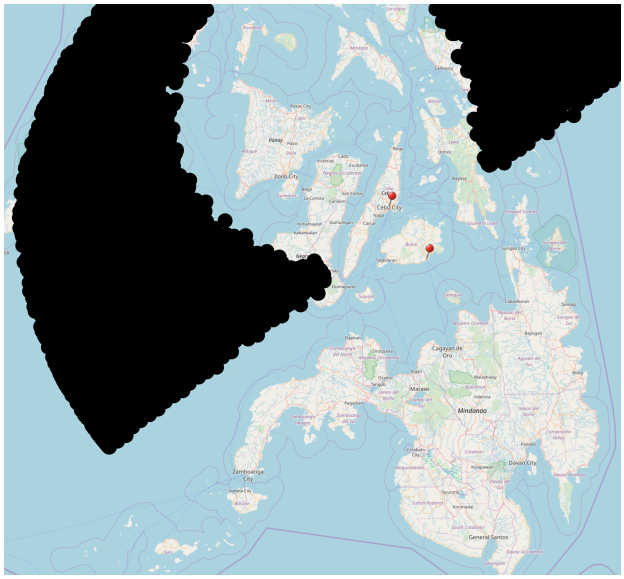


Figure 5: The posterior belief over the position of Ship #9 in the HADR scenario after cooperating in all steps. (Map cartography and tiles © OpenStreetMap contributors.)

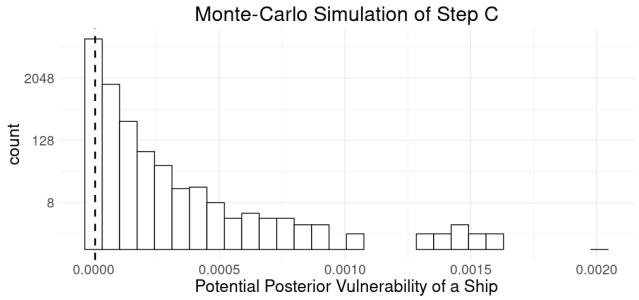


Figure 6: Predictive Vulnerability of Step C

into account *any possible use* of the result. Therefore additional computation over the results of Step C (or any prior step) does not affect the vulnerability of private data.

Privacy-aware Workflow Adaptation

The vulnerability assessment computed by the QIF capability provides insights to data owners as to the security of private information that they wish to protect. We exploit these insights within our workflow manager to adapt the scheduling process in order to ensure adherence to privacy objectives. More specifically, we use the QIF capability in two ways: in a *predictive mode* to estimate the amount of leakage associated with potentially performing a particular query or task and in a *posterior mode* to track actual leakage based on the specific values that a query or task returns.

When executing a particular task our workflow manager invokes the predictive mode to estimate leakage. If the estimate of aggregate leakage for designated private data does not exceed set thresholds, then the workflow proceeds and the posterior leakage analysis is invoked to determine actual leakage values. If the estimate does exceed the threshold then the workflow is either terminated or (if possible) modified via a *remediation strategy* to keep leakage below the threshold.

The idea behind a remediation strategy is to modify the problem or state in ways that will likely reduce impacts on private data. For example, our aid delivery problem requires computing the reachability of a port by a given deadline. Knowing that a given ship can reach the port by that deadline reveals information about the combination of ship position and maximum speed. One simple remediation strategy is to postpone the deadline, which will reveal less information about the position and speed values (e.g., the fact that a ship can reach a port by a given deadline reveals something about the lower bound for its max-speed; a later deadline introduces greater uncertainty as to what that speed might be by decreasing that lower bound). Our Lumen-based adaptive workflow engine includes such remediation strategies to enable adaptivity based on QIF predictive analyses.

Privacy thresholds are implemented using an existing policy framework within Lumen that was developed previously to enable users to impose boundaries on the behaviors of autonomous agents (Myers and Morley 2003). The privacy policies have the general form:

```
Keep below <percentage>
the probability of knowing
<private-data> within <tolerance>
```

Below we show two examples used in the system, one for the Aid Provider nation and one for the Aid Recipient nation.

Aid Provider Sample Policy:

"Keep below 10 % the probability of knowing the location of my ships within 50 NMs"

Aid Recipient Sample Policy:

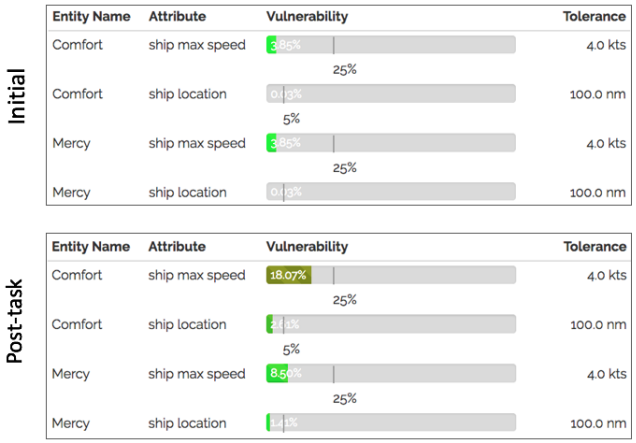


Figure 7: Comparison of initial (prior) and post-task (posterior) vulnerability assessments

"Keep below 20 % the probability of knowing the port harbor depth within 40 feet"

Figure 7 shows sample vulnerability assessments for two types of private data (max-speed, location) for a select set of ships belonging to an individual Aid Provider nation. The top image shows the initial vulnerabilities of the data, prior to performing any computations; the bottom image shows the vunlerabilities after workflow completion. The display shows the QIF-derived vulnerability level as a colored bar representing the adversary's likelihood of guessing the private data within the specified tolerance. The vertical line bisecting the display for each piece of private data marks the policy-prescribed threshold of acceptability for the vulnerability. We note that the initial vulnerabilities for the ship locations are non-zero but so small as to not be perceptible in the image.

Future Work

Here, we consider two avenues for future work.

Analogs can be drawn between our use of the QIF analysis to predict and track vunlerabilities of private data within the scheduling workflow and prior work on estimating resource usage in workflows (Morley, Myers, and Yorke-Smith 2006). Although we currently consider individual actions incrementally as the workflow executes, we envision performing predictive vulnerability assessments of entire workflows prior to execution, to enable informed choices about alternative approaches before any information usage has occurred. Generating useful assessments will require predictive QIF techniques that consider cases beyond worst-case leakage, whose inherent pessimism can make them of limited value for certain vulnerability assessment tasks. Longer term, such analyses could also potentially open the door to using first-principles planning techniques to synthesize privacy-aware workflows on an as needed basis that are tailored to the specifics of a given task and privacy requirements.

The scalability of QIF techniques can be an issue for systems where there are complex relationships between sets of variables. Some work has been done on attaining scalability by enhancing static analysis techniques with approximations that speed up analysis with probabilistic bounds on certainty (Sweet et al. 2018). However, there is still further work required before the QIF analysis of arbitrary channels could scale to use cases such as the end-to-end HADR scenario considered in this paper. In particular, the methods described in this paper utilize bespoke analyses for the channels under consideration, providing a more scalable approach at the cost of generality. One future direction may be to design Domain Specific Languages that enable description of a scenario and its analysis in tandem.

Conclusion

A key challenge facing coalitions is how to collaborate without releasing information that could jeopardize national (or organizational) interests. In this paper, we consider this challenge for a realistic scheduling problem tied to aid delivery. Our work makes several contributions. First, we show how state-of-the-art secure multi-party computation can be used to safeguard private information with an overall distributed scheduling solution to the aid delivery problem. A second contribution relates to the use of quantitative information flow (QIF): even with secure multi-party computation, scheduling outputs can reveal information about coalition members' private data. We show how QIF can be applied to assess the vulnerability of providate data for both prospective (i.e., where results are not known) and actual (i.e., where results are known) computations. As a third contribution, these assessments can be used to adapt the scheduling algorithm to ensure it remains within accepted vulnerability thresholds established by data owners.

Acknowledgments

This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA), the United States Air Force, and the Space and Naval Warfare Systems Center, Pacific (SSC Pacific) under Contracts No. FA8750-16-C-0022 and N66001-15-C-4071. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA, the Department of Defense, the United States Air Force, or SSC Pacific.

The authors thank Stealth Software Technologies, Inc. (in particular; Steve Lu, Rafail Ostrovsky, Paul Bunn, and Chongwon Cho) for providing the MPC software that was used on this effort.

References

[Alvim et al. 2012] Alvim, M. S.; Chatzikokolakis, K.; Palamidessi, C.; and Smith, G. 2012. Measuring information leakage using generalized gain functions. In *2012 IEEE 25th Computer Security Foundations Symposium*, 265–279. IEEE.

[Archer et al. 2018] Archer, D. W.; Bogdanov, D.; Lindell, Y.; Kamm, L.; Nielsen, K.; Pagter, J. I.; Smart, N. P.; and

Wright, R. N. 2018. From Keys to Databases—Real-World Applications of Secure Multi-Party Computation. *The Computer Journal* 61(12):1749–1771.

[Goldreich and Ostrovsky 1996] Goldreich, O., and Ostrovsky, R. 1996. Software protection and simulation on oblivious rams. *J. ACM* 43(3):431–473.

[Goldreich, Micali, and Wigderson 1987] Goldreich, O.; Micali, S.; and Wigderson, A. 1987. How to play any mental game. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, 218–229. New York, NY, USA: ACM.

[Kerschbaum et al. 2011] Kerschbaum, F.; Schroepfer, A.; Zilli, A.; Pibernik, R.; Catrina, O.; de Hoogh, S.; Schoenmakers, B.; Cimato, S.; and Damiani, E. 2011. Secure collaborative supply-chain management. *Computer* 44(9):38–43.

[Morley, Myers, and Yorke-Smith 2006] Morley, D. N.; Myers, K. L.; and Yorke-Smith, N. 2006. Continuous Refinement of Resource Estimates . In *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multi Agent Systems*, 858–865.

[Myers and Morley 2003] Myers, K., and Morley, D. 2003. Policy-based Agent Directability. In Hexmoor, H.; Castelfranchi, C.; and Falcone, R., eds., *Agent Autonomy*. Kluwer. 143–162.

[Myers et al. 2011] Myers, K.; Kolojejchick, J.; Angiolillo, C.; Cummings, T.; Garvey, T.; Gervasio, M.; Haines, W.; Jones, C.; Knittel, J.; Morley, D.; et al. 2011. Learning by demonstration technology for military planning and decision making: A deployment story. In *Twenty-Third IAAI Conference*.

[Phillips, Ting, and Demurjian 2002] Phillips, Jr., C. E.; Ting, T.; and Demurjian, S. A. 2002. Information sharing and security in dynamic coalitions. In *Proceedings of the Seventh ACM Symposium on Access Control Models and Technologies*, SACMAT '02, 87–96. New York, NY, USA: ACM.

[Smith 2009] Smith, G. 2009. On the foundations of quantitative information flow. In *International Conference on Foundations of Software Science and Computational Structures*, 288–302. Springer.

[Smith 2011] Smith, G. 2011. Quantifying information flow using min-entropy. In *Proceedings of the 2011 Eighth International Conference on Quantitative Evaluation of SysTems*, QEST '11, 159–167. Washington, DC, USA: IEEE Computer Society.

[Sweet et al. 2018] Sweet, I.; Calderón Trilla, J. M.; Scherrer, C.; Hicks, M.; and Magill, S. 2018. What's the over/under? probabilistic bounds on information leakage. In *International Conference on Principles of Security and Trust*, 3–27. Springer, Cham.

[Yao 1982] Yao, A. C. 1982. Protocols for Secure Computations. In *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, 160–164.

Planning and Scheduling for Cooperative Concurrent Agents with Different Qualifications in the Domain of Home Health Care Management

Colja A. Becker and Ingo J. Timm
Business Informatics I, Trier University
Behringstrasse 21 - Campus II
54296 Trier, Germany

Abstract

Rising demand for home health care services combined with a shortage of professionals leads to increasing workload of existing employees. However, there is a performance limit so that it will no longer be possible to offer these services maintaining quality and economic viability without changing operational management. To cope with this situation, we propose a concept of task bundle splitting options among several cooperative agents with different qualifications. Further, we integrate this concept in a planning and scheduling algorithm for multiple concurrent agent actions which can increase efficiency and can improve use of limited resources in operational processes. For this purpose, possible concurrent actions of agents with different qualifications were evaluated combined with the scheduling process and compared to alternatives. As a first step, this contribution presents the concept as well as an algorithm generating an optimal solution and gives an insight into future work.

Introduction

Many countries face the challenge of coping with increasing demand for care services. For example, in Germany the number of people in need of care will rise by around 32 percent by 2030, resulting in a shortage of care personnel (Rothgang et al. 2016). Besides stationary facilities and the support of relatives, home health care (HHC) is one possibility to receive care services. Here, caregivers are equipped with cars and drive to the patients' homes to render the required services.

To cope with an increasing demand in HHC, additional caregivers must be hired by service providers. However, the availability of (professional) caregivers on the labor market is very limited. Rising demand for HHC services combined with a shortage of professionals leads to the problem that the workload of existing employees increases. There is a performance limit so that it will no longer be possible to offer HHC services maintaining quality and economic viability without a change in operational management. Following this, managing existing human resources in HHC gains in relevance to enable efficient employment.

Inspired by *cooperative multiagent planning* as well as task decomposition in *hierarchical task network planning*, we propose a concept of task bundle splitting options among several cooperative agents (caregivers) with different qualifications. Further, we integrate this concept in a planning and scheduling algorithm for multiple concurrent agent actions which can increase efficiency and can improve use of limited resources in operational processes in the domain of HHC.

Literature Review

Since an increase in efficiency and an improvement in using limited resources can rise coordination effort, the usage of methods from the field of *multiagent systems* seems suitable. Moreover, knowledge and scheduling issues have a distributed structure among the participants in the domain of HHC. Here, operational management processes in terms of planning and scheduling can be supported by multiagent systems as well as decision support systems using agent technology (Becker, Lorig, and Timm 2019).

López-Santana et al. developed an MAS combined with a *mixed integer programming* model which takes cargivers' qualifications into account such that corresponding scheduling and routing is achieved (López-Santana, Espejo-Díaz, and Méndez-Giraldo 2016). The approach aimed at minimizing travel times of caregivers as well as delays in arrival times at customer locations. Similar, the approach by Xie and Wang focuses on minimizing service costs by creating an initial schedule using an optimization model and the schedule will be updated periodically during runtime (Xie and Wang 2017). The latter is based on communication between agents and a central re-scheduling. The approach by Marcon et al. uses a global optimizer to assign each caregiver to a set of customers with a corresponding route proposal, which can be adapted later by the caregiver (Marcon et al. 2017). Following this, a scheduling and routing solution is given. Here, each caregiver interacts with his own patients, so interchangeability is not possible, and there is no coordination between caregivers in order to reach a better joint solution. By changing the local decision-making mechanisms, different higher-level objectives can be pursued, e.g., minimizing waiting times. Remaining approaches which support planning and scheduling in operational HHC management surveyed by Becker et al. provide information management, standard scheduling solutions, frameworks, communication platforms, and basic coordination solutions.

Considering all approaches, the use of several different

qualifications in conjunction with joint processing of subsets of tasks for better deployment of limited resources has not been considered so far.

Problem Description

In cooperation with an experienced HHC provider, we analyzed different processes related to the operational management of service provision. This also includes planning and scheduling in this domain. The following description of the problem is derived from the associated observations.

A service provider employs a set of caregivers C and each of them has his or her own level of education. These qualification levels are given by the set Q. Every service from the set of all provided services S is assigned to a qualification level, too. Further, each service $s \in S$ has a specific duration.

$$u : C \to Q \qquad w : S \to Q \qquad Q \subset \mathbb{N}$$

Following this, executing a service by a caregiver requires a qualification level at least equal (or greater) to the assigned qualification level. Further, an HHC provider has a set of patients P, which is called customers, who request services for certain times of a day. For this purpose, a day is divided into different time intervals Y based on the set T of all time points of a day.

$$Y = \{\ (y^n_{start}, y^n_{end})\ \mid\ y^n_{start}, y^n_{end} \in T\ \ \wedge\ \ y^n_{start} < y^n_{end} \\ \wedge\ \ y^{n-1}_{end} < y^n_{start}\ \ \wedge\ \ y^n_{end} < y^{n+1}_{start}\ \}$$

For example, a day might be divided into an early shift, a midday shift, and a late shift. A customer is able to request a service for one or more shifts per day and may also request different services in the same shift. All possible customer orders (service requests) are specified by the Cartesian product of services S and time intervals Y. Thus, using the power set the function r expresses the demand of a customer $p \in P$.

$$r : P \to \mathcal{P}(S \times Y)$$

One order $(s, y) \in r(p)$ of customer p will be assigned to a caregiver and a certain time window by the function a which corresponds to an operational management task in order to generate a schedule for the considered day. The caregiver assignment has to satisfy the required qualification level $q \in Q$ and the time window must be completely within the time interval given by the corresponding shift $y \in Y$.

$$a : r(p) \to T \times T \times C, \quad (s, y) \mapsto (t_{start}, t_{end}, c)$$
$$\text{where} \quad c \in C\ \ \wedge\ \ y = (y_{start}, y_{end}) \\ \wedge\ \ y_{start} \le t_{start} < t_{end} \le y_{end} \\ \wedge\ \ u(c) \ge w(s)$$

The full schedule Z is often created manually at present by an operational manager for each day in advance using simple scheduling support software without specific scheduling algorithms. A schedule entry contains a customer p as well as one of his orders $x = (s, y)$ and an assigned caregiver $c \in C$ who has to render the requested service in the time window starting at t_{start} and ending at t_{end} at the respective customer's location.

$$Z = \{\ (p, x, j)\ \mid\ p \in P\ \wedge\ x \in r(p)\ \wedge\ j = a(x)\ \}$$

In operations, caregivers are often equipped with mobile devices and corresponding software for knowledge sharing and documentation tasks. Customer data and related order data as well as central schedule data is linked with the software. By using these devices, caregivers know where to go and what to do. Meanwhile, process times for internal documentation associated with service provision at each customer location are automatically recorded by the mobile devices.

The environment is represented by a directed graph $g = (V, E)$ where each customer $p \in P$ is assigned to a definite node $v \in V$ and each node is linked to each other node by a directed edge $e \in E$. Further, each edge is assigned to a value which describes the related travel time. For simplicity, all caregivers start at the HHC office which is also represented by a node of the graph.

$$k : P \to V \qquad f : E \to \mathbb{N}$$

All services for a customer $p \in P$ in a single day's time interval can be referred to as a requested task bundle for this customer. More precisely, a task bundle is a subset of the assigned subset by the function $r(p)$, such that all primitive tasks (services) for the customer p in a specific time interval on a certain day will be performed directly in sequence at the customer's node.

In the following, as a first step the concept will focus on only one time interval without certain time windows for orders. The accomplishment of all task bundles in this time interval is the goal of the problem, i.e. one task bundle for each customer, while minimizing the overall processing time. Depending on the number of caregivers C, the task bundles can be performed concurrently.

Concept

According to the classification by Torreno et al., the concept presented below belongs to the conceptual scheme "Planning *for* multiple agents" (Torreño et al. 2018), in which *one* agent plans and *n* agents execute the plan. In Addition, scheduling is done by using the duration of the actions S as well as the travel times given by the function $f(e)$ for edges $e \in E$. The number of execution agents corresponds to the cardinality of the set C and the planning agent can be seen as a central planning unit at the HHC office. As mentioned before, caregivers are cooperative agents, which share the same goal, and act concurrently. Further, all actions are considered to be deterministic. An action is either an element of the set S, which requires a certain qualification level $q \in Q$ or just a move action to get from one node of the graph g to another one. Obviously, moving between nodes does not require a certain qualification level unlike rendering a specific service at a customer's node.

Task Bundle Splitting

Different qualifications are provided by the agents C and assigning each task bundle to exactly one agent, i.e. assigning each customer to only one caregiver, can result in idle times of some agents with certain qualifications and overload of others depending on the actual conditions, e.g., order situation, travel distances, and distribution of qualifications. Caregivers with a high qualification level are considered as

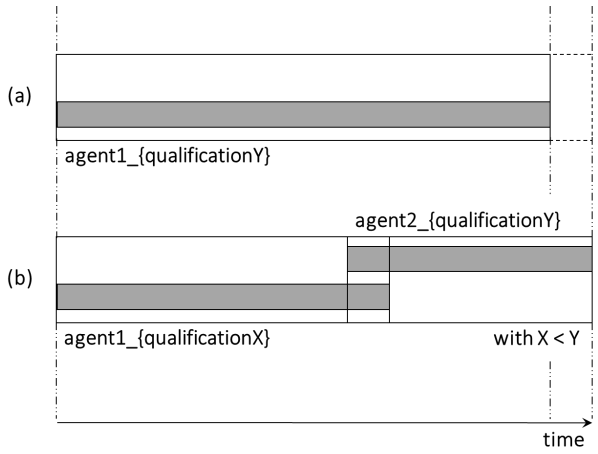


Figure 1: Two alternatives to accomplish a task bundle.

a very limited resource and it is assumed that more agents with a lower qualification level exist. Since some services in a task bundle can require a lower qualification level than other services in the same task bundle, assigning each task bundle to exactly one caregiver may require them to do some work for which they are overqualified. The latter can lead to error susceptibility and dissatisfaction among employees. Especially if caregivers with a high qualification level are highly requested due to the current order situation, the time for tasks of lower qualification levels is quite costly for these employees.

As a main part of this concept, we suggest to compare splitting of task bundles with the conventional procedure as an inherent component of the planning and scheduling process. In Figure 1, the two alternatives of processing a task bundle are depicted. The first variant (a) is the conventional procedure where one task bundle is assigned to exactly one agent. All contained tasks, consisting of moving to the corresponding node and rendering requested services, are performed one after the other by the assigned agent. Here, the agent's qualification level must be sufficiently high to perform all primitive tasks of this task bundle. The alternative (b) shows the splitting procedure. First, the lower qualified agent moves to the customer and makes the usual preparations as well as the other requested services according to its qualification. After completion of these tasks, the higher qualified agent arrives at the customer's location and the two employees exchange information about current customer-related content. The information exchange is encapsulated with a fix time value as a overlapping coordination task for each of these agents starting with the arrival of the second agent. Further, this joint task can be used for customer-related issues which require two caregivers at the same time, e.g., lifting the patient out of bed. In some cases, the second agent does not arrive seamlessly with the completion of the last task of the first agent, so the latter has to wait for the arrival of the second agent, and this time can be used for further concerns of the customer like different human needs or desires. After the joint coordination task, the lower qualified agent moves on with its schedule and the agent with the higher qualification performs the high-qualification services. It is important to note that by accumulating all individual task durations in a task bundle, the entire splitted task bundle takes a greater duration value due to the additional coordination task. Since the agent with the higher qualification level is considered as a very limited resource, the splitted task bundle always starts with the lower qualified agent in order to avoid idle times of the more scarce resource. In addition, the whole customer service, i.e. the entire task bundle, should not be interrupted out of consideration for a humane treatment of the customer. Following this, leaving the customer's location before arriving of the second agent is not permitted to the lower qualified agent.

Temporal Planning

In this concept, planning and scheduling based on the order data for a chosen time interval is executed as *forward state space search*. The initial state contains order data and environment data. The order data comprises all task bundles for the selected time interval, i.e. one task bundle for one customer containing all primitive tasks $s \in S$. To include the customer's location, a node attribute is given for each individual task bundle. As mentioned before, in the goal state all task bundles are accomplished.

Since this phase of our research project neglects runtime complexity, the search for a goal state is performed as simple tree-based *breadth-first search* in which the scheduling process is integrated. In Algorithm 1, the pseudocode for planning and scheduling in the domain of HHC is presented. In order to process every state, a queue is used with a loop and generated successors are added to the queue. If a state still contains task bundles to be done, all possible actions of idle agents were gathered in according to each agent's qualification level. Here, an action means taking a task bundle which is not in progress and not accomplished so far. Further, every combination of the possible actions of all idle agents are computed. In this procedure, a combination contains only actions which are not already assigned to another agent in the same combination. Because agents are acting concurrently, we do not care about the order in this combination, so the term *combination* is used instead of *permutation*. Each generated combination represents a successor link and is then used to create further states. So, a state will be linked to a successor if the state is not a goal state and it contains one or more idle agents which can choose an action to perform. Otherwise, there is still work in progress, so any task bundle already have been done or is currently in progress and the successor state will be a goal state.

As mentioned before, in this concept scheduling is an inherent process using action durations. While creating a successor state by applying an action combination of the respective link, agents which are currently performing an action remain in their statuses and idle agents which are affected by the generated combination will be assigned to further work. Following this, a search procedure is conducted in order to find the next earliest event to set a time value for the state's clock. Such an event is either an accomplished task bundle of an operating agent, i.e. changing the status of an agent,

Input: initial state

Output: schedule/schedules Z

```
1  initialState ← init();
2  queue.add(initialState);
3  WHILE queue.containsElement() {
4      state ← queue.get(0);
5      IF state.orderData.containsElement() {
6          idleAgents ← getAgents(C, state);
7          agentOptions ← ∅;
8          FOREACH c ∈ idleAgents {
9              acts ← computeOptions(c, state);
10             agentOptions.add( (c, acts) );
11         }
12         sLinks ← combinatorics(agentOptions);
13         FOREACH k ∈ sLinks {
14             successor ← createState(k);
15             successor.clock ← searchEvent();
16             queue.add(successor);
17         }
18     ELSE
19         goalStates.add(state)
20     }
21     queue.remove(state);
22 }
23 rStates ← minProcessingTime(goalStates);
24 minZ ← rStates.getSchedules();
25 RETURN minZ
```

Algorithm 1: Pseudocode for planning and scheduling of cooperative agents with concurrent actions.

or a buffer event which is introduced to allow for idle agents to perform an action as a second part of a splitted task bundle which require travel time by moving from one node to the other. By this means, all possibilities of idle agents to take action in time were covered. If one or more agents accomplish a task bundle, they will be added to the set of idle agents which is the starting point of the new state. There is a possibility that these agents may take further actions, and again, every combination of possible actions of all idle agents are computed and further successors are generated.

The possibility of splitting a task bundle is integrated as a part of an action combination, too. If respective qualifications among agents are available, generating action combinations comprises performing a task bundle conventionally by one agent as well as splitting a task bundle into two parts as described in the previous section. While splitting a task bundle, the first part of the lower qualified agent is directly integrated in the action combinations and the second part of this task bundle is added to the order data to allow for successor states to generate further action combinations at the corresponding time value.

In the end, all goal states' processing times are examined. As a result, one or more goal states containing an equal value as the minimum time value of all goal states are used to extract related schedule data. This output provides the optimal solution to the problem.

Evaluation

The previously described concept has been implemented in the *Java* programming language. This allows for evaluation and further experiments. As a first step, a small example scenario was created and applied to the prototype. In the following, the scenario is presented together with the experiment results.

The example scenario comprises four customers and two caregivers. One agent has the qualification level "1" and the other agent has the qualification level "3". In Table 1, all services for this example are listed. This data fragment was extracted from real-world data. The second column shows the corresponding durations in minutes and the third column shows the required qualification levels. In Table 2, the example order data is given. Each line shows one task

Table 1: Service data used in the example scenario.

S.-ID	Min.	Q.	Description
1	3	3	*Eye rinsing*
2	5	3	*Respiratory toilet*
3	3	2	*Glucose measurement*
4	2	2	*Injection of medication*
5	8	1	*Assistance with movements*
6	5	1	*Assistance with excretions*
7	5	1	*Assistance with bedding*
8	25	1	*Washing and dressing extended*
9	17	1	*Washing and dressing basic*

Table 2: Example order data.

Order-ID	Node	Services	Involved Q. Level(s)
1	4	{ 5, 6, 9 }	1
2	2	{ 1, 8 }	1, 3
3	1	{ 4, 6, 7 }	1, 2
4	3	{ 2, 3 }	2, 3

Table 3: Ten best solutions of the experiment.

Plan-ID	Rank	Processing Time	Splitting
1	1	54	1
2	1	54	1
3	1	54	1
4	2	63	0
5	2	63	0
6	2	63	0
7	2	63	0
8	3	65	2
9	3	65	2
10	4	71	1

bundle requested by a customer at a certain node on the graph g, which represents the environment. For simplification, the HHC office as starting location for all agents was set to node 4. Further, all edges $e \in E$ were assigned to the value "5". So, moving from one node to another node takes 5 minutes. Because some of the entries of Table 2 contains services with different qualification levels, these task bundles can be splitted in the planning process. The last column shows the involved qualification levels for each task bundle to clarify the relationships. The time value for the coordination task of a splitted task bundle was set to the constant value "1" as a simple example.

The application of the prototype generates 30 goal states. In Table 3, some results of the experiment are given. Each line shows a goal state with its related processing time in minutes in decreasing order. So, the ten best solutions are shown in the table and the first three entries contains the shortest processing time. Further, the information about using task bundle splitting in a solution plan is given by the last column. If one or more task bundles are splitted in a plan, the line shows the number of splitted task bundles in this column otherwise zero, which corresponds to a conventional solution method without splitting. In order to give more insight into the comparison, the schedules of the solutions 3, 4, and 9 are given in Table 4. Note that the schedule entries' time intervals include times for moving from one node to another node at the beginning of each interval. For example, the first entry of the schedule for solution plan 3 contains the processing of task bundle 4 while driving to the related node takes 5 minutes and rendering all services of this task bundle takes 8 minutes, which adds up to 13. During agent 2 accomplishes task bundle 4, agent 1 processes the frist part of the splitted task bundle 3, which continues until minute 15. After that, agent 1 has to wait three minutes until agent 2 arrives. This additional time can be used for unscheduled customer desires. When the second agent arrives, the joint coordination task takes one minute. Then, with the beginning of minute 19 the first agent moves on to the next node according to its schedule, while the second agent processes the second part of the splitted task bundle.

Table 4: Schedules of solution plan 3, 4, and 9.

Plan-ID	Agent-ID	Time	Order-ID
3	2	00 - 13	4
3	1	00 - 15	3_Part-1
3	2	13 - 21	3_Part-2
3	1	19 - 54	1
3	2	21 - 54	2
4	2	00 - 33	2
4	1	00 - 35	1
4	2	33 - 50	3
4	2	50 - 63	4
9	2	00 - 35	1
9	1	00 - 30	2_Part-1
9	2	35 - 44	2_Part-2
9	2	44 - 57	4
9	1	41 - 56	3_Part-1
9	2	57 - 65	3_Part-2

As shown in the result in Table 3, in this scenario an improvement of processing time in the amount of 14.3 percent can be achieved by using task bundle splitting (line 1-3) instead of the conventional solution method (line 4-7). The simple planning and scheduling algorithm works well for small scenarios, but takes too long for greater real-word scenarios. Nevertheless, the concept of task bundle splitting can be successful as shown above, so handling with greather real-world scenarios will be part of further work.

Future Work

As a next step, we will work on reducing search space as well as using technologies for increasing performance. The former will focus on applying heuristics to the concept. The latter will focus on methods using GPU computational power. Moreover, we will investigate how more general existing planning techniques can deal with this problem. For further evaluation, we have already gathered real-world data in oder to examine further steps of our concept with order data, travel times, and more service data taken from the real-world domain of HHC. In addition, comparing our prototype to state of the art temporal planners will be part of further evaluation steps. Moreover, we are working on integrat-

ing a standard *planning domain definition language* (PDDL) into our prototype as well as extending the concept regarding planning and scheduling for several time intervals and with respect to different time windows of customer orders.

One of the biggest next steps in the long term will be the extension of our concept to a dynamic runtime solution. In current operational management in the domain of HHC, delays in operational processes result in overtime hours of employees and potential time gains in these processes cannot be used to compensate for time delays with other employees. In addition, caregiver outages and unplanned urgent customer requests are possible in daily operations and make efforts for efficiency more difficult. Hence, low cost flexible adjustment of individual tasks or schedules for adaptively dealing with a dynamic environment is desirable. Especially multiagent technology is known for offering flexible solutions and adaptive IT systems (Kirn 2006). Moreover, knowledge and scheduling issues have a distributed structure among the participants and taking up-to-date local data of the real world into account can be necessary to achieve a proper planning result.

Dynamic Planning and Scheduling

To increase flexibility in caregivers' operations and efficiency in the use of resources, we further propose an agent-oriented framework for dynamic planning and scheduling, which will be described in the following. In Figure 2, the framework is depicted. Before the beginning of the day, initial planning and scheduling as presented in the previous sections provide the schedule Z. The connected database includes the current schedule and all information described before, e.g., customer orders for several time intervals. After computing an initial solution, this schedule can be modified by a dynamic planning and scheduling procedure. Especially during the service delivery process, the schedule Z will be modified to cope with a dynamic environment. For this purpose, the database provides required information during runtime as well, e.g., assigned qualification levels.

The inner HHC system components and their environment can be distinguished into real-world and virtual layer. Each real-world caregiver $c \in C$ is represented by a software agent in the virtual layer and is able to communicate with other agents. Using caregivers' mobile devices, a distributed structure can be established. During the service delivery process, each caregiver agent reacts on environmental-based planning disturbances like delays in service executions or travel times. If the further compliance with the own schedule segment is at risk, the agent tries to modify its schedule by searching alternative plans on its own as well as in combination with coordination and communication with other agents. Alternatively, a central re-planning is initiated. In addition, during the service delivery process, an agent checks several group-related task lists of new urgent customer orders and computes possible schedule modifications to include one or more new requests like every other agent does. The schedule modification with the lowest costs for the entire group of caregivers will be chosen. Also positive schedule deviations are used to reach a better joint solution. For example, there is a greater saving of time while rendering services at a customer's location, so the caregiver agent searches and compares alternative schedules under the new circumstances.

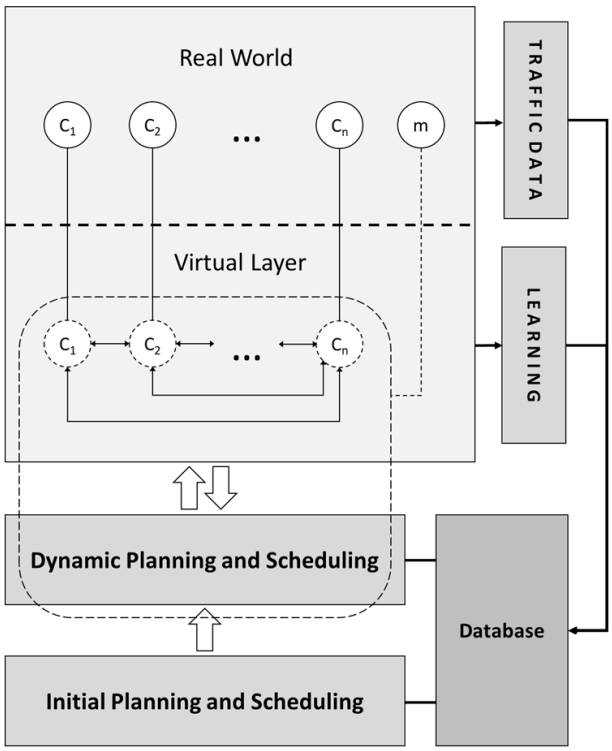


Figure 2: Agent-oriented Framework for Automated Dynamic Planning and Scheduling in HHC Management.

Caregivers of the real-world layer are continuously instructed with the next task bundle of the current schedule by their virtual agents using mobile devices. So, if something is changed in the background regarding scheduled tasks after the next task bundle, the caregiver does not have to worry about it, but simply continues to follow the instructions from one task bundle to the next.

Further, customers P, a road network including traffic, and the operational manager are parts of the environment of the real world. In Figure 2, the latter is referred to as m and is capable of influencing the coordination between the caregivers. During the service delivery process, the manager filters new short-term customer requests and adds urgent requests to the group-related task lists mentioned before. Usually, customers with urgent medical issues call the HHC provider's office and the operational manager decides what to do. For every shift $y \in Y$ of the current day, a group-related task list containing new urgent customer orders exists and the lists are checked by the caregiver agents in order to assign new entries during runtime.

Furthermore, caregiver outages during the service delivery process are possible, e.g., car accidents or private emergencies of employees, but the medical care of customers have to be ensured. To this end, an affected caregiver can

use his or her mobile device to announce the outage and the virtual representative handles the allocation of his or her customer orders to the remaining caregivers. If the outage is announced to the office before starting the service delivery process the operational manager will just invoke the initial scheduling algorithm again.

Using Advanced Data and Learning Mechanisms

Besides the data described previously, **further data** is necessary for planning and scheduling in order to generate better results in the long run. Initially, each service $s \in S$ is assigned to a time value which is required for basic scheduling issues. Furthermore, constraints based on different relationships between customers and caregivers exists. For instance, a female customer only wants to be treated by a female caregiver or a caregiver does not want to treat a specific person. Maintaining a long-term assignment of a caregiver to a customer instead of having alternating caregivers might also be in customer's interest which could increase scheduling effort. In addition, some caregivers do not want to perform certain services even though they have the appropriate qualification level. The reasons for this may vary, such as uncertainty due to lack of experience or physical aptitude. Beyond that, there are legal requirements for various aspects like break time specifications which are available in the database and must be taken into account in the scheduling process.

Regarding spatial aspects, the HHC office and all customer locations form a structure of nodes and weighted edges, which was introduced as the graph g. In this sense, static travel time matrices for different hours of a day are also stored in the database and they will be used for initial scheduling. During service delivery process, the traffic data module as shown in Figure 2 requests public traffic data for each edge using different real-world sources and updates edge weights at short time periods. Also the static travel time matrices will be updated periodically by the traffic data module. Traffic data and corresponding route data will be queried by caregiver agents during runtime for application in searching scheduling alternatives and in order to keep to the current schedule.

At runtime, different **learning mechanisms** working on the virtual layer generating additional data and update existing values in the database. Close to the stored traffic data, deviations related to certain routes are learned from caregiver agents' movement in the real world. For example, *reinforcement learning* can be applied to allow for a better routing in terms of cargiver's movement from one node to another using travel times for feedback information to the respective caregiver agent. Further, a value for deviations in service execution at customer's location is learned for each customer using automated documentation data from caregivers' mobile devices. Because recorded documentation times refer to entire customer visits instead of single service executions, a learned value is assigned to a set of services (task bundle). It is not uncommon for certain task bundles to be repetitively requested by a customer on a daily or weekly basis. The learned values can be used for other scheduling processes to obtain better planning results over time. With the approval of employees, these planning values can also be extended to include caregivers performing service execution. As a result, more differentiated values are available for planning and scheduling for each agent.

Conclusion and Outlook

Increasing demand in the domain of home health care as well as a shortage of professionals faces the operational management with challenges regarding usage of limited resources and increasing efficiency while taking human needs and desires into account. For this reason, improvements of planning and scheduling issues in the domain of HHC are desirable. As a first step, we introduced a concept of splitting task bundles in temporal planning for cooperative agents with different qualifications. By applying this concept, concurrent processing of several task bundles can be improved due to better usage of limited resources as shown in an example scenario.

Future work will focus on further investigating what can *AI Planning* do for the described problem. Using this knowledge, we will extend our concept and reduce search space by applying heuristics. Furthermore, using real-world data and comparing our prototype to state of the art temporal planners will be part of next evaluation steps. In the long term, we aim at developing a dynamic planning and scheduling approach including the presented concept in order to increase efficiency in operational processes.

References

Becker, C. A.; Lorig, F.; and Timm, I. J. 2019. Multiagent Systems to Support Planning and Scheduling in Home Health Care Management: A Literature Review. In Koch et al., ed., *Artificial Intelligence in Health*, 13–28. Springer International Publishing.

Kirn, S. 2006. Flexibility of Multiagent Systems. In Kirn, S.; Herzog, O.; Lockemann, P.; and Spaniol, O., eds., *Multiagent Engineering: Theory and Applications in Enterprises*. Springer Berlin Heidelberg. 53–69.

López-Santana, E. R.; Espejo-Díaz, J. A.; and Méndez-Giraldo, G. A. 2016. Multi-agent Approach for Solving the Dynamic Home Health Care Routing Problem. In *Workshop on Engineering Applications*, 188–200. Springer.

Marcon, E.; Chaabane, S.; Sallez, Y.; Bonte, T.; and Trentesaux, D. 2017. A Multi-Agent System Based on Reactive Decision Rules for Solving the Caregiver Routing Problem in Home Health Care. *Simulation Modelling Practice and Theory* 74:134–151.

Rothgang, H.; Kalwitzki, T.; Unger, R.; and Amsbeck, H. 2016. Pflege in Deutschland im Jahr 2030 - regionale Verteilung und Herausforderungen LebensWerte Kommune. *Gütersloh: Bertelsmann Stiftung*.

Torreño, A.; Onaindia, E.; Komenda, A.; and Štolba, M. 2018. Cooperative Multi-Agent Planning: A Survey. *ACM Computing Surveys (CSUR)* 50(6):84.

Xie, Z., and Wang, C. 2017. A Periodic Repair Algorithm for Dynamic Scheduling in Home Health Care Using Agent-Based Model. 245–250. IEEE.

Learning-based Preference Prediction for Constrained Multi-Criteria Path-Planning

Kevin Osanlou[1,3], **Christophe Guettier**[1], **Andrei Bursuc**[2], **Tristan Cazenave**[3], **Eric Jacopin**[4],
[1] Safran Electronics & Defense
[2] valeo.ai
[3] LAMSADE, Université Paris-Dauphine
[4] CREC, Ecoles de Saint-Cyr Coëtquidan
kevin.osanlou@safrangroup.com christophe.guettier@safrangroup.com andrei.bursuc@valeo.com
tristan.cazenave@lamsade.dauphine.fr eric.jacopin@st-cyr.terre-net.defense.gouv.fr

Abstract

Learning-based methods are increasingly popular for search algorithms in single-criterion optimization problems. In contrast, for multiple-criteria optimization there are significantly fewer approaches despite the existence of numerous applications. Constrained path-planning for Autonomous Ground Vehicles (AGV) is one such application, where an AGV is typically deployed in disaster relief or search and rescue applications in off-road environments. The agent can be faced with the following dilemma : optimize a source-destination path according to a known criterion and an uncertain criterion under operational constraints. The known criterion is associated to the cost of the path, representing the distance. The uncertain criterion represents the feasibility of driving through the path without requiring human intervention. It depends on various external parameters such as the physics of the vehicle, the state of the explored terrains or weather conditions. In this work, we leverage knowledge acquired through offline simulations by training a neural network model to predict the uncertain criterion. We integrate this model inside a path-planner which can solve problems online. Finally, we conduct experiments on realistic AGV scenarios which illustrate that the proposed framework requires human intervention less frequently, trading for a limited increase in the path distance.

1 Introduction

Operations carried out by an autonomous ground vehicle (AGV) are constrained by terrain structure, observation abilities, embedded resources. For instance, in disaster relief operations or area surveillance, maneuvers must consider terrain knowledge. In most cases, the AGV ability to maneuver in its environment has direct impact on operational efficiency. Several perception capabilities (online mapping, geolocation, optronics, LIDAR) enable it to update its environment awareness online. Different mission planning layers can then provide continued navigation plans which are used for controlling the robotic platform, enabling it to fulfill a set mission. The AGV automatically manages its trajectory and follows navigation waypoints using control and time sequence algorithms.

Such mission planners could integrate A* algorithms (Hart, Nilsson, and Raphael 1968) as a best-first search approach in the space of available paths. For a complete overview of static algorithms (*e.g.*, A*), replanning algorithms (*e.g.*, D*), anytime algorithms (*e.g.*, ARA*), and anytime replanning algorithms (*e.g.*, AD*), we refer the reader to (Ferguson, Likhachev, and Stentz 2005). Algorithms stemming from A* can handle some heuristic metrics but can become complex to develop when dealing with several constraints simultaneously like mandatory waypoints and distance metrics, or several optimization criteria. Our approach is based on Constraint Programming (CP). CP provides a powerful baseline to model and solve combinatorial and / or constraint satisfaction problems (CSP). It has been introduced in the late 70s (Laurière 1978) and has been developed until now (Hentenryck, Saraswat, and Deville 1998; Ajili and Wallace 2004; Carlsson 2015), with several real-world autonomous system applications, in space (Bornschlegl, Guettier, and Poncet 2000; Simonin et al. 2015), aeronautics (Guettier and Lucas 2016) and defense (Goldman et al. 2002).

Nevertheless, difficult weather and off-road conditions can make it impossible for an AGV to proceed through some paths autonomously. The intervention of a human, either onboard or with a remote control system, would be required in such situations. However, the purpose of an AGV is to reduce the crew workload in the first place. Coming up with a navigation plan which requires minimum human intervention and remains acceptable in terms of a metric such as the total distance is therefore critical. This work focuses on this particular issue, especially in a context where the decision criterion for human intervention is uncertain and needs to be learned offline. We refer to this uncertain criterion as *autonomous feasibility*. In our approach, we learn this criterion to assess where human intervention would be needed.

To this purpose, we use a simulated AGV environment that includes a decision function for this criterion as part of a higher-level environment representation. Computing the autonomous feasibility by running this function for online mission planning would require simulating the entire environment. This is impractical for real-time applications. Instead, we train a neural network to approximate this criterion offline. When planning the mission online, we use the neural network to make predictions for the autonomous feasibility. We define an optimization strategy for a CP-based planner and use it to compute navigation plans. We conduct

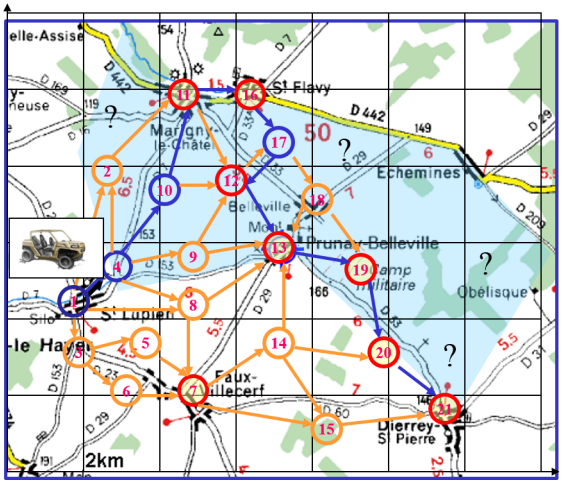


Figure 1: Search and rescue mission. Potential paths for an AGV throughout a flooded area between the Seine and the Vanne rivers in the area of Troyes, France. The blue area represents the expected flooded area. Orange, red and blue circles represent different positions the AGV can proceed to. Red circles are possible waypoints while blue circles are mandatory ones. Orange edges represent trafficability and blue ones a potential optimal solution in terms of distance cost.

experiments on realistic scenarios and show that the proposed framework reduces need for human interactions, trading with an acceptable increase of the total path distance.

2 Context and Problem Formalization

AGV operations require online path-planning, done accordingly with the demands of an operator who defines the objectives of a mission. For instance, in disaster relief, the vehicle has to perform some reconnaissance tasks in specific areas. Figure 1 shows a flooded area and possible paths to assess disaster damages. Possible paths are defined using a graph representation, where edges and nodes represent respectively ground mobility and accessible waypoints. Graphs are defined during mission preparation by terrain analysis and situation assessment. The AGV system responds to an operator demand by finding a route from a starting point to a destination, and by maneuvering through mandatory waypoints. During the mission, some paths may not be trafficable or new flooded areas to investigate may be identified, requiring immediate replanning. Additionally, some paths may prove difficult for the AGV given weather and road conditions and require the crew to take control of the vehicle to proceed forward. Aside from the terrain structure itself, these issues are mainly caused by changes in weather conditions:

- Rainfall: heavy rain may cause new muddy areas to occur, affecting cross-over duration between two waypoints or increasing the risk of losing platform control.
- Fog: heavy fog may cause a performance drop of the LIDAR, making it harder for the AGV to keep track of the surrounding environment.
- Wind: heavy winds may require a high level of corrections in the followed trajectories.

In Figure 1, the AGV starts from its initial position (position 1). Areas in blue are flooded and the disaster perimeter must be evaluated by the vehicle. All nodes circled in red have to be visited, *e.g.*, refugees and casualties are likely to be found there. A typical damage assessment would require up to 10 mandatory nodes to visit. The first criterion is the global traverse distance that meets all visit objectives, and must be minimized. The second criterion is the autonomous feasibility, which needs to be maximized so as to reduce the likelihood of requiring human intervention.

Let $\mathcal{G} = (V, E)$ be a connected weighted graph. Each edge has a weight representing a distance metric. In addition to this weight, a binary feature represents the autonomous feasibility of the edge, which is not known a priori and depends on both weather conditions and terrain structure. While terrain structure mostly depends on the graph $\mathcal{G}$, the weather affecting the AGV is defined by three variables:

- $x_1 \in \{0, 1, ..., 10\}$ is the intensity of rainfall,
- $x_2 \in \{0, 1, ..., 10\}$ is the thickness of the fog,
- $x_3 \in \{0, 1, ..., 10\}$ is the strength of the wind

A typical instance I of the path-planning problem we consider is defined as follows:

$$I = (s, d, M)$$

where:

- $s \in V$ is the start node in graph $\mathcal{G}$,
- $d \in V$ is the destination node in graph $\mathcal{G}$,
- $M \subset V$ is a set of distinct mandatory nodes that need to be visited at least once, regardless of the order of visit.

In order to solve instance I, the AGV has to find a path from node s to node d that passes by each node in M at least once. There is no limit to how many times a node can be visited in a path. The solution path should compromise between minimizing the total path distance and maximizing the autonomous feasibility.

3 Environment Simulation

The environment in which the AGV evolves is modelled in a 3D map. Vertices defined in $\mathcal{G}$ represent positions in the environment, while edges are represented by a set of continuous sub-positions linking vertices. Figure 2 shows a typical environment simulated by the 4d Virtualiz software in which the AGV proceeds. The simulation is realistic as it takes into account not only vehicle physics, but more importantly the vehicle's sensors, as well as core programs. Among such programs lie the environment-building functions, which enable the vehicle to build a state of its surrounding environment from its sensors. Key functions such as obstacle avoidance or waypoint-follow functions are also implemented in the simulator mimicking real life situations with high fidelity. We leverage the simulated environment to design and test out our autonomous system in scenarios similar to real life conditions.

Figure 2: An AGV evolving in an environment created by the 4d Virtualiz simulator.

When a graph is defined, the simulator links it to the 3D map and several built-in functionalities become available. We can thus generate custom missions for the AGV by defining a start position, an end position, and a list of mandatory waypoints. Additional environment parameters are available in such simulators and we consider a few representative ones in this work. In particular, we experiment with the rain variable x_1, the fog variable x_2 and the wind variable x_3. When sending the AGV on a defined mission $I = (s, d, M)$ taking a path P, the simulating system will make the vehicle drive on a set of edges $e \in P$. Edges resulting in autonomous failure (vehicle getting stuck or taking longer than a set timeout threshold) $Q \subset P$ correspond to difficult sections of the path, which would require manual control of the vehicle. The criteria for autonomous feasibility depends on both current weather conditions and terrain structure and topology, as well as the vehicle's autonomous capabilities.

In order to learn to approximate the autonomous feasibility criterion, we use this simulated environment to create training data. To this end, we generate different weather conditions by changing the variables x_1, x_2, x_3 and send the vehicle on random missions. For a given set of values x_1, x_2, x_3 and a path P that the vehicle has to follow, we retrieve the list of edges $Q \subset P$ which the simulator judges difficult for autonomous maneuvers. For each edge $e_i \in Q$, we store in a dataset D a feature vector $\mathbf{x}_i = [x_1, x_2, x_3, \max(\text{slope}_i), \text{dist}_i]$ and $y_i = 0$. Variable $\max(\text{slope}_i)$ is the maximum slope of edge e_i, dist_i the distance of edge e_i and y_i is the preference label associated with $\mathbf{x}_i$. With $y_i = 0$, the edge should be avoided if possible when planning under these weather conditions. Similarly for each edge $e_i \in P \backslash Q$, we store the value $\mathbf{x}_i$ and $y_i = 1$. These edges should be preferred when planning under these conditions. Regarding the structure of the terrain, we are unable to retrieve more information for an edge than only its distance and maximum slope. While this lack of information results in a limitation for learning performance, our experiments show that the performance of our framework remains satisfying.

4 Neural Network Training

In this section, we first provide a brief introduction to neural networks and then describe the manner we leverage them for our problem. We train the neural network for identifying the difficult areas on the map using terrain and weather information and supervision from the decisions of the simulator concerning the respective sections on the map.

Neural networks (NNs) allow computing and learning of multiple levels of abstraction of data through models with millions of trainable parameters. It is known that a sufficiently large neural network can approximate any continuous function (Funahashi 1989). Although the cost of training such a network can be prohibitive, modern training practices for multi-layer networks usually allow reasonable approximations for a large variety of problems. With this in mind, we attempt to train a neural network to approximate the decisions of the 3D simulator over edges of the graph map.

4.1 Neural Networks

Recently NNs, in particular deep neural networks, made a comeback in the research spotlight after achieving major breakthroughs in various areas of computer vision (Krizhevsky, Sutskever, and Hinton 2012), (Simonyan and Zisserman 2014), (He et al. 2016), (Ren et al. 2015), (Redmon et al. 2016), (Long, Shelhamer, and Darrell 2015), neural machine translation (Sutskever, Vinyals, and Le 2014), computer games (Silver et al. 2016) and many more fields. While the fundamental principles of training neural networks are known since many years, the recent improvements are due to a mix of availability of large image datasets, advances in GPU-based computation and increased shared community effort.

In spite of the complex structure of a NN, the main mechanism is rather straightforward. A *feedforward neural network*, or *multi-layer perceptron (MLP)*, with L layers describes a function $f(\mathbf{x}; \boldsymbol{\theta}) : \mathbb{R}^{d_\mathbf{x}} \mapsto \mathbb{R}^{d_y}$ that maps an input vector $\mathbf{x} \in \mathbb{R}^{d_\mathbf{x}}$ to an output vector or scalar value $y \in \mathbb{R}^{d_y}$. Vector $\mathbf{x}$ is the input data that we need to analyze (*e.g.*, an image, a graph, a feature vector, etc.), while $\mathbf{y}$ is the expected decision from the NN (*e.g.*, a class index, a scalar, a heatmap, etc.). The function f performs L successive operations over the input $\mathbf{x}$:

$$h^{(l)} = f^{(l)}(h^{(l-1)}; \theta^{(l)}) = \sigma\left(\theta^{(l)} h^{(l-1)} + b^{(l)}\right) \qquad (1)$$

where $h^{(l)}$ is the hidden state of the network and $f^{(l)}$ is the mapping function performed at layer l and parameterized by trainable parameters $\theta^{(l)}$ and bias $b^{(l)}$, and piecewise activation function $\sigma(\cdot)$; $h^{(0)} = \mathbf{x}$. We denote by $\boldsymbol{\theta} = \{\theta^{(1)}, \ldots, \theta^{(L)}\}$ the entire set of parameters of the network. Intermediate layers are actually a combination of linear classifiers followed by a piece-wise non-linearity from the activation function. Layers with this form are termed *fully-connected layers*.

NNs are typically trained with labeled training data, *i.e.* a set of input-output pairs $(\mathbf{x}_i, y_i)$, $i = 1, \ldots, N$, where N is the size of the training set. During training we aim to minimize the training loss:

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \ell(\hat{y}_i, y_i), \qquad (2)$$

where $\hat{y_i} = f(\mathbf{x}_i; \boldsymbol{\theta})$ is the estimation of y_i by the NN and $\ell : \mathbb{R}^{d_L} \times \mathbb{R}^{d_L} \mapsto \mathbb{R}$ is the loss function. The loss ℓ measures the distance between the true label y_i and the estimated one $\hat{y_i}$. Through *backpropagation* (Rumelhart et al. 1988), the information from the loss is transmitted to all $\boldsymbol{\theta}$ and gradients of each θ_l are computed w.r.t. the loss ℓ. The optimal values of the parameters $\boldsymbol{\theta}$ are then found via stochastic gradient descent (SGD) which updates $\boldsymbol{\theta}$ iteratively towards the minimization of $\mathcal{L}$. The input data is randomly grouped into mini-batches and parameters are updated after each pass. The entire dataset is passed through the network multiple times and the parameters are updated after each pass until reaching a satisfactory optimum.

4.2 Training Setup

We define a neural network f that takes as input a vector $\mathbf{x}_i = [x_1, x_2, x_3, \max(\text{slope}_i), \text{dist}_i]$ and outputs the probability $\hat{y}$ of the edge e_i being a preferred edge for autonomous navigation or not. The network f consists of 4 fully-connected layers interleaved with *ReLU* non-linearities and with a $sigmoid$ activation at the end. The output of the sigmoid $\in [0, 1]$ is rounded to the closest integer 0 or 1 when classifying an edge under given weather and terrain conditions.

The simulator serves as *teacher* to the NN, which learns here to mimic the simulator's decisions based on the path configuration and weather. Following the creation of dataset D in section (§ 3) we use the pairs of edges and labels $(\mathbf{x}_i, y_i)$ to train the neural network to correctly predict the label node $\hat{y}_i$. For supervision we use the binary cross-entropy loss, typically used for binary classification tasks:

$$\ell(\hat{y}_i, y_i) = -[y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y_i})] \quad (3)$$

We train the NN using SGD with momentum. In order to prevent overfitting, we do *early stopping*, *i.e.*, we halt the training once the average loss on the validation set stops decreasing and starts increasing. In our experiments, the neural network f achieves an accuracy of 79% on the validation set. This is due to the lack of features which come into play in deciding the autonomous feasibility for an edge. In section (§6), this performance is tested and evaluated on new situations. In comparison, we also ran a logistic regression on the same training set, which achieved a validation accuracy of 71%. We believe some feature engineering may be necessary to provide slightly more relevant features for the logistic regression.

5 Constrained Multi-Criteria Optimization for Navigation and Maneuver Planning

We aim to compute an optimized navigation plan in cross-country areas. In our approach, the navigation plan is represented as a path sequence of waypoints in a predefined graph. A "good" plan must minimize distance and maximize autonomous feasibility while satisfying mandatory waypoints.

Path planning is achieved using Constraint Programming (CP), here with a model-based constraint solving approach (Guettier and Lucas 2016) and cost objective functions (equations 7 and 8). The problem is formulated in CP as a Constraint Optimization Problem (COP). Distance and autonomous feasibility are considered as primary and secondary cost objectives, respectively. We propose a multi-criteria optimization algorithm, based on global search, and adapted from branch and bound (B&B) techniques (Narendra and Fukunaga 1977).

Both COP formulation and search techniques are implemented with the CLP(FD) domain of SICStus Prolog library (Carlsson 2015). It uses the state-of-the-art in discrete constrained optimization techniques and Arc Consistency-5 (AC-5) (Deville and Van Hentenryck 1991; Van Hentenryck, Deville, and Teng 1992) for constraint propagation, implemented as CLP(FD) predicates.

The search technique is hybridized with a probing method (Guettier and Lucas 2016), allowing automatic structuring of the global search tree. In this paper, probing focuses on learned and predicted autonomous feasibility in order to define an upper bound to the secondary metric. Probing takes as input predicted autonomous feasibility, builds up a heuristic sub-optimal path based on it as a preference, and lastly initializes the secondary cost criterion. The resulting algorithm is a Probe-based Constraint Multi-Criteria Optimizer denoted as PCMCO.

5.1 Planning Model with Flow Constraints for Multi-Criteria Optimization

The PCMCO elaborates a classical flow formulation with integrals, widely used in operation research (Gondran and Minoux 1995). For a given path-planning problem $I = (s, d, M)$, the set of possible paths is modelled as a graph $\mathcal{G} = (V, E)$, where V is the set of vertices and E the set of elementary paths between vertices. A set of flow variables $\varphi_e \in \{0, 1\}$, where $e \in E$, models a possible path from $start \in V$ to $end \in V$. A flow variable for an edge $e = (v, v')$ is denoted as $\varphi_{vv'}$. An edge e belongs to the navigation plan if and only if $\varphi_e = 1$. The resulting navigation plan is represented as $\Phi = \{e |\ e \in E,\ \varphi_e = 1\}$. From an initial position to a requested final one, path consistency is enforced by flow conservation equations, where $\omega^+(v) \subset E$ and $\omega^-(v) \subset E$ represent respectively outgoing and incoming edges from vertex $v \in V$. Since flow variables are $\{0, 1\}$, equation (4) ensures path connectivity and uniqueness while equation (5) imposes limit conditions for starting the path at s and ending it at d:

$$\sum_{e \in \omega^+(v)} \varphi_e = \sum_{e \in \omega^-(v)} \varphi_e \leq N \quad (4)$$

$$\sum_{e \in \omega^+(s)} \varphi_e = 1, \quad \sum_{e \in \omega^-(d)} \varphi_e = 1, \quad (5)$$

These constraints provide a linear chain alternating pass-by waypoint and navigation along the graph edges. Constant N indicates the maximum number of times the vehicle can pass by a waypoint. With this formulation, the plan may contain cycles over several waypoints. Mandatory waypoints are imposed using constraint (6). Path length is given by the metric (7), and we will consider the path length as

the primary optimization D_{end} criterion to minimize, where constants $d_{vv'}$ represent elementary path distance between vertices. They are provided off-line, at mission preparation time. Likewise, the secondary criterion P_{end} has the same formulation and is based on autonomous feasibility (8). In turn, constants $p_{vv'}$ are edge preferences resulting from the predictions of the neural network f on autonomous feasibility for each edge $e \in E$.

$$\forall v \in M \sum_{e \in \omega^+(v)} \varphi_e \geq 1 \tag{6}$$

$$\forall v \in V, D_v = \sum_{v'v \in \omega^-(v)} \varphi_{v'v} d_{vv'} \tag{7}$$

$$\forall v \in V, P_v = \sum_{v'v \in \omega^-(v)} \varphi_{v'v} p_{vv'} \tag{8}$$

5.2 Global Search Algorithm

The global search technique underlying PCMCO guarantees completeness, as well as proof of completeness. It is based on classical algorithmic components:

- Variable filtering with correct values, using specific labeling predicates to instantiate problem domain variables. AC-5 being incomplete, value filtering guarantees search completeness.
- Tree search with standard backtracking when instantiating a variable fails.
- Branch and Bound (B&B) for both primary and secondary cost optimization, using minimize predicate.

Within the B&B algorithm, the primary cost D_{end} drives the optimization loop. We extend the algorithm with preference optimization P_{end} to converge towards a pareto optimal solution. At each iteration k, we impose that $P_{end}^{k+1} \leq P_{end}^{k}$ as a secondary optimization schema. This constraint is weaker than $D_{end}^{k+1} < D_{end}^{k}$, classically applied to the primary distance cost, which corresponds to the default operational semantic predicate *minimize* of the SICStus Prolog library. The P_{end}^{0} is initialized by probing with an arbitrary heuristic solution obtained with the Dijsktra algorithm. In this manner, B&B will favor learned preferences.

Note that in general probing techniques (Sakkout and Wallace 2000), the order can be redefined within the search structure (Ruml 2001). Similarly, in our 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. Mainly, first fail variable selection is used in addition to the initial probing order.

These algorithmic designs have already been reported with different probing heuristics (Guettier and Lucas 2016), such as A* or meta-heuristics such as Ant Colony Optimization (Lucas et al. 2010),(Lucas, Guettier, and Siarry 2009). However, other multi-criteria optimization techniques could be used, for instance based on valued constraint satisfaction problems (VCSP) (Schiex et al. 1995) or soft constraints (Domshlak et al. 2003). In our design, the search is still complete, guaranteeing proof of completeness, but demonstrates efficient pruning.

6 Experiments

For a given problem, minimizing the total distance of a solution path while maximizing autonomous feasibility are contradictory objectives requiring a compromise. This section carries two purposes. The first is to verify that the neural network f is capable of making consistent predictions to avoid difficult edges. The second is to evaluate the compromise made by the CP-based solver described previously.

We generate 200 random benchmark instances associated with a graph $\mathcal{G}$ (Guettier 2007) that is representative of real scenarios for AGV search & rescue operations. We consider three different types of weather conditions: *fine*, *moderate* and *difficult*. For each weather type, we randomly select 50 instances, and we compare the solutions given by two solvers. The first solver is the reference probe-based constrained optimizer (PCO), which does not explore any preference criterion and only optimizes the distance. The second solver is the upgraded version with multi-criteria optimization (denoted as PCMCO). It takes into account the preference predictions of the neural network f for current weather conditions. For each edge $e_i \in E$, the preference prediction is obtained with a forward pass of the feature vector described in section (§4). We denote the resulting hybridization as NN + PCMCO.

Table 1: Experiments carried out on benchmark instances of graph $\mathcal{G}$. The first metric reported is the distance of the solution path, in meters. The second one is the number of human interventions required in the solution path. For both metrics, we compute the mean, median (med) and standard deviation (std) over all benchmark instances.

Weather & Method:	Distance (m)			Interventions		
	mean	med	std	mean	med	std
Fine weather						
PCO	4463	4246	840	1.6	2	1.1
NN + PCMCO	4912	5016	954	0.1	0	0.3
Moderate weather						
PCO	4166	4102	675	2.3	2	1.2
NN + PCMCO	5431	5390	1003	0.4	0	0.6
Difficult weather						
PCO	4207	4115	687	4.1	4	1.2
NN + PCMCO	5153	5256	881	2.5	2	1.4

We study the influence of the edge preferences given by the neural network f on the solution path. For each instance, we compute the solution path given by each solver. The total distance is then computed by summing the distances of all edges in the solution path. The solution path is also simulated in the 3D simulation environment to count the number of required human interventions. The human intervention count used in this section is a criterion which is opposite to the autonomous feasibility criterion, and should therefore be minimized. Results are averaged per instance and reported in table 1.

For fine weather conditions, the use of the neural network

preferences enables NN + PCMCO to almost never require human assistance in exchange for a 10% higher distance cost than PCO's. On the other hand, PCO requires more than 1 human intervention per instance on average. For moderate weather conditions, we see those gaps widening. A 30% higher distance cost allows NN + PCMCO to require far less human interventions than PCO. Lastly, for difficult weather conditions, we observe that NN + PCMCO incurs a 22% higher distance cost. While NN + PCMCO requires far less human interventions than PCO, it still requires more than 2 human interventions on average. This is explained by the fact that difficult weather conditions cause a majority of edges to be difficult for autonomous driving. The solution path has no choice but to include some of those edges. This also explains the lower distance cost increase than for moderate weather conditions.

Additionally, we run statistical tests to compare PCO and NN+PCMCO and summarize them in table 2. Firstly, a paired sample t-test is done, for each weather condition, to compare the mean path distance given by PCO and NN+PCMCO. The high t-values obtained, combined with very low p-values, indicate that the distance costs found by PCO and NN+PCMCO differ significantly and that it is very unlikely to be due to coincidence. Secondly, a $\tilde{\chi}^2$ test is performed on the intervention count criterion for each weather condition. The high p-values observed validate the hypothesis that NN+PCMCO acts independently of PCO in terms of autonomous feasibility.

Table 2: Statistical tests run on benchmark results. The paired sample t-test is run on the distance criterion, while the $\tilde{\chi}^2$ test is run on the intervention count criterion.

Test Method	Paired t-test		$\tilde{\chi}^2$ test	
	t-value	p-value	$\tilde{\chi}^2$-value	p-value
Fine weather	7.28	10^{-9}	19.1	0.99
Moderate weather	8.18	10^{-9}	19.4	0.89
Difficult weather	6.51	10^{-7}	9.37	0.99

These results highlight the fact that the neural network f makes consistent predictions, and that NN + PCMCO offers a good compromise between distance metric and autonomous feasibility.

7 Conclusion

We introduced a method for online constrained path-planning problems with two optimization criteria, based on learned preferences. The distance criterion needs to be minimized, while the autonomous feasibility criterion, which is uncertain, has to be maximized. Our approach proposes offline learning of a model for autonomous feasibility in simulation environments. We also introduced a CP-based algorithm which takes into account the model's prediction of autonomous feasibility and compromises between both criteria for online path-planning. Experiments suggest the proposed framework is capable of finding a good compromise which offers a higher autonomous feasibility for an acceptable increase in distance cost. AGV crew could benefit from such an approach, mostly in situations where their workload needs to be reduced.

References

Ajili, F., and Wallace, M. 2004. Constraint and integer programming : toward a unified methodology. *Operations research/computer science interfaces series*.

Bornschlegl, E.; Guettier, C.; and Poncet, J.-C. 2000. Automatic planning for autonomous spacecraft constellations. In *Proceedings of the 2nd International NASA Workshop on Planning and Scheduling for Space*.

Carlsson, M. 2015. *SICSTUS Prolog user's manual*.

Deville, Y., and Van Hentenryck, P. 1991. An efficient arc consistency algorithm for a class of csp problems. In *Proceedings of the 12th International Joint Conference on Artificial intelligence (IJCAI)*, volume 1, 325–330.

Domshlak, C.; Rossi, F.; Venable, K. B.; and Walsh, T. 2003. Reasoning about soft constraints and conditional preferences: complexity results and approximation techniques. In *IJCAI*.

Ferguson, D.; Likhachev, M.; and Stentz, A. T. 2005. A guide to heuristic-based path planning. In *Proceedings of the International Workshop on Planning under Uncertainty for Autonomous Systems, International Conference on Automated Planning and Scheduling (ICAPS)*.

Funahashi, K.-I. 1989. On the approximate realization of continuous mappings by neural networks. *Neural networks* 2(3):183–192.

Goldman, R.; Haigh, K.; Musliner, D.; and Pelican, M. 2002. Macbeth: A multi-agent constraint-based planner. In *Proceedings of the 21st Digital Avionics Systems Conference*, volume 2, 7E3:1–8.

Gondran, M., and Minoux, M. 1995. Graphes et algorithmes.

Guettier, C., and Lucas, F. 2016. A constraint-based approach for planning unmanned aerial vehicle activities. *The Knowledge Engineering Review* 31(5):486–497.

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.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

Hentenryck, P. V.; Saraswat, V. A.; and Deville, Y. 1998. Design, implementation, and evaluation of the constraint language cc(fd). *J. Log. Program.* 37(1-3):139–164.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural net-

works. In *Advances in neural information processing systems*, 1097–1105.

Laurière, J.-L. 1978. A language and a program for stating and solving combinatorial problems. *Artificial Intelligence* 10:29–127.

Long, J.; Shelhamer, E.; and Darrell, T. 2015. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 3431–3440.

Lucas, F.; Guettier, C.; Siarry, P.; de La Fortelle, A.; and Milcent, A.-M. 2010. Constrained navigation with mandatory waypoints in uncertain environment. *International Journal of Information Sciences and Computer Engineering (IJISCE)* 1:75–85.

Lucas, F.; Guettier, C.; and Siarry, P. 2009. Hybridisation of constraint solving with an ant colony algorithm for on-line vehicle path planning. In *Proceedings of the 4th Workshop on Planning and Plan Execution for Real-World Systems, ICAPS'09*.

Narendra, P. M., and Fukunaga, K. 1977. A branch and bound algorithm for feature subset selection. *IEEE Transactions on Computers* C-26:917–922.

Redmon, J.; Divvala, S.; Girshick, R.; and Farhadi, A. 2016. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 779–788.

Ren, S.; He, K.; Girshick, R.; and Sun, J. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, 91–99.

Rumelhart, D. E.; Hinton, G. E.; Williams, R. J.; et al. 1988. Learning representations by back-propagating errors. *Cognitive modeling* 5(3):1.

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.

Schiex, T.; Fargier, H.; Verfaillie, G.; et al. 1995. Valued constraint satisfaction problems: Hard and easy problems. *IJCAI (1)* 95:631–639.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of go with deep neural networks and tree search. *nature* 529(7587):484–489.

Simonin, G.; Artigues, C.; Hebrard, E.; and Lopez, P. 2015. Scheduling scientific experiments for comet exploration. *Constraints* 20:77–99.

Simonyan, K., and Zisserman, A. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Sutskever, I.; Vinyals, O.; and Le, Q. V. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, 3104–3112.

Van Hentenryck, P.; Deville, Y.; and Teng, C. 1992. A generic arc-consistency algorithm and its specializations. *Artificial Intelligence* 57:291–321.

The Value of Incorporating Social Preferences in Dynamic Ridesharing

Sandhya Saisubramanian[1] **Connor Basich**[1] **Shlomo Zilberstein**[1] **Claudia V. Goldman**[2]
[1]College of Information and Computer Sciences, University of Massachusetts Amherst, USA
[2]General Motors, Advanced Technical Center, Israel
{saisubramanian, cbasich, shlomo}@cs.umass.edu, claudia.goldman@gm.com

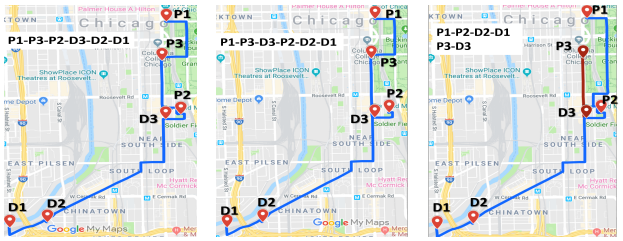
Abstract

Dynamic ridesharing services (DRS) play a major role in improving the efficiency of urban transportation. User satisfaction in dynamic ridesharing is determined by multiple factors such as travel time, cost, and social compatibility with co-passengers. Existing DRS optimize profit by maximizing the operational value for service providers or minimize travel time for users but they neglect the social experience of riders, which significantly influences the total value of the service to users. We propose DROPS, a dynamic ridesharing framework that factors the riders' *social preferences* in the matching process so as to improve the quality of the trips formed. Scheduling trips for users is a multi-objective optimization that aims to maximize the operational value for the service provider, while simultaneously maximizing the value of the trip for the users. The user value is estimated based on compatibility between co-passengers and the ride time. We then present a real-time matching algorithm for trip formation. Finally, we evaluate our approach empirically using real-world taxi trips data, and a population model including social preferences based on user surveys. The results demonstrate improvement in riders' social compatibility, without significantly affecting the vehicle miles for the service provider and travel time for users.

Introduction

Dynamic ridesharing services, such as UberPool and Lyft-Line, are becoming an increasingly popular means of commute, especially in large cities (Chan and Shaheen 2012; Bathla et al. 2018). Dynamic ridesharing is characterized by matching multiple requests that arrive in real-time, for a one-way and one-time trip. We consider a setting in which a service provider operates a vehicle fleet and schedules cars to pick up and drop off passengers in response to a stream of requests, which includes matching requests with each other. There are two important factors that explain the growing attractiveness of DRS for customers: (i) cost effectiveness and (ii) ease of finding a ride in large cities where it is comparatively hard to find a taxi otherwise. For a service provider, dynamic ridesharing helps serve customers with possibly fewer vehicles, thus reducing their operational cost.

A common objective for optimizing riders' satisfaction in existing ridesharing systems is to minimize travel time (Ma, Zheng, and Wolfson 2013; Agatz et al. 2012; Bathla et al. 2018). In practice, however, there are many other factors that affect user satisfaction in dynamic ridesharing, apart from travel time. Since a user could be traveling with strangers in the ride, their compatibility plays a major role in the user's satisfaction. In fact, there is growing evidence that desire for personal space and security when riding with strangers pose a major barrier to using ridesharing for many users (Tao and Wu 2008; Agatz et al. 2012). For example, a female passenger may prefer to ride only with female co-passengers. The user may have a different set of preferences depending on the time of day and the location — preferences are trip-specific and not necessarily user-specific.



(a) (b) (c)

Figure 1: An example illustrating the influence of social preferences in trip formation. P denotes a pickup location and D denotes a dropoff location. A trajectory that maximizes operational value to the service provider is shown in (a). Incorporating and satisfying users' social preferences may lead to modification in the trajectory (b) or result in a different trip formation (c).

Consider a scenario with three requests where r_1 and r_2 are male and r_3 is a female passenger. Let these requests arrive at the same time (Figure 1), such that optimizing the operational value for the service provider forms a trip with these requests (1(a)). However, this may violate the users' social preferences and the trip may need to be altered to satisfy the preferences, such as the following:

- If the passengers prefer riding with co-passengers of the same gender but are indifferent to riding with co-passengers of a different gender, then it is desirable to minimize their ride time overlap in the vehicle by altering the pick up and drop off order (Figure 1(b)); and
- When the riders prefer co-passengers of the same gender and wish to *avoid* co-passengers of other gender, then it is better to form two trips (Figure 1(c)).

If the service does not provide a mechanism to express such social preferences and forms trips that violate these preferences (as in 1(a)), the customers may not use the service. Current DRS, however, do not account for social preferences in their optimization, despite being indicated as a major concern for users in several surveys (Agatz et al. 2012; Michalak et al. 1994; Furuhata et al. 2013; Svangren, Skov, and Kjeldskov 2018).

We present DROPS (**D**ynamic **R**idesharing **O**ptimization using social **P**reference**S**), a dynamic ridesharing framework that facilitates incorporating social preferences of the users in the trip formation process. A weight vector over preferences indicates the importance of each factor in determining the trip value to the user. The goal is to form trips that optimize both operational value for the service provider and value of the trip to the passengers, which incentivizes them to continue using the service and benefits the service provider. The value of a trip to a user is calculated based on their social compatibility with other co-passengers, the ride time, and ride cost. We solve this *bi-objective* optimization problem using scalarization (Roijers et al. 2013), which solves a linear combination of the multiple objectives. The relative importance of each objective can be controlled using the weight vector for the objectives. Given a set of riders, we evaluate their potential shared trip using an optimal trajectory planning algorithm. Candidate trips are formed using our real-time greedy algorithm that adds customers to a trip only if the trip's value is above a certain threshold.

We consider three basic social factors — age, gender, and user rating— along with a time preference indicating if the user is in a rush. The viability of factoring social preferences into the trips scheduling process is evaluated empirically. The experiments examine the impact of matching with social preferences (social matching) on users and the service provider. We test our approach on a *real-world taxi trips dataset* and compare the results with that of three baselines, each focusing on optimizing different components of the objective for trip formation. The population model and preferences used in our experiments were acquired using web-based *user surveys*, which was conducted in two phases and had 489 responses. The survey was conducted specifically to determine how different potential riders evaluate social ridesharing. Our results show that incorporating social preferences of users in the trip formation improves the overall user satisfaction, without significantly affecting the operational cost for the service provider.

Our primary contributions are: (i) presenting DROPS, a system for dynamic ridesharing with social preferences; (ii) proposing a real-time greedy algorithm for trip formation; and (iii) extensive empirical evaluation showing the benefits of social matching in dynamic ridesharing using real-world taxi data and a population model based on user surveys.

Related Work

Dynamic ridesharing has gained popularity since the early 2000's due to the cost benefits it offers to the users and service providers, apart from its contributions to sustainable environment resulting from efficient vehicle usage. Dynamic ridesharing is characterized by user requests that arrive in real-time and are matched with vehicles (Levofsky and Greenberg 2001). Another popular ridesharing setting is the car-pooling where users travel together for a particular purpose and the trips are usually recurring (Chan and Shaheen 2012). Our work differs from car-pooling as we focus on a dynamic ridesharing setting with a service provider who operates the vehicle fleet instead of individual car owners and trips that are typically non-recurring.

Optimizing dynamic ridesharing services has been an active research area, attracting researchers from diverse fields such as operations research, transportation, and artificial intelligence (Agatz et al. 2012; Chan and Shaheen 2012; Di Febbraro, Gattorna, and Sacco 2013; Alonso-Mora et al. 2017). Existing literature on dynamic ridesharing can be classified broadly based on the objective function and the solution method employed. Optimization-based approaches are the common solution technique employed (Santos and Xavier 2013; Ma, Zheng, and Wolfson 2013; Di Febbraro, Gattorna, and Sacco 2013; Biswas et al. 2017; Alonso-Mora et al. 2017; Dickerson et al. 2018; Bei and Zhang 2018). Other approaches include partition-based (Pelzer et al. 2015), auction-based mechanisms (Cheng, Nguyen, and Lau 2014), and genetic algorithms (Herbawi and Weber 2012). Researchers have employed these techniques largely to optimize the routing or travel time (Furuhata et al. 2013; Agatz et al. 2012; Herbawi and Weber 2012; Pelzer et al. 2015; Santos and Xavier 2013; Biswas et al. 2017). Specifically, the commonly used objectives for determining ridesharing matches are: (i) minimizing system-wide vehicle-miles; (ii) minimizing system-wide travel time; and (iii) maximizing number of participants.

A critical missing component of these objectives is the in-ride user experience. Numerous studies have outlined the need for learning and understanding user preferences in the context of ridesharing, beyond simple factors like time windows (Chan and Shaheen 2012; Agatz et al. 2012; Thaithatkul et al. 2015). Multiple surveys have acknowledged that it is essential to account for users' social preferences to improve dynamic ridesharing (Agatz et al. 2012; Di Febbraro, Gattorna, and Sacco 2013; Furuhata et al. 2013; Selker and Saphir 2010; Chan and Shaheen 2012; Montazery and Wilson 2016; Tao and Wu 2008; Miller and How 2017; Svangren, Skov, and Kjeldskov 2018; Bistaffa, Farinelli, and Ramchurn 2015). To address this discrepancy, we present a dynamic ridesharing framework that allows for representing and satisfying the social preferences of the users in trip formation.

Problem Formulation

The DROPS framework facilitates customizing rides to improve user compatibility by incorporating the social preferences of users. Let $\mathcal{R}^t$ denote the finite set of unserved (non-dispatched) requests at time t and $\mathcal{V}^t$ denote the finite set of available vehicles at time t. Each request $r \in \mathcal{R}^t$ is denoted by $\langle s, e, i, \vec{p}, U \rangle$. Each vehicle $v \in \mathcal{V}^t$ is denoted by the tuple $\langle ID, \omega \rangle$. Refer Table 1 for the definitions of variables and constants employed in the formulation.

We consider social preferences in each request that correspond to three social factors: age, gender, and rating of users.

Additionally, we consider a time preference to indicate if the user is in a rush. We identified these factors based on the results of our user surveys, conducted specifically to determine user expectations in ridesharing services. The preferences ($\vec{p}$) are denoted as $+1$, -1, or 0, indicating the user's desirability, undesirability, or indifference about a certain value of a factor. For example, a preference of $+1$ for $rating \geq 4$ denotes that the person prefers riding with co-passengers who have a minimum rating of 4, and a preference of -1 for $rating \leq 3$ denotes that the person wishes to avoid riding with co-passengers who have a rating of 3 or below. That is, if rating on a scale of 1 to 5 is treated as a vector, then these preferences are denoted as $\langle -1, -1, -1, +1, +1 \rangle$. The weights $\vec{w} = [w_t, w_a, w_g, w_s]^T$ correspond to the time, age, gender, and rating, respectively.

A solution to an instance of this problem is a set of trips Λ, where each trip $\lambda \in \Lambda$ is a matching of requests to a vehicle and is denoted by $\langle R, v, \tau \rangle$. The value of a trip is denoted by $V(\lambda)$. The objective is to maximize the cumulative value of all trips dispatched in a given horizon H,

$$\max \sum_{t \in H} \sum_{\lambda \in \Lambda^t} V(\lambda).$$

Multi-objective formulation Since the goal is to schedule trips that maximize the operational value for the service provider as well as maximizing the overall user value, this is naturally a bi-objective optimization. To solve this, we employ scalarization (Roijers et al. 2013), which projects a multi-objective value to a single scalar value by parameterizing the objectives using a weight vector. The weight value for each objective indicates its relative importance, thus resulting in a single objective function for optimization. Let β_o denote the weight corresponding to the operational value and let β_u denote the weight corresponding to the user value. Then, $\forall \lambda$, the trip value is:

$$V(\lambda) = \beta_o \underbrace{\sum_{r \in R_\lambda} (x_r - d_r) - c^\tau_\lambda}_{\text{operational value}} + \beta_u \underbrace{\sum_{r \in R_\lambda} (\alpha_r + d_r)}_{\text{user value}} \quad (1)$$

The operational value and the user value are measured in dollars ($) and normalized to the same scale before scalarization. The operational value to the service provider depends on the cost of operating the vehicle for the trip c^τ_λ and the amount paid by the riders, which is the difference between the amount charged for the trip (x_r) and the discount offered for using ridesharing (d_r). The value of the trip to a user depends on the user utility (α_r) and the discount gained for using ridesharing (d_r). The user utility (α_r) is the difference between the users' social compatibility with all their co-passengers and the extra travel time incurred by using ridesharing. The social compatibility for a request is calculated as the cumulative weighted difference between the preferences $\vec{p_r}$ and demographics of each co-passenger.

We now explain the social utility calculation using a simple example. Consider two requests r_1 (female) and r_2 (male) that arrive at the same time and have the same source and destination coordinates, same age (30), and rating (4).

Variables	Definitions
Λ^t	Set of trips formed at time t
$V(\lambda)$	Value of trip λ
β_o, β_u	scalarization weights
$R_\lambda = \{r_1, \dots r_k\}$	Set of requests matched for the trip
c^τ_λ	Cost of using the vehicle for the trip corresponding to the ride route τ
ω_v	Passenger capacity of vehicle
s_r, e_r	Start (pick-up), end (drop-off) locations of r for the trip
α_r	User's social utility
x_r	Amount charged to r for the trip
d_r	Discount for using ridesharing
i_r	Request initiation time
$\vec{p_r}$	Social and time preferences of r
$\vec{w_r}$	User's weights for preferences $\vec{p_r}$
U_r	User demographics: {age, gender, rating}
ID_v	Vehicle ID

Table 1: Notations

r_1 prefers $(+1)$ female co-passengers with age in the range 20-40 with rating ≥ 4 and expresses undesirability (-1) for all other values of social factors. Let the weights of these social preferences be $\vec{w_{r_1}} = [0.3, 0.3, 0.2, 0.5]^T$, corresponding to time, age, gender, and rating. The social compatibility for r_1 with respect to r_2 is $0.3 - 0.2 + 0.5 = 0.6$. Let the extra trip time be 2 minutes, then $\alpha_{r_1} = 0.6 - 0.3 * 2 = 0$.

Solution Approach

Given a set of requests and vehicles, our solution approach consists of two components: (i) trip formation and (ii) trip dispatch. Figure 2 is an illustration of our solution approach. In each decision cycle, the trip formation component matches requests with each other and to vehicles, and the dispatch component decides which trips are dispatched. We restrict the scope of matching in this paper to requests and vehicles that have not been dispatched. That is, we do not consider a vehicle *en-route* (already driving on the road) in the scheduling process and therefore do not match requests to such vehicles. The route planner calculates the optimal trajectory for picking up and dropping off a given set of requests.

Trip Formation

In this phase, requests are matched with other requests and assigned a vehicle to form a trip. The matching is performed using a greedy approach outlined in Algorithm 1. The input to the algorithm is the set of requests and a trip value threshold δ that indicates the required minimum improvement in trip value to form trips. The algorithm adds a request to the best trip (maximum improvement) that improves the trip value at least by a factor of δ and if the trip size has not exceeded the maximum capacity of the vehicle (Lines 7-16). Standard hyperparameter tuning or sample average

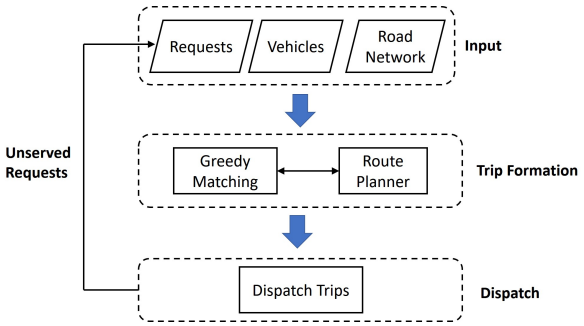


Figure 2: Overview of the solution method.

approximation (Kleywegt, Shapiro, and Homem-de Mello 2002) may be used to estimate δ. The trip value is estimated using Equation 1.

Each request is assigned to the best trip that satisfies the threshold improvement. If no such trip is found, then a new trip is created with the request (Lines 19-22). This ensures that all requests are associated with a trip. The route planner computes trajectories that determine the pick up and drop off order for a given set of requests. All possible trajectories are generated and the one that maximizes the trip value for a given set of requests is selected as the route τ for the trip. During the trip formation, the best route is updated whenever a new request is added to a trip (Line 8, 21). The output of this algorithm is the set of all trips formed, Λ^t.

Algorithm 1: Greedy Matching $(\mathcal{R}^t, \delta)$

1 $\Lambda^t \leftarrow \emptyset$
2 **foreach** $r \in \mathcal{R}^t$ **do**
3 matched = false
4 **if** $|\Lambda^t| > 0$ **then**
5 $\lambda_{best}, \lambda_{rem} \leftarrow \emptyset$
6 Best_Value $= -\infty$
7 **foreach** $\lambda \in \Lambda^t$ *with* $|R_\lambda| < \omega_\lambda$ **do**
8 Calculate best route for $\lambda' = \lambda + r$
9 **if** $\frac{V(\lambda')-V(\lambda)}{V(\lambda)} \geq \delta$ *and* $V(\lambda') >$ *Best_Value* **then**
10 $\lambda_{rem} \leftarrow \lambda$; $\lambda_{best} \leftarrow \lambda'$
11 Best_Value $= V(\lambda_{best})$
12 matched = true
13 **end**
14 **end**
15 **if** *matched = true* **then**
16 $\Lambda^t \leftarrow (\Lambda^t \backslash \lambda_{rem}) \cup \lambda_{best}$
17 **end**
18 **end**
19 **if** *matched = false* **then**
20 Create new trip λ with request r
21 Calculate best route for λ
22 $\Lambda^t \leftarrow \Lambda^t \cup \lambda$
23 **end**
24 **end**
25 **return** Λ^t

Partitioning Requests for Scalability The computational complexity of the matching algorithm discussed above increases rapidly with the increase in number of requests. To counter this computational complexity, we exploit the notion of independence among requests. Two requests q and r are said to be independent if serving them together in the same trip is not desirable in terms of trip value. Hence, all the requests over different days or requests with non-overlapping source-destination pairs are independent. The requests can be partitioned based on their dependence and matches may be formed among each partition in parallel. Sometimes, it is non-trivial to estimate an exact partitioning of requests with respect to routes, without forming trips and calculating the best route possible. In such cases, the underlying map may be partitioned into geographic zones to form trips in each zone independently by considering the requests originating in that zone, as in our experiments.

Trip Dispatch

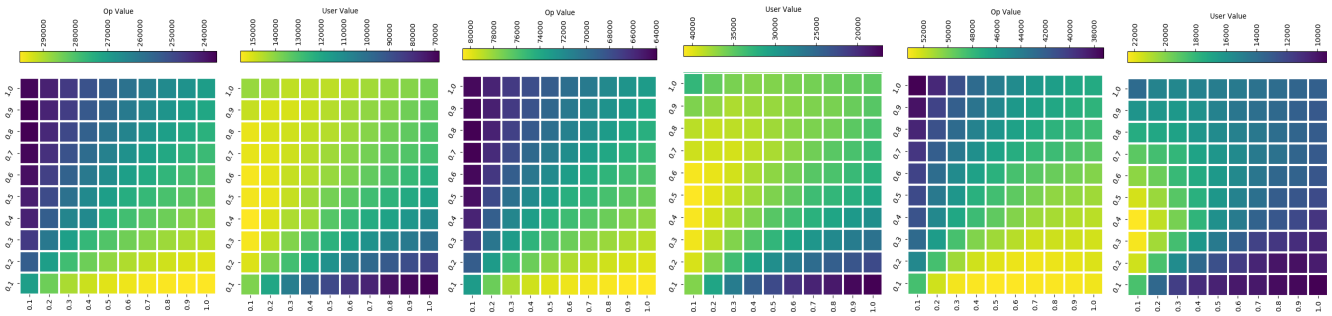
The trips formed in the matching phase are dispatched in this phase if at least one of the following conditions is satisfied: (i) trip value is above the predefined dispatch threshold; or (ii) a request in the trip has remained unserved for a certain period of time since its arrival (queue time). The dispatch threshold for trip value and the queue time for the requests are determined by the service provider. For example, all requests that are unserved for five minutes or more since their arrival time may be dispatched irrespective of the trip value, depending on vehicle availability. In our experiments, trips that satisfy the queue time threshold are given a higher priority over the trips with lower queue time but higher trip value. This ensures that certain requests do not remain unserved forever due to lower trip value. The trips are then dispatched based on availability of vehicles, $\mathcal{V}^t$. At the end of decision cycle t, all unserved requests — requests in trips that are not dispatched — are added to the requests set for the next decision cycle, $\mathcal{R}^{t+1}$.

Experimental Results

The experiments evaluate the impact of using social preferences in ridesharing, with respect to users and the service provider. We built a realistic simulator of ridesharing using the Chicago taxi trips dataset[1] and a population model based on extensive user surveys. We compare the results obtained using social preferences in dynamic ridesharing matching (SM) with that of three baselines: (B_1) maximizing only the operational value, $\beta_u = 0, \beta_o = 1$; (B_2) maximizing only user value, $\beta_u = 1, \beta_o = 0$; and (B_3) maximizing the comprehensive trip value in Equation 1 but without considering user's social preferences corresponding to age, gender, and rating, $w_a = 0, w_g = 0, w_s = 0$, for the trip formation. Algorithm 1 is used to form trips using each baseline objective.

The algorithms and the simulation system were implemented by us on an Intel Xeon 3.10 GHz computer with 16GB of RAM, using a homogeneous vehicle fleet with a

[1] https://data.cityofchicago.org/Transportation/Taxi-Trips/wrvz-psew



(a) Op. Value (Z8) (b) User Value (Z8) (c) Op. Value (Z28) (d) User Value (Z28) (e) Op. Value (Z56) (f) User Value (Z56)

Figure 3: Heat map of the operational (Op.) value and user value corresponding to different weights in each zone.

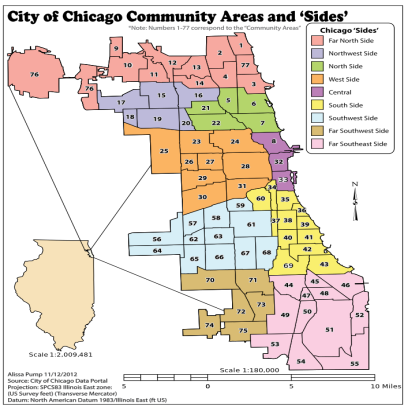


Figure 4: A map of Chicago divided into zones.

seat capacity of 4 for the evaluation. Each decision cycle is 30 seconds in real-time and the horizon H is one day. We assume that the number of vehicles is not bounded since the benefits of social matching are best illustrated in this case and all techniques are equally affected by vehicle restriction. We set the trip threshold δ to zero for the greedy algorithm; requests are added to the best trips possible as long as the current value of the trip is not diminished. This allows us to examine the benefit of social matching uniformly across zones by using a conservative value. However, in practice this hyperparameter may be tuned to further optimize performance subject to the service provider's objective. The request queue time threshold for dispatch is set to five minutes. The travel time and distances are calculated using straight line distances between the coordinates and a vehicle speed of 30 miles per hour. While these experiments do not account for the actual routes and traffic conditions, these factors are not likely to change the relative merits of each approach and the conclusions of the study.

Population Model and Dataset

The population model considered in our experiments is based on the results of online surveys that was conducted in North America. The survey had 489 responses which indicated that users would like to be matched with people who are similar to them. The demographic information such as age and gender, for our experiments, is drawn from the actual Chicago demographic distributions[2]. The preferences ($\vec{p}$) and the weights ($\vec{w}$) are based on the survey results. The survey also indicates that some users are unwilling to use ridesharing when social preferences are not taken into account. To reflect this, certain users were marked as reluctant for ridesharing in the absence of social matching and these users were always dispatched as solo rides, when forming trips with the baseline objectives.

The Chicago taxi trips data consists of trip-specific information such as start time and end time of the taxi ride, trip duration, trip fare, and the latitude and longitude coordinates for pick up and drop off locations along with the geographic zone corresponding to these locations. A map of Chicago divided into zones[3] is shown in Figure 4. We partition the data from each zone into training and testing sets. The weights for scalarization were estimated empirically using the training data (Figure 3). In Figure 3, the x-axis is the weight for operational value (β_o) and the y-axis denotes the weight corresponding to user value (β_u). The weights used for the test sets are $\beta_o = 0.8$ and $\beta_u = 0.6$ for zones 8 and 28, and we used $\beta_o = 0.5$ and $\beta_u = 0.5$ for experiments on zone 56. Our algorithm is evaluated along different metrics on the test set which uses data from two consecutive weeks in April 2015. We consider requests originating in zones 8, 28, and 56, whose requests densities are high, medium, and low respectively. The average number of requests per day in each of these zones is 20000, 7000, and 1500 respectively.

Analysis of Tradeoffs

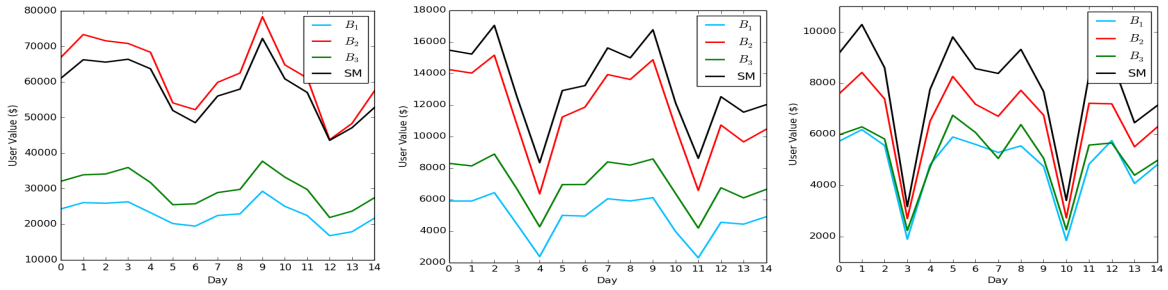
Since user value and the operational value are often competing metrics, we analyze the quality of trips formed with respect to each of these.

Impact on Users We measure the impact on users based on the total user value (Figure 5), average social utility per minute (Figure 6), and the increase in ride time, relative to a solo trip (Figure 7).

Trips formed by maximizing operational value (B_1) have the least user value across all zones, as expected. Our approach (SM) achieves user value close to that of optimiz-

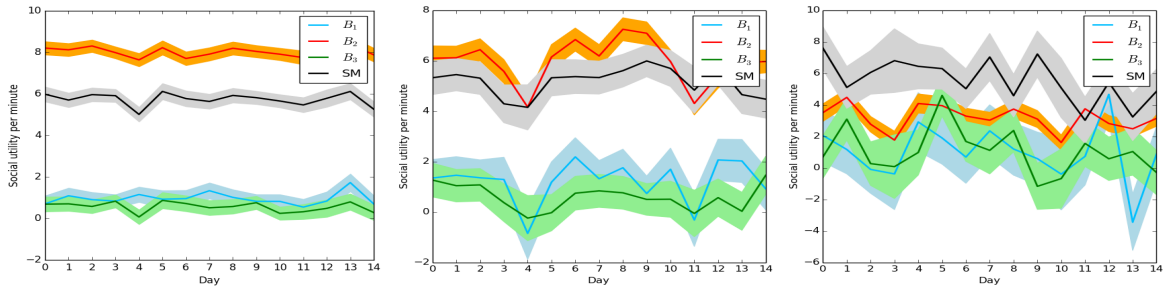
[2]http://chicago.areaconnect.com/statistics.htm

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



(a) Zone 8 (b) Zone 28 (c) Zone 56

Figure 5: Total user value of trips dispatched on each day.



(a) Zone 8 (b) Zone 28 (c) Zone 56

Figure 6: Average social utility per minute for users on each day. This measures the social compatibility of users with their co-passengers in the trips formed using each objective function.

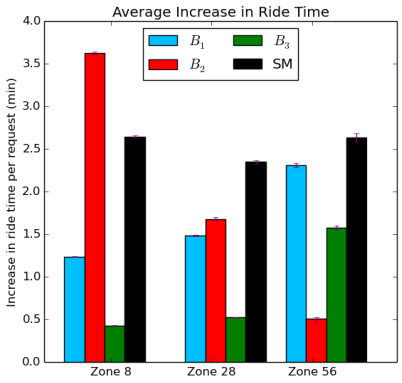


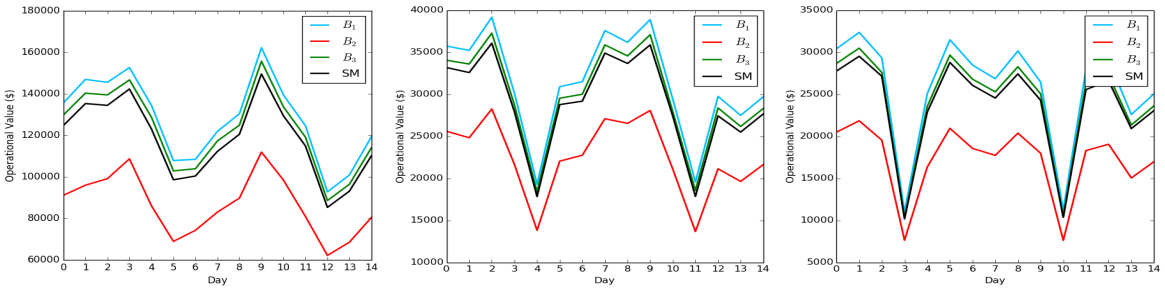
Figure 7: Average increase in ride time of users compared to solo rides. Our approach, on average, increases the ride time by at most 2.5 minutes, well within the acceptable range found in our user surveys.

ing for user value alone (B_2), and sometimes better than B_2. This is because, in some cases, the values of the trips formed by optimizing B_2 objective may not meet the dispatch threshold in which the case the trips are dispatched after five minutes, which eventually reduces the user value. Our approach overcomes this drawback by optimizing for both the objectives, providing greater cumulative value for a given trip and enabling it to be dispatched more quickly.

The social utility (α_r) per minute measures the average social compatibility of users with their co-passengers. To account for the different ride times of the trips, we measure the average utility per minute, along with standard error (Figure 6). We observe that SM consistently performs similar to or better than B_2, showing that the user value is improved through better matching, and not merely based on the ride time or discount offered.

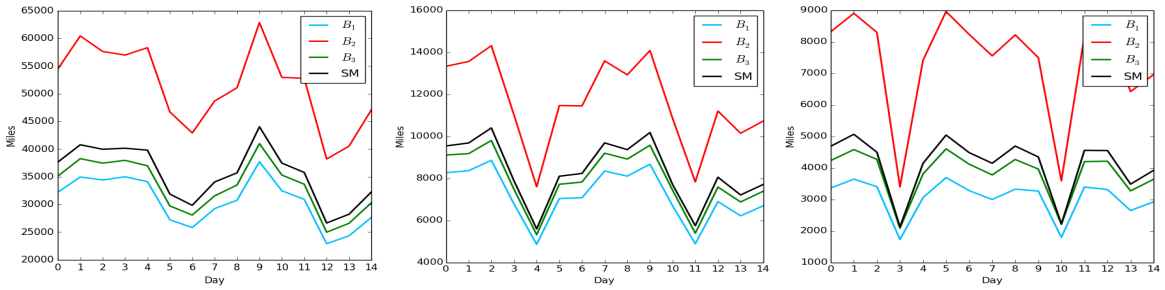
We also evaluated the increase in ride time of the different techniques, compared to solo ride (Figure 7). The average ride times are in the range 10-20 minutes for requests originating in zones of interest. Though the increase in ride time of our technique is around three minutes, note that ridesharing, in general, incurs additional ride time. The increase in ride time of our technique is well within the range that users consider acceptable (at most 5 minutes) according to the survey results. The social compatibility typically offsets the increase in ride time for the users, thus resulting in increased user utility when forming trips using our approach.

Impact on the Service Provider The impact on service provider is determined based on the operational value and the total miles driven, to give a sense of degree of variation induced by social matching on the trip routes and quality of service. As expected, objective B_1 achieves the highest operational value and maximizing B_2 objective has the lowest operational value (Figure 8). The operational value achieved by our approach (SM) is closer to that of B_1, with a slightly higher miles driven (Figure 9) and higher user utility. The total number of trips formed by our approach is also comparable to that of B_1. This shows that our approach improves the quality of trips without significantly affecting the total miles driven or the cost of operating the service by the provider.



(a) Zone 8 (b) Zone 28 (c) Zone 56

Figure 8: Total operational value of trips dispatched on each day.



(a) Zone 8 (b) Zone 28 (c) Zone 56

Figure 9: Total miles driven on each day.

Scalability and Robustness

Since matching is performed every 30 seconds, it is important to ensure that the matching algorithm is fast so that it may be effectively used in real-time. The run time (in seconds) of our matching algorithm is 0.5 on average in the zone with high request density (zone 8), 0.12 in zone 28, and 0.003 in zone 56, demonstrating the scalability of DROPS.

We also compared our matching algorithm to a hindsight greedy matching with access to all the requests in a day, including future ones. The purpose of this experiment is to evaluate the gain in operational value and user value that could be achieved when knowledge of future requests is available. We compare the total operational value obtained using our approach with that of optimizing only for operational value with all requests in a day. Similarly, the total user value obtained with our approach, with requests arriving in real-time, is compared with that of optimizing for user value only and with access to all requests in a day. Trips are formed using the best-fit greedy algorithm (Algorithm 1) for our approach and for the hindsight evaluation.

The results, summarized in Table 2, show that our approach achieves at least ∼89% of the operational value and up to ∼84% of the user value compared to the hindsight matching in all three zones, indicating that any prediction method of future requests would yield very limited performance gains in the operational value. However, some improvements in user value could be achieved with knowledge of future requests by forming trips where the users have a higher social compatibility with co-passengers.

Metrics \ Zone	Zone 8	Zone 28	Zone 56
Operational Value	91.96%	93.03%	89.87%
User Value	83.62%	82.43%	66.71%

Table 2: Performance relative to hindsight optimization.

Conclusion and Future Work

Dynamic ridesharing is an increasingly appealing commuter option. However, numerous surveys have indicated that users' concerns, primarily about the social characteristics of co-passengers, pose a major barrier to using ridesharing for a segment of the population. We present the DROPS system for optimizing dynamic ridesharing with social preferences and present an efficient real-time matching algorithm that can handle effectively high density zones.

Our results demonstrate that factoring social preferences into the matching process helps improve the user value, without significantly affecting the operational value to the service provider. Furthermore, survey results indicate that services that perform social matching are likely to incentivize more individuals to use the service. We conclude that while social matching is beneficial overall, it is not always guaranteed to result in improved performance. Factoring social preferences into the matching process is most beneficial in zones with a high request density per decision cycle and greater compatibility among ridesharing users.

In the future, we aim to examine ways to extend the matching model to consider nearby trips that have already been dispatched and are currently en-route. We will also

consider more complex ways to factor the competing objectives using more general multi-objective planning algorithms (Wray, Zilberstein, and Mouaddib 2015). Additionally, based on the performance analysis of our approach with that of a hindsight trip formation, we aim to employ a predictive model for future requests to improve the user value. While we anticipate some performance gains, we do not expect the relative benefits of social matching to diminish.

Acknowledgments

We thank Shannon Roberts and Fangda Zhang for conducting the user survey and for fruitful discussions on designing the population model.

References

Agatz, N.; Erera, A.; Savelsbergh, M.; and Wang, X. 2012. Optimization for dynamic ride-sharing: A review. *European Journal of Operational Research* 223(2):295–303.

Alonso-Mora, J.; Samaranayake, S.; Wallar, A.; Frazzoli, E.; and Rus, D. 2017. On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *National Academy of Sciences* 114(3):462–467.

Bathla, K.; Raychoudhury, V.; Saxena, D.; and Kshemkalyani, A. D. 2018. Real-time distributed taxi ride sharing. In *Proc. of the 21st International Conference on Intelligent Transportation Systems*.

Bei, X., and Zhang, S. 2018. Algorithms for trip-vehicle assignment in ride-sharing. In *Proc. of the 32nd AAAI Conference on Artificial Intelligence*.

Bistaffa, F.; Farinelli, A.; and Ramchurn, S. D. 2015. Sharing rides with friends: A coalition formation algorithm for ridesharing. In *Proc. of the 29th AAAI Conference on Artificial Intelligence*.

Biswas, A.; Gopalakrishnan, R.; Tulabandhula, T.; Mukherjee, K.; Metrewar, A.; and Thangaraj, R. S. 2017. Profit optimization in commercial ridesharing. In *Proc. of the 16th Conference on Autonomous Agents and MultiAgent Systems*.

Chan, N. D., and Shaheen, S. A. 2012. Ridesharing in north america: Past, present, and future. *Transport Reviews* 32:93–112.

Cheng, S.-F.; Nguyen, D. T.; and Lau, H. C. 2014. Mechanisms for arranging ride sharing and fare splitting for last-mile travel demands. In *Proc. of the 13t International Conference on Autonomous agents and MultiAgent systems*.

Di Febbraro, A.; Gattorna, E.; and Sacco, N. 2013. Optimization of dynamic ridesharing systems. *Transportation Research Record: Journal of the Transportation Research Board* (2359):44–50.

Dickerson, J. P.; Sankararaman, K. A.; Srinivasan, A.; and Xu, P. 2018. Allocation problems in ride-sharing platforms: Online matching with offline reusable resources. In *Proc. of the 32nd AAAI Conference on Artificial Intelligence*.

Furuhata, M.; Dessouky, M.; Ordóñez, F.; Brunet, M.-E.; Wang, X.; and Koenig, S. 2013. Ridesharing: The state-of-the-art and future directions. *Transportation Research* 57:28–46.

Herbawi, W., and Weber, M. 2012. The ridematching problem with time windows in dynamic ridesharing: A model and a genetic algorithm. In *Proc. of the IEEE Congress on Evolutionary Computation*.

Kleywegt, A. J.; Shapiro, A.; and Homem-de Mello, T. 2002. The sample average approximation method for stochastic discrete optimization. *SIAM Journal on Optimization* 12(2):479–502.

Levofsky, A., and Greenberg, A. 2001. Organized dynamic ride sharing: The potential environmental benefits and the opportunity for advancing the concept. In *Transportation Research Board*, 7–11.

Ma, S.; Zheng, Y.; and Wolfson, O. 2013. T-share: A large-scale dynamic taxi ridesharing service. In *Proc. of the 29th International Conference on Data Engineering*.

Michalak, S.; Spyridakis, J.; Haselkorn, M.; Goble, B.; and Blumenthal, C. 1994. Assessing users' needs for dynamic ridesharing. *Proc. of the Transportation Research Record* 32–32.

Miller, J., and How, J. P. 2017. Predictive positioning and quality of service ridesharing for campus mobility on demand systems. In *Proc. of the IEEE International Conference on Robotics and Automation*.

Montazery, M., and Wilson, N. 2016. Learning user preferences in matching for ridesharing. In *Proc. of the International Conference on Agents and Artificial Intelligence*.

Pelzer, D.; Xiao, J.; Zehe, D.; Lees, M. H.; Knoll, A. C.; and Aydt, H. 2015. A partition-based match making algorithm for dynamic ridesharing. *IEEE Transactions on Intelligent Transportation Systems* 16(5):2587–2598.

Roijers, D. M.; Vamplew, P.; Whiteson, S.; and Dazeley, R. 2013. A survey of multi-objective sequential decision-making. *Journal of Artificial Intelligence Research* 48:67–113.

Santos, D. O., and Xavier, E. C. 2013. Dynamic taxi and ridesharing: A framework and heuristics for the optimization problem. In *Proc. of the 23rd International Joint Conference on Artificial Intelligence*.

Selker, T., and Saphir, P. H. 2010. TravelRole: A carpooling/physical social network creator. In *Proc. of the IEEE Int'l Symp. on Collaborative Technologies and Systems*.

Svangren, M. K.; Skov, M. B.; and Kjeldskov, J. 2018. Passenger trip planning using ride-sharing services. In *Proc. of the CHI Conf. on Human Factors in Computing Systems*.

Tao, C., and Wu, C. 2008. Behavioral responses to dynamic ridesharing services- the case of taxi-sharing project in Taipei. In *Proc. of the International Conference on Service Operations and Logistics, and Informatics*.

Thaithatkul, P.; Seo, T.; Kusakabe, T.; and Asakura, Y. 2015. A passengers matching problem in ridesharing systems by considering user preference. *Journal of the Eastern Asia Society for Transportation Studies* 11:1416–1432.

Wray, K. H.; Zilberstein, S.; and Mouaddib, A.-I. 2015. Multi-objective MDPs with conditional lexicographic reward preferences. In *Proc. of the 29th Conference on Artificial Intelligence*.

A Capacitated Vehicle Routing and Scheduling Problem for Passengers: A Modelling and Solution Approach

Sergio Ferrer, Miguel A. Salido, Federico Barber and Adriana Giret
{serfers2, msalido,fbarber, agiret}@dsic.upv.es
Universitat Politècnica de València
Spain

Abstract

In main cities, many of our daily transport requirements are executed by service provider's that must optimize their resources in order to provide the services. Some examples of such services are transportation of school children, courier services, bus tour, etc. In these services, delivery or timely arrival are very important and desirable features that require scheduling and routing of vehicles. One of the most studied combinatorial optimization problems is the Vehicle Routing Problem (VRP) due to its directly application to many real-world cases. This paper describes a novel version of the VRP, named Capacitated Vehicle Routing and Scheduling Problem for Passengers (CVRSPP). The aim of this problem is to schedule a set of buses to different services satisfying a set of constraints. This problem models a real case of the actual discretionary transport industry for groups of passengers, in which every group can hire a bus to travel from one city to any other. The travelers have some requirements that must be satisfied by the transport company and the solution must satisfy the needs from the transport company. When all these constraints are considered, the proposed problem (CVRSPP) can be considered a Capacitated with Fixed Service Time, Maximum Waiting Time and No Depots VRP problem. To this end, a formal mathematical model is proposed and two metaheuristics are developed to solve real-life instances. The empirical results show that the proposed techniques are more competitive than other adapted approaches for solving the VRP.

Introduction

The Vehicle Routing and Scheduling Problem (VRSP) is a well-known problem studied in the literature. The interest and relevance of the VRSP comes from its directly application in real-world environments where the problem is solved by many industries in order to provide their services or to schedule their resources to optimize the associated costs to the logistic needs (Uchoa et al. 2017). The VRP is a complex combinatorial optimization problem that can be seen as a combination of 2 problems: the Travelling Salesperson Problem (TSP) and the Bin Packing Problem (BPP) which are well known NP-hard problems (Tavares et al. 2003; Korf 2002).

The basic version of the VRP (Dantzig and Ramser 1959) consists of delivering a set of packages to a set of customers distributed on a map taking into account that all delivery vehicles start and finish their service at one single depot, minimizing the cost of the routes and the size of the required fleet. This first approach of the VRP makes some assumptions in order to simplify the problem, such as: having one single depot, a homogeneous fleet of vehicles, infinite load for each vehicle, etc.

Due to the high applicability of this problem to multiple contexts, the VRP basic assumptions make it difficult to directly apply it to real life problems, so several variants of the basic version have been proposed. They try to adapt the formalization of the problem to the real needs of the application environment by removing the basic assumptions or by adding new ones. Some of the most studied versions of the VRP are:

- **Capacitated Vehicle Routing Problem (CVRP)**(Gendreau, Laporte, and Potvin 2002a). In this version all vehicles have a maximum capacity that cannot be exceeded and the whole fleet is considered to be uniform, so the maximum capacity is the same for all vehicles. A more complex version of this approach, but also more realistic, can be formalized. In the Multi-Capacity VRP (MCVRP) version (Baldacci, Battarra, and Vigo 2008), every single vehicle has associated a maximum-capacity, that might vary among vehicles.
- **Multiple Depot Vehicle Routing Problem (MDVRP)** (Lahyani, Coelho, and Renaud 2018). This version models the case in which a delivery company has different depots spread across the map. If the costumers are originally clustered on depots, the problem could be solved by solving multiple VRPs independently, but when depots and customers are not related, the problem must be modeled as MDVRP. Solving a MDVRP requires to assign each customer to a depot and then sort them in order to minimize the cost of the travel time and the size of the fleet.
- **Vehicle Routing and Scheduling Problem with Time Windows (VRPTW)**(Solomon 1987; El-Sherbeny 2010). In this generalization of the VRP each costumer i is associated with a time-window $[a_i, b_i]$. Delivery to the customer i must be made before b_i, the time-window upper bound. The vehicle can arrive to the customer address i before a_i, the time-window lower bound, but in order to

service the client, it must have to wait until a_i. In some contexts, the VRPTW also has a time window $[a_0, b_0]$ for the depot. Vehicles cannot leave the depot before a_0 and must be back before b_0.

Many techniques of different nature can be found in literature to solve VRP and its several versions. It is well-known that exact algorithms can only solve small instances of the problem (Laporte 1992; Dinh, Fukasawa, and Luedtke) and they become unviable quickly as the problem grows. Given the intrinsic difficulty of this problem, approximation methods seem to be the most promising for practical size problems. In (Rey et al. 2018), the authors propose a new hybrid approach based on Ant Colony Optimization (ACO) combined with Route First-Cluster Second methods and Local Search procedures to produce high quality solutions for the VRP. Furthermore, the implementation can be executed on multicore CPUs and GPUs using the computing power of modern GPUs programming technologies. It outperforms current ACO-based VRP solvers and proves to be competitive with other high performing metaheuristic solvers.

In (Wei et al. 2018), the VRP with two-dimensional loading constraints (2L-CVRP) is studied (Iori 2005). It designs a set of min-cost routes that start and finish their paths in the depot in order to serve all customers with two-dimensional rectangular weighted items. The paper proposes a Simulated Annealing algorithm with a special mechanism that allows to cooling and raising the temperature repeatedly in order to solve four different versions of 2L-CRVP. The results outperform all existing algorithms on the four versions and reach or improve the best-known solutions for most instances.

In (Yi and Bortfeldt 2018), the capacitated vehicle routing problem with three-dimensional loading constraints (3L-CVRP) (Gendreau et al. 2006) is solved. The authors combine existing state-of-the-art approaches in a high-level framework that stepwise solves the problem. In the first step, a Genetic Algorithm (GA) (Moura and Oliveira 2009) is proposed for solving the container loading problem to find good placements for the packages inside the vehicles. Finally, in the second step, the routing problem is solved by means of a hybrid algorithm which combines a Tabu Search with a Tree Search Algorithm (Bortfeldt 2012). The results show that using the proposed high-level system, the computational effort can be significantly reduced.

In this paper, a new version of the VRSP to transport groups of passengers is proposed. In state-of-the-art approaches, the majority of VRP works are focused on package delivery scenarios. Some applications on passengers bus transportation can be found in (Bowerman, Hall, and Calamai 1995; Özkan Ünsal and Yiğit 2018; Miranda et al. 2018). Nevertheless, they are normally focused on optimizing regular or school routes with a cyclic behavior (the same route and/or schedule every day). The problem we are dealing with in this paper does not have cyclic behavior since it is focused on on-demand discretionary routes for transporting groups of passengers instead of groups of packages. We propose and compare several techniques in order to study which of them are better for this new context for VRP.

It is also interesting to distinguish between routing problems and scheduling problems. If the customers being serviced have no time restrictions and there are no precedence relationships, then the problem is a pure routing problem. However, if there is a specified time for the service to take place, then a scheduling problem exists. Otherwise, we are dealing with a combined routing and scheduling problem (Haksever et al. 2000). In our case, both time and precedence constraints are in place, which define a combined routing and scheduling problem. In (Beck, Prosser, and Selensky 2003), the authors propose a mapping for any VRP to get an equivalent Job Shop Scheduling problem (JSP). Following that approach the services and vehicles from VRP are respectively treated as jobs and machines in JSP.

Problem proposal

In this section, a new version of the VRP is proposed. The main objective of the proposed version is to schedule a set of trips/jobs in a set of vehicles/machines trying to minimize the total traveled distance. Through out this paper, the trips definition and features are from a real company case that tries to optimize passenger transportation at a national level. Passengers transportation entails a set of constraints that change the nature of the pure VRP and compels us to consider new features: *Capacitated* with *Fixed Service Time*, *Maximum Waiting Time* and *No Depots*:

- **Capacitated**: if a group of N passengers needs to travel from one city to another, the assigned vehicle (machine) for this service (job) must have, at least, N available seats. As in the classical version of the CVRP, the selected vehicle may exceed the needs of the service. Henceforth, it is assumed that, for each group of passengers of size K, there is always one vehicle with size greater or equal to K. Furthermore, group partitioning is not allowed , i.e. a given group cannot simultaneously travel in multiple vehicles.
- **Fixed Service Time** (FST): if a group of passengers travels from city A to city B, the service time (time needed to travel from A to B) is fixed and no pre-emption is allowed. Notice that this is not the conventional *Time Window* (TW) in VRPTW. TW is defined as a temporal slot in which a package must be delivered to a customer, but within the TW the vehicle may do different deliveries to multiple customers and it has freedom to decide when to service the customer associated with the TW. Due to the time constraint added by this feature, the VRP is transformed into a VRSP as concluded in (Haksever et al. 2000).
- **No Depots** (ND): a group of passengers may hire a bus from any city of the network. All cities are supposed to have available buses, so the concept of *depot*, which is a requirement for delivery industry, is not required for passengers transportation industry. Also notice that this is not the Multiple Depot (MD) approach from MDVRP. The main difference between MD and ND is that, in MD approaches, depots are normally supposed to be a small subset of the whole set of cities but in ND every single city can indistinctly be a depot or not and it can change its condition depending on the needs of the problem. This

approach forces to consider that each vehicle has its own depot corresponding to its native city.

- **Maximum Waiting Time** (MWT): all vehicles must return to their native/origin cities, i.e. the vehicles first departing city, but they can perform more services until returning to their hometown. To wait in a non-native city until the next service starts is allowed but it implies extra costs (subsistence and accommodation allowance for the vehicle driver, taxes for parking fees on public roads, etc.). Thus, MWT is the maximum time that the vehicles are allowed to wait between services instead of returning to their native city.

If all these features are considered, the classical VRP is transformed into the new proposed version of the problem: *Capacitated Vehicle Routing and Scheduling Problem for Passengers* (CVRSPP).

Problem specification

An instance of the proposed problem is the combination of four elements: a graph $G = (V, E, C)$, representing the map, a specification of the customers demand D, a maximum waiting time MWT and a set B of available vehicles. Each element is formalized as follows:

- $G = (V, E, C)$ where
  - $V = \{v_1, v_2, \ldots, v_m\}$ is the set of vertices of the graph, each one representing a city, where $m = |V|$ is the total number of cities.
  - $E = \{(v_i, v_j) \mid i \neq j;\ v_i, v_j \in V\}$ is a set of edges of the graph. An edge between 2 cities means that it is possible to travel between them.
  - $C = \{c_{v_i,v_j} = [t_{i,j}, d_{i,j}],\ \forall (i,j) \in E\}$ is the set of costs associated to the edges in E. Notice that each edge has two different associated costs:
    - $t_{i,j}$: is the time needed for traveling from v_i to v_j.
    - $d_{i,j}$: is the distance between cities v_i and v_j.
- $D = \{d_1, d_2, \ldots, d_N\}$ is the set of N requested services. Each service $d_i = [p_i, q_i, r_i, s_i]$ is composed of four parameters:
  - $p_i \in V$: is the departure city for service i.
  - $q_i \in V$: is the arrival city for service i.
  - r_i: is the departure time for service i. Service i must start at r_i in p_i and must end at $r_i + t_{p_i q_i}$ in q_i. Delays are not allowed.
  - s_i: is the size (number of passengers) of service i.
- MWT: is the global parameter for the whole instance indicating the Maximum Waiting Time allowed between services.
- $B = \{b_1, b_2, \ldots, b_a\}$: is the set of available vehicles. A vehicle b_i can transport a maximum of c_i passengers. The number of total available buses is $a = |B|$.

  In many environments, where this problem can be applied, the set B is not taken into account. For example, let's suppose a travel agency that wants to hire buses for the transportation of its customers to different airports from multiple cities during a large period of time. In this context, the agency can hire as many buses as it needs from multiple bus companies around the country. Henceforth this approach will be used, which, in practice, only implies that consider B as an infinite set, or at least, a large enough set to assign one different vehicle to each service.

An instantiation of the problem is a tuple $s = [x_1, x_2, \ldots, x_N] \mid x_i \in B$ where x_i is the vehicle/machine assigned to service/job i. Notice that, in the proposed problem, the departure and arrival time is fixed, so the objective is not to sort the services/jobs, but how to schedule them in vehicles/machines in order to save resources. Thus, a complete solution is an assignment of machines to all jobs. Jobs assigned to the same machine are sorted by their departure time. An instantiation s is a *solution* S if the following constraints are satisfied:

1. All services/jobs assigned to the same vehicle/machine must be compatible. Two services/jobs are compatible if they satisfy the following constraints:
   - They do not overlap in time, and also there is enough time for traveling from the arrival city of the first service to the departure city of the second service (eq. 1).

     $$r_j > r_i + t_{p_i q_i} + t_{q_i p_j} \quad \forall i, j \in D \mid x_i = x_j \wedge r_j \geq r_i \tag{1}$$

   - The waiting time between two services is less or equal to MWT (eq. 2).

     $$\begin{gathered} r_j - (r_i + t_{p_i q_i} + t_{q_i p_j}) \leq MWT \\ \forall i, j \in D \mid x_i = x_j \ \wedge \ r_j > r_i \end{gathered} \tag{2}$$

2. The capacity of the vehicle is not exceeded (eq. 3).

   $$c_{x_i} \geq s_i \ \forall x_i \in s \tag{3}$$

The cost of a solution $S = [x_1, x_2, \ldots, x_M] \mid x_i \in B$ can be measured in terms of multiple factors, such as: the size of the fleet needed or the total unused kilometers (unused kilometers are the kilometers that the vehicles need for traveling between jobs without passengers). The size of the fleet can be defined as:

$$|F| : F = \{x_i \in S\} \tag{4}$$

To formally define unused kilometers, some auxiliary definitions are provided:

- $SV_v = \{i \mid x_i \in S \wedge x_i = v\}$: is the set of services assigned to vehicle v.
- $SB_j = \{i \in \{1, N\} \mid r_i + tp_i q_i < r_j \wedge x_i = x_j\}$: SB_j: is the set of services assigned to the same vehicle than service j but scheduled before it.
- $JP_j = \underset{i \in SB_j}{\operatorname{argmax}} \ r_i$ is the job that immediately precede job j.

Using these definitions, the total unused kilometers (UK) traveled by all the vehicles of a given solution can be formally defined as:

$$UK = \sum_{v \in F} \Big(\sum_{j \in SV_v} d_{q_{JP_j} p_j} \Big) + d_{q_{y_v} p_{z_v}} \tag{5}$$

where:

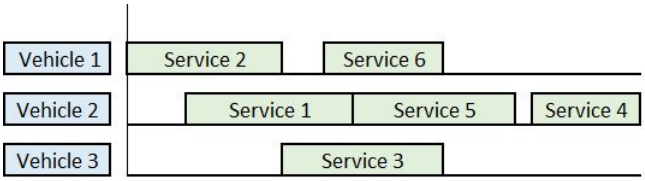


Figure 1: Solution example

- $y_v = \underset{i\,:\,x_i \in s \,\wedge\, x_i = v}{argmax} r_i$: is the last service assigned to vehicle v.
- $z_v = \underset{i\,:\,x_i \in s \,\wedge\, x_i = v}{argmin} r_i$: is the first service assigned to vehicle v.
- $d_{q_{y_v} p_{z_v}}$: is the distance between the last visited city by vehicle v and the first one. This distance must be added to the evaluation since returning to its own depot is mandatory for every vehicle.

The objective of the search process could be to minimize both factors $|F|$ and UK in a multi-objective way in order to minimize both, the needed fleet and the total unused kilometers traveled by the vehicles. To minimize $|F|$ can be very helpful in companies that have a small or medium fleet. However in this work, we will focus on minimizing only UK since the real world case under investigation does not have any constraint on the number of available vehicles because we can hire any number of needed vehicles all around the country.

Solving Techniques

In order to solve the proposed problem we have designed and tested different techniques. In this section, two different metaheuristics for solving the proposed problem are presented.

Solution representation

All the proposed techniques will use the same representation of an instantiation. Let's suppose a problem with 6 services that can be carried out by 3 vehicles as is shown in figure 1. This instantiation could be expressed unambiguously in the terms described in the previous section as $s = [2, 1, 3, 2, 2, 1]$. This representation is formally useful because of its simplicity but it has some computational problems: for example, the computational cost to determine whether an instantiation s is feasible or not is high.

In order to find a computationally affordable representation, a procedure based on *push forward* proposed in (Solomon 1987) have been developed. With this new procedure, an instantiation s takes the form of a list containing all the services IDs in any order. Feasibility of a solution is not compromised by the selected order for the services, that is, all possible permutations of the services represent a feasible solution, which also means that every possible instantiation s is also a solution S . Using the proposed procedure, the instantiation showed in figure 1 could be represented as:

$$s = [2, 6, 1, 5, 4, 3]$$

Algorithm 1: Compatibility checking

input : Pair of jobs=(s1,s2)
output: True if s1 and s2 are compatible. False otherwise.

1 **Function** *compatibles(s1,s2)*:
2 condition_1 = $r_{s2} \geq r_{s1} + t_{p_{s1} q_{s1}} + t_{q_{s1} p_{s2}}$ //(eq. 1) ;
3 condition_2 = $r_j - (r_i + t_{p_i q_i} + t_{q_i p_j}) \leq MWT$ //(eq.2) ;
4 return condition_1 $\wedge$ condition_2 ;

Notice that there is no separator to indicate when the next service is assigned to a new vehicle. This is because, if a separator exists, some combinations could be unfeasible. So, given a list of services IDs in any order (a solution, by definition), it is necessary to carry out a technique to build a schedule and then evaluate it. Thus, 2 different modifications of the *push forward* technique have been developed: *Light Evaluation* (LE) and *Heavy Evaluation* (HE).

Light Evaluation (LE) Let's suppose a solution $S = [9, 2, 1, 3, 5, 7, 4, 10, 6, 11, 8, 12]$ for an instance problem of 12 jobs (see fig. 2). LE technique divides, by a Greedy Algorithm, the whole set of services in multiple subsets, each one corresponding to a different vehicle. This Algorithm (alg. 2) checks, for each used vehicle (line 4), if the job could fit the last position in that vehicle (line 6) and add it in that position (line 7). If the job does not fit any available vehicle (line 10), a new vehicle is added and the job is introduced on it (line 11). Figure 2 better depicts the positions that LE checks in order to introduce a new service into the current vehicle. Gray squares are the tested position. If a service does not fits any gray square according to the constraints (lines 2 and 3 from alg. 1), then a new vehicle is added with the service inside itself (line 11).

Algorithm 2: LE algorithm

input : A list of services IDs: solution S
output: A division of the services IDs in subsets, each one corresponding to a vehicle.

1 vehicles = [ ] ;
2 **for** $s \in S$ **do**
3 introduced = False ;
4 **for** $b \in vehicles$ **do**
5 last_position = Length(b) - 1 ;
6 **if** *compatibles(b[last_position], s)* **then**
7 b.append(s) ;
8 introduced = True ;
9 break;
10 **if** $\neg introduced$ **then**
11 vehicles.append([s]) ;
12 return vehicles;

Heavy Evaluation (HE) LE is a fast way to build a feasible schedule from any solution s because it only checks one

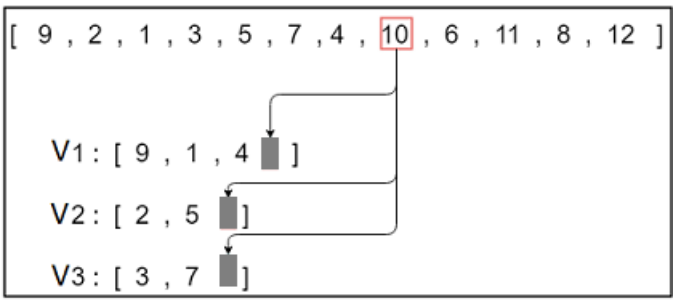


Figure 2: LE checked positions

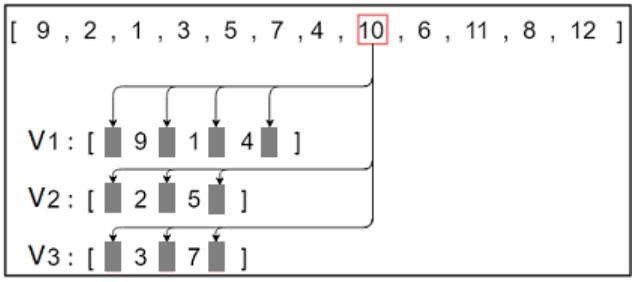


Figure 3: HE checked positions

Algorithm 3: HE algorithm

input : A list of services IDs: solution S
output: A division of the services IDs in subsets, each one corresponding to a vehicle.

```
1  vehicles = [ ] ;
2  for s ∈ S do
3      introduced = False ;
4      for b ∈ vehicles do
5          for s1 ∈ b do
6              if compatibles(s1, s) then
7                  b.append(s, position(s1)) ;
8                  introduced = True ;
9                  break;
10         if introduced then
11             break;
12     if ¬introduced then
13         vehicles.append([s]) ;
14 return vehicles;
```

single position for each available vehicle. This procedure can be modified to check all positions in the vehicle. This variation will probably find better solutions but the computational cost is higher. This alternative is presented as *Heavy Evaluation*. A comparison between these 2 procedures will be carried out in the evaluation section. Algorithm 3 shows the HE procedure and figure 3 shows the positions (gray squares) that HE checks for introducing a new service into the current vehicle.

GRASP

Greedy Randomized Adaptive Search Procedure (GRASP) is a metaheuristic commonly used to solve combinatorial optimization problems. The GRASP metaheuristic proposes a constructive phase, where a solution is found in a randomized greedy way and a local search phase where the solution is improved. The first phase is executed until a timeout is reached and the process returns the best solution found. Afterwards the second phase tries to improve the solution. Two variants of the GRASP algorithm have been implemented: GRASP with local search after the constructive process (GRASP after) and GRASP with local search during the constructive process (GRASP during). Algorithm 4 shows the GRASP procedure. Line 4 is used only in "GRASP during" version and line 8 is used only in "GRASP after" version. Evaluation in line 5 is carried out in terms of UK (see eq. 5)

Algorithm 4: GRASP algorithm

input : A list S of sorted services and a timeout T
output: A division of the services IDs in subsets, each one corresponding to a vehicle.

```
1  best_eval = inf;
2  while ¬timeout do
3      sched = constructive_phase(S);
4      sched = local_search(sched) //GRASP
         DURING;
5      if evaluation(sched) < best_eval then
6          best_eval = evaluation(sched);
7          best = sched;
8  best = local_search(best) //GRASP AFTER;
9  return best ;
```

GRASP constructive phase works as follows:

1. Services are introduced, sorted by their r_i (input).
2. A process looks for a subset of compatible services (lines 7-8) and schedule them in the same vehicle (line 10). Scheduled services are removed from the list (line 11) and the process continues with the remaining services (line 7). The probability of linking two services (line 9) i, j is inversely proportional to the distance that separates them:

$$1 - \frac{d_{q_i p_j}}{max\ d_{q_w p_z} \forall w, z \in [1, d]} \quad (6)$$

3. The process continues until the list of services remains empty (line 3). Algorithm 5 shows the GRASP constructive phase procedure.

Local search phase is executed after the constructive phase or after the timeout assigned to the constructive phase, depending on the GRASP version that is being executed. The algorithm that implements the local search phase is a simple process that iteratively tries to swap the position of two services, to analyze if the new solution improves the previous one, according to eq. 5. This procedure is proposed in alg. 6. Swap function in line 4 is a simple script that swaps the position in the schedule of the 2 services passed as parameters. All possible pair of jobs are checked for swapping (lines 1-2) in a O(n^2) algorithm and the GRASP process

Algorithm 5: GRASP constructive phase

input : A list S of sorted services
output: A division of the services IDs in subsets, each one corresponding to a vehicle.

```
1  vehicles = [ ] ;
2  max_distance = max d_{q_w p_z} ∀w, z ∈ [1, d] ;
3  for s ∈ S do
4      last = s ;
5      new_vehicle = [s] ;
6      S.remove(s) ;
7      for s2 ∈ S do
8          if compatibles(last, s2) then
9              if random() ≤ 1 − d_{last_i s2_j} / max_distance then
10                 new_vehicle.append(s2) ;
11                 S.remove(s2) ;
12                 last = s2 ;
13     vehicles.append(new_vehicle) ;
14 Function compatibles(s1,s2):
15     condition_0 = s1 ≠ s2 ;
16     condition_1 = r_{s2} ≥ r_{s1} + t_{p_{s1} q_{s1}} + t_{q_{s1} p_{s2}}
         //(eq. 1 ;
17     condition_2 =
         r_j − (r_i + t_{p_i q_i} + t_{q_i p_j}) ≤ MWT //eq.2 ;
18     return condition_0 ∧ condition_1 ∧ condition_2
```

ends giving as output an improved solution or the original one.

Simulated Annealing (SA)

Simulated Annealing (Gendreau, Laporte, and Potvin 2002b) is a metaheuristic that tries to emulate the behavior of hot materials cooling down slowly until reaching regular solid structures. It is supposed that the slower the material cools, the more regular and perfect will be the solid structure reached. An iteration of the SA consists on transforming a current solution s_t into a new solution s'_t by making random minor changes on s_t. If s'_t is better than s_t, then s_{t+1} will be s'_t, otherwise s'_t is accepted as s_{t+1} with a probability that is usually decreasing as execution progresses. Formally:

$$s_{t+1} = \begin{cases} s'_t & \text{if f}(s'_t) > \text{f}(s_t) \\ s'_t \text{ with probability } p_t & \text{if f}(s'_t) \leq \text{f}(s_t) \\ s_t & \text{otherwise} \end{cases}$$

where:

- f(x) is the application of equation 5 to the solution x.
- p_t is the probability to accept a solution that worsens the current solution. This probability is normally defined as:

$$p_t = exp\left(-\frac{f(s_t) - f(s'_t)}{\theta_t}\right)$$

  where θ_t is the current time-step of the algorithm execution. This time-step allows that, as the execution progresses, it is increasingly difficult to accept solutions that worsen the current solution.

Algorithm 6: GRASP local search phase

input : A schedule SCH = list of services separated in vehicles
output: An improved schedule, if found. Otherwise, the input schedule

```
1  for s1 ∈ vehicles do
2      for s2 ∈ vehicles do
3          eval1 = evaluation(SCH);
4          SCH.swap(s1,s2);
5          eval2 = evaluation(SCH);
6          if eval2<eval1 then
7              break;
8          else
9              SCH.swap(s2,s1) //undo swap
10 return SCH;
```

The proposed SA (alg. 7) transforms a solution s_t into s'_t by swapping the position of two randomly selected services (lines 8-14). Evaluation of a solution is carried out by using equation 5 after the LE or HE procedures (line 15). The SA needs an initial solution to start its execution. In order to build this initial solution, two different approaches have been developed:

- The initial solution is the result of randomly shuffle all services.
- The initial solution comes from sorting all services by their departure time r_i.

Evaluation

To evaluate the proposed techniques, it is necessary to create some problem instances. Since the CVRSPP is a new version of the problem, there is no benchmark available in the literature to test different solving techniques. Thus, some real data instances provided by a confidential collaborating company have been analyzed to generate a new synthetic but realistic benchmark. The collaborating company is a touristic operator that works at a national level in Spain, hiring and combining the services in a centralized way. We received the whole set of services that the company carried out during one complete year, and the created benchmark tries to respect the nature of this real data.

An instance of the problem is composed of 2 parts: a map and a set of services to be carried out by vehicles. The generation of maps and services are independent processes.

A map is actually the graph $G = (V, E, C)$ defined in the *problem specification* section. The map has been created following the following steps:

1. A 2-dimensional Euclidean grid of size 100x100 is created.
2. 50 points, each one representing a city, are randomly located on the grid. Each city is a vertex of the graph.
3. Vertices are located in a Euclidean space, so there exists an Euclidean distance between each pair of vertices. This also means that you can travel from every city to any other

Algorithm 7: Simulated Annealing

input : A List S of services and initial temperature K
output: A scheduled solution

```
best_s = S ;
current_s = S ;
current_eval = evaluation(LE(current_s)) // or HE ;
best_eval = current_eval ;
iterations = 0 ;
while T>1 do
    iterations += 1 ;
    pos1 = random.int(0, length(S)) ;
    pos2 = random.int(0, length(S)) ;
    new_s = copy(current_s);
    //swap positions ;
    aux = new_s[pos1] ;
    new_s[pos1] = new_s[pos2] ;
    new_s[pos2] = aux ;
    new_eval = evaluation(LE(new_s)) // or HE ;
    if new_eval < current_eval then
        best_s = new_s ;
        current_s = new_s ;
        current_eval = new_eval ;
    else
        energy = exp(-(current_eval-new_eval)/iterations) ;
        if energy<random() then
            current_s = new_s ;
            current_eval = new_eval ;
    T = T*0.99 ;
return best_s ;
```

city in the map in a straight line. The edges of the graph represent these straight lines.

4. Distances between two cities $d_{i,j}$ in C are the Euclidean distances in the grid. Assuming that distances are measured in kilometers and that vehicles travel at v km/h, the times $t_{i,j}$ in C between cities are defined as:

$$t_{i,j} = \frac{d_{i,j} + random(-1,1) * 0.25 * d_{i,j}}{v}$$

The random component can add or subtract up to 25% of the real distance between cities (function $random(a,b)$ returns a float number $\in$ [a,b]). This causes that, as happens in the real-world cases, smaller distances could need more time to be traveled than bigger ones.

The set of services $D = \{[p_i, q_i, r_i, s_i] \mid \forall i \in [1,d]\}$ is also generated in a randomized way, but in this case, some features of the real-world have been emulated in order to have more realistic instances:

- p_i: In real-world instances, it has been observed that about 10% of cities on the map were chosen by 62% of the services as departure cities. This commonly occurs with important cities of a country. In order to emulate this behavior, 10% of the cities are randomly selected as *important cities* and 62% of the p_i in services are select from the set of *important cities* while the 38% of the remaining p_i are randomly selected.
- q_i: The *important cities* that are commonly selected by passengers as departure cities are also selected as arrival cities by the 32% of the services. Again, this situation is emulated by assigning 32% of the services to q_i from the set of *important cities*. Again, the remaining services are randomly assigned as arrival city.
- r_i: The time horizon for each instance is set to 15 days. In order to have a discrete quantization of the 15 days, they are measured in "number of quarter hours". Accordingly, the departure time (r_i) for each service is randomly selected in the interval [0, 1440] (1440 is the total number of quarter hours in 15 days). For example, if a service i has its $r_i = 136$ means that its departure time is 10:00 AM of the second day.
- s_i: The most commonly vehicles used for passengers transportation have one of the following sizes: 30, 54, 55 or 70 seats. In real-world cases, 70% of the services use vehicles with 54 or 55 seats, so s_i is assigned respecting this percentage. Remaining 30% is randomly assigned.

Different sizes of instances were generated by modifying the parameter d (number of total services). 3 classes of instances were built, depending on the size, with 50 instances each class. Table 1 shows a description of the benchmark.

The proposed benchmark for the CVRSPP were solved with 3 different techniques: GRASP and SA from the previous section and an adaptation of the Genetic Algorithm (GA) proposed in (Baker and Ayechew 2003). The parametrization for this GA can be summarized as:

1. the representation of an individual is made according to LE and HE techniques. Both versions will be compared.
2. the size of the population is fixed to 600 individuals.
3. 300 individuals are selected to be crossed.
4. each individual has 30% of mutation probability. Mutation consists on swapping two randomly selected services.
5. crossover is carried out by 2-Point Crossover (Kora and Yadlapalli 2017).
6. fitness of a solution is calculated in terms of unused kilometers (UK) (eq. 5).
7. regarding the substitution method, all the new individuals that are inserted into the population are ordered by fitness and the best 600 individuals are selected.

All techniques have been executed with a 300 seconds timeout or a convergence criterion. This criterion stops the search if it performs 1000 iterations without improvement.

Class name	Size	Number of instances
I_250	250 services	50
I_500	500 services	50
I_1000	1000 services	50

Table 1: Benchmark

		Unused Kilometres (eq.5)		
Technique	Variant	I_250	I_500	I_1000
GA	LE	128695,44	268615,94	547433,06
	HE	80089,4	158643,53	304641,56
SA	LE	79879,88	170450,52	346754,82
	HE	82983,04	163806,78	309075,5
	LE+sort	65660,04	117246,74	196706,14
	HE+sort	65961,04	117.022'32	196.462'4
GRASP	After	**48763,74**	**87410,34**	**148574,12**
	During	50839,06	93497,38	159622,9

Table 2: Unused kilometers for different techniques

		Execution time (s)		
Technique	Variant	I_250	I_500	I_1000
GA	LE	306,33	315,39	343,87
	HE	356,5	523,5	647,57
SA	LE	5,23	15,38	45,38
	HE	6,12	26,07	106,668
	LE+sort	6,93	19,9	59,16
	HE+sort	68,85	229,68	836,96
GRASP	After	5,7	7,68	16,43
	During	**5,14**	**5,62**	**7,6**

Table 3: Execution time for different techniques

Table 2 shows the unused kilometers reached for each technique with all techniques and variations. The table shows the arithmetic average of the 50 instances per class. GRASP has no LE and HE versions because LE and HE are procedures to build the schedule from a representation, but GRASP has its own procedure (algorithm 5). The annealing versions tagged with "+sort" means that the initial solution for the SA was created sorting the services by their r_i while SA versions without this tag were executed initializing the solution randomly.

As shown in table 2 it is not easy to determinate which of the HE or LE procedures are the best options for the proposed problem because on each technique, they obtain different results: on GA the best results come from HE but with SA the best results come from LE, depending on the instance size. It is also interesting to point out that, in all techniques, when the problem doubles its size; the evaluation function (unused kilometers) has a similar behavior since it doubles its value, approximately.

Table 3 shows the average execution time for solving each instance size. It can be observed that SA and GRASP have better behavior that GA in all instances. It is also observed than GA exceeds the timeout (300 s.) in all its executions. This is due to the fact that GA spends all the time evaluating the initial population (the process cannot be interrupted during the initialization). When the initialization is finished, the search process should start but the timeout has been exceeded, so the process finishes its execution returning the best value in the initial population. The main problem of the GA is that the search process needs, at least, a medium-size population to find good solutions but this population has a very high computational cost to be maintained (evaluated and checked for feasibility). Notice that SA and GRASP finish the execution by the convergence criterion without reaching the timeout. Finally, GRASP maintains the best behavior minimizing the unused kilometers and converging in low time. Particularly, depending on the location of the local search in the GRASP algorithm, the best results are obtained in unused kilometers or in execution time. In any case, a GRASP algorithm is considered a competitive metaheuristic for solving this class of problems.

Figure 4 shows the number of buses and the number of unused kilometers for solving all instances of class I_1000. It can be observed that most of the instances use between 175 and 210 buses to solve their problems. The number of unused kilometers was ranged between 142000 and 154000 kms for most instances. However there is no relationship between both parameters. Using more buses does not represent a lower amount of unused kilometers. Similar results were obtained for I_250 and I_500.

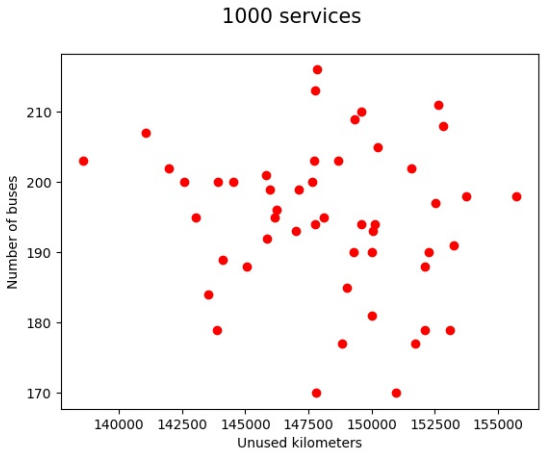


Figure 4: Unused kilometers vs Number of buses for I_1000

Conclusions and Future Work

This paper proposes a novel version of the VRP, named Capacitated Vehicle Routing and Scheduling Problem for Passengers (CVRSPP). The main objective of this problem is to schedule a set of buses to different services satisfying a set of constraints. When all these constraints are considered, the proposed problem can be considered a Capacitated with Fixed Service Time, Maximum Waiting Time and No Depots VRP problem. This problem models a real case of the actual discretionary transport industry for groups of passengers. The formal mathematical model has been presented and two metaheuristics have been developed to solve this problem. A benchmark for the proposed problem has been developed and it will be available at the research group webpage. The generated benchmark respects the nature of real-world cases providing realistic instances. The results shows that GRASP is a competitive technique compared with other adapted approaches for solving the VRP. It is able to save up to 30% of unused kilometers with respect the solutions

obtained by experts in real life instances.

As future work it is proposed to add more constraints to the problem in order to make it more realistic. Such constraints are related with including a maximum driving time per driver, including some special features in some buses and services (fridge, access for the handicapped, TV, etc.). It is also a future work to tackle the dynamic rescheduling of this problem since, in real-world cases, vehicles are often damaged during services, traffic jams can delay arrival times, etc. These situations can transform a feasible solution into a non-feasible solution that requires to solve these problems in a dynamic way.

Acknowledgements

The paper has been partially supported by the Spanish research project TIN2016-80856-R and TIN2015-65515-C4-1-R.

References

Baker, B. M., and Ayechew, M. 2003. A genetic algorithm for the vehicle routing problem. *Computers and Operations Research* 30(5):787 – 800.

Baldacci, R.; Battarra, M.; and Vigo, D. 2008. *Routing a Heterogeneous Fleet of Vehicles*. Boston, MA: Springer US. 3–27.

Beck, J. C.; Prosser, P.; and Selensky, E. 2003. Vehicle routing and job shop scheduling: What's the difference? In *ICAPS*.

Bortfeldt, A. 2012. A hybrid algorithm for the capacitated vehicle routing problem with three-dimensional loading constraints. *Comput. Oper. Res.* 39(9):2248–2257.

Bowerman, R.; Hall, B.; and Calamai, P. 1995. A multi-objective optimization approach to urban school bus routing: Formulation and solution method. *Transportation Research Part A: Policy and Practice* 29(2):107 – 123.

Dantzig, G. B., and Ramser, J. H. 1959. The truck dispatching problem. *Management Science* 6(1):80–91.

Dinh, T.; Fukasawa, R.; and Luedtke, J. Exact algorithms for the chance-constrained vehicle routing problem. *Mathematical Programming* 172(1):105–138.

El-Sherbeny, N. A. 2010. Vehicle routing with time windows: An overview of exact, heuristic and metaheuristic methods. *Journal of King Saud University - Science* 22(3):123 – 131.

Gendreau, M.; Iori, M.; Laporte, G.; and Martello, S. 2006. A tabu search algorithm for a routing and container loading problem. *Transportation Science* 40(3):342–350.

Gendreau, M.; Laporte, G.; and Potvin, J.-Y. 2002a. *6. Metaheuristics for the Capacitated VRP*. 129–154.

Gendreau, M.; Laporte, G.; and Potvin, J.-Y. 2002b. *6. Metaheuristics for the Capacitated VRP*. 129–154.

Haksever, C.; Render, B.; Russell, R. S.; and Murdick, R. G. 2000. *Service Management and Operations (2nd Edition)*.

Iori, M. 2005. Metaheuristic algorithms for combinatorial optimization problems. *4OR* 3(2):163–166.

Kora, P., and Yadlapalli, P. 2017. Crossover operators in genetic algorithms: A review. *International Journal of Computer Applications* 162(10).

Korf, R. E. 2002. A new algorithm for optimal bin packing. In *Eighteenth National Conference on Artificial Intelligence*, 731–736. Menlo Park, CA, USA: American Association for Artificial Intelligence.

Lahyani, R.; Coelho, L. C.; and Renaud, J. 2018. Alternative formulations and improved bounds for the multi-depot fleet size and mix vehicle routing problem. *OR Spectrum* 40(1):125–157.

Laporte, G. 1992. The vehicle routing problem: An overview of exact and approximate algorithms. *European Journal of Operational Research* 59(3):345 – 358.

Miranda, D. M.; de Camargo, R. S.; Conceição, S. V.; Porto, M. F.; and Nunes, N. T. 2018. A multi-loading school bus routing problem. *Expert Systems with Applications* 101:228 – 242.

Moura, A., and Oliveira, J. F. 2009. An integrated approach to the vehicle routing and container loading problems. *OR Spectrum* 31(4):775–800.

Özkan Ünsal, and Yiğit, T. 2018. Using the genetic algorithm for the optimization of dynamic school bus routing problem. *BRAIN. Broad Research in Artificial Intelligence and Neuroscience* 9(2):6–21.

Rey, A.; Prieto, M.; Gómez, J. I.; Tenllado, C.; and Hidalgo, J. I. 2018. A cpu-gpu parallel ant colony optimization solver for the vehicle routing problem. In Sim, K., and Kaufmann, P., eds., *Applications of Evolutionary Computation*, 653–667. Cham: Springer International Publishing.

Solomon, M. M. 1987. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Oper. Res.* 35(2):254–265.

Tavares, J.; Pereira, F. B.; Machado, P.; and Costa, E. 2003. Crossover and diversity: A study about gvr. In *In Proceedings of the Analysis and Design of Representations and Operators (ADoRo'2003*, 27–33.

Uchoa, E.; Pecin, D.; Pessoa, A.; Poggi, M.; Vidal, T.; and Subramanian, A. 2017. New benchmark instances for the capacitated vehicle routing problem. *European Journal of Operational Research* 257(3):845 – 858.

Wei, L.; Zhang, Z.; Zhang, D.; and Leung, S. C. 2018. A simulated annealing algorithm for the capacitated vehicle routing problem with two-dimensional loading constraints. *European Journal of Operational Research* 265(3):843 – 859.

Yi, J., and Bortfeldt, A. 2018. The capacitated vehicle routing problem with three-dimensional loading constraints and split delivery—a case study. In Fink, A.; Fügenschuh, A.; and Geiger, M. J., eds., *Operations Research Proceedings 2016*, 351–356. Cham: Springer International Publishing.

Automated Reasoning in Real Domains

Enabling Limited Resource-Bounded Disjunction in Scheduling

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

Abstract

We describe three approaches to enabling an extremely computationally limited embedded scheduler to consider a small number of alternative activities based on resource availability. We consider the case where the scheduler is so computationally limited that it cannot backtrack search. The first two approaches precompile resource checks (called guards) that only enable selection of a preferred alternative activity if sufficient resources are estimated to be available to schedule the remaining activities. The final approach mimics backtracking by invoking the scheduler multiple times with the alternative activities. We present an evaluation of these techniques on mission scenarios (called sol types) from NASA's next planetary rover where these techniques are being evaluated for inclusion in an onboard scheduler.

Introduction

Embedded schedulers must often operate with very limited computational resources. Due to such limitations, it is not always feasible to develop a scheduler with a backtracking search algorithm. This makes it challenging to perform even simple schedule optimization when doing so may use resources needed for yet unscheduled activities.

In this paper, we present three algorithms to enable such a scheduler to consider a very limited type of preferred activity while still scheduling all required (hereafter called *mandatory*) activities. Preferred activities are grouped into *switch groups*, sets of activities, where each activity in the set is called a *switch case*, and exactly one of the activities in the set must be scheduled. They differ only by how much time, energy, and data volume they consume and the goal is for the scheduler to schedule the most desirable activity (coincidentally the most resource consuming activity) without sacrificing any other mandatory activity.

The target scheduler is a non-backtracking scheduler to be onboard the NASA Mars 2020 planetary rover (Rabideau and Benowitz 2017) that schedules in priority first order and never removes or moves an activity after it is placed during a single run of the scheduler. Because the scheduler does not backtrack, it is challenging to ensure that scheduling a consumptive switch case will not use too many resources and therefore prevent a later (in terms of scheduling order, not necessarily time order) mandatory activity from being scheduled.

The onboard scheduler is designed to make the rover more robust to run-time variations by rescheduling multiple times during execution (Gaines et al. 2016a). If an activity ends earlier or later than expected, then rescheduling will allow the scheduler to consider changes in resource consumption and reschedule accordingly. Our algorithms to schedule switch groups must also be robust to varying execution durations and rescheduling.

We have developed several approaches to handle scheduling switch groups. The first two, called guards, involve reserving enough sensitive resources (time, energy, data volume) to ensure all later required activities can be scheduled. The third approach emulates backtracking under certain conditions by reinvoking the scheduler multiple times. These three techniques are currently being considered for implementation in the Mars 2020 onboard scheduler.

Problem Definition

For the scheduling problem we adopt the definitions in (Rabideau and Benowitz 2017). The scheduler is given

- a list of activities
  $A_1\langle p_1, d_1, R_1, e_1, dv_1, \Gamma_1, T_1, D_1\rangle \ldots$
  $A_n\langle p_n, d_n, R_n, e_n, dv_n, \Gamma_n, T_n, D_n\rangle$
- where p_i is the scheduling priority of activity A_i;
- d_i is the nominal, or predicted, duration of activity A_i;
- R_i is the set of unit resources $R_{i_1} \ldots R_{i_m}$ that activity A_i will use;
- e_i and dv_i are the rates at which the consumable resources energy and data volume respectively are consumed by activity A_i;
- $\Gamma_{i_1} \ldots \Gamma_{i_r}$ are non-depletable resources used such as sequence engines available or peak power for activity A_i;
- T_i is a set of start time windows $[T_{i_{j\_start}}, T_{i_{j\_preferred}}, T_{i_{j\_end}}] \ldots [T_{i_{k\_start}}, T_{i_{k\_preferred}}, T_{i_{k\_end}}]$ for activity A_i. [1] ;

[1] If a preferred start time, $T_{i_{j\_preferred}}$ is not specified for window j then it is by default $T_{i_{j\_start}}$

- D_i is a set of activity dependency constraints for activity A_i where $A_p \rightarrow A_q$ means A_q must execute successfully before A_p starts.

The goal of the scheduler is to schedule all mandatory activities and the best switch cases possible while respecting individual and plan-wide constraints.

Each activity is assigned a *scheduling priority*. This priority determines the order in which the activity will be considered for addition to the schedule. The scheduler attempts to schedule the activities in priority order, therefore: (1) higher priority activities can block lower priority activities from being scheduled and (2) higher priority activities are more likely to appear in the schedule.

Mandatory Activities are activities, $m_1 \ldots m_j \subseteq A$, that must be scheduled. The presumption is that the problem as specified is *valid*, that is to say that a schedule exists that includes all of the mandatory activities, respects all of the provided constraints, and does not exceed available resources.

In addition, activities can be grouped into *Switch Groups*. The activities within a switch group are called *switch cases* and vary by how many resources (time, energy, and data volume) they consume. It is mandatory to schedule exactly one switch case and preferable to schedule a more resource intensive one, but not at the expense of another mandatory activity. For example, one of the Mars 2020 instruments takes images to fill mosaics which can vary in size; for instance we might consider $1x4$, $2x4$, or $4x4$ 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:

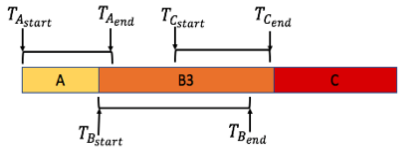
$$SwitchGroup = \begin{cases} Mosaic_{1x4} & d = 100 \text{ sec} \\ Mosaic_{2x4} & d = 200 \text{ sec} \\ Mosaic_{4x4} & d = 400 \text{ sec} \end{cases} \quad (1)$$

The desire is for the scheduler to schedule the activity $Mosaic_{4x4}$ but if it does not fit then try scheduling $Mosaic_{2x4}$, and eventually try $Mosaic_{1x4}$ if the other two fail to schedule. It is not worth scheduling a more consumptive switch case if doing so will prevent a future, lower priority mandatory activity from being scheduled due to lack of resources. Because our computationally limited scheduler cannot search or backtrack, it is a challenge to predict if a higher level switch case will be able to fit in the schedule without consuming resources that will cause another lower priority mandatory activity to be forced out of the schedule.

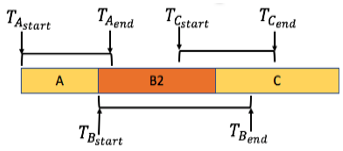
Consider the following example in Figure 1 where the switch group consists of activities B1, B2, and B3 and $d_{B3} > d_{B2} > d_{B1}$. Each activity in this example also has one start time window from $T_{i_{start}}$ to $T_{i_{end}}$.

B3 is the most resource intensive and has the highest priority so the scheduler will first try scheduling B3. As shown in Figure 1a, scheduling B3 will prevent the scheduler from placing activity C at a time satisfying its execution constraints. So, B3 should not be scheduled.

The question might arise as to why switch groups cannot simply be scheduled last in terms of scheduling order. This is difficult for several reasons: 1) We would like to avoid gaps



(a) Scheduling B3 first prevents activity C from being scheduled within its start time window.



(b) B2 can be successfully scheduled without dropping any other mandatory activities.

Figure 1: Challenge to Schedule Switch Cases.

in the schedule which is most effectively done by scheduling primarily left to right temporally, and 2) if another activity is dependent on an activity in a switch group, then scheduling the switch group last would introduce complications to ensure that the dependencies are satisfied.

The remainder of the paper is organized as follows. First, we describe several plan wide energy constraints that must be satisfied. Then, we discuss two guard approaches to schedule preferred activities, which place conditions on the scheduler that restrict the placement of switch cases under certain conditions. We then discuss various versions of an approach which emulates backtracking by reinvoking the scheduler multiple times with the switch cases. We present empirical results to evaluate and compare these approaches.

Energy Constraints

There are several energy constraints which must be satisfied throughout scheduling and execution. The scheduling process for each *sol*, or Mars day, begins with the assumption that the rover is asleep for the entire time spanning the sol. Each time the scheduler places an activity, the rover must be awake so the energy level declines. When the rover is asleep the energy level increases.

Two crucial energy values which must be taken into account are the *Minimum State of Charge (SOC)* and the *Minimum Handover State of Charge*. The state of charge, or energy value, cannot dip below the Minimum SOC at any point. If scheduling an activity would cause the energy value to dip below the Minimum SOC, then that activity will not be scheduled. In addition, the state of charge cannot be below the *Minimum Handover SOC* at the *Handover Time*, in effect when the next schedule starts (e.g., the handover SOC of the previous plan is the expected beginning SOC for the subsequent schedule).

In order to preserve battery life, the scheduler must also consider the *Maximum State of Charge* constraint. Exceeding the Maximum SOC hurts long term battery performance and the rover will perform *shunting*. To prevent it from exceeding this value, the rover may be kept awake.

Guard Approaches

First we will discuss two guard methods to schedule switch cases, the Fixed Point guard and the Sol Wide guard. Both of these methods attempt to schedule switch cases by reserving enough time and energy to schedule the remaining mandatory activities. For switch groups, this means that resources will be reserved for the least resource consuming activity since it is mandatory to schedule exactly one activity in the switch group. The method through which both of these guard approaches reserve enough time to schedule future mandatory activities is the same. They differ in how they ensure there is enough energy. While the Fixed Point guard reserves enough energy at a single fixed time point - the time at which the least resource consuming switch case is scheduled to end in the nominal schedule, the Sol Wide guard attempts to reserve sufficient energy by keeping track of the energy balance in the entire plan, or sol.

In this discussion, we do not attempt to reserve data volume while computing the guards as it is not expected to be as constraining of a resource as time or energy. We aim to take data volume into account as we continue to do work on this topic.

Both the time and energy guards are calculated offline before execution occurs using a nominal schedule. Then, while rescheduling during execution, the constraints given by the guards are applied to ensure that scheduling a higher level switch case will not prevent a future mandatory activity from being scheduled. If activities have ended sufficiently early and freed up resources, then it may be possible to reschedule with a more consumptive switch case.

Guarding for Time

First, we will discuss how the Fixed Point and Sol Wide guards ensure enough time will be reserved to schedule remaining mandatory activities while attempting to schedule a more resource consuming switch case.

If a preferred time, $T_{i_{j\_preferred}}$, is specified for an activity, the scheduler will try to place an activity closest to its preferred time while obeying all other constraints. Otherwise, the scheduler will try to place the activity as early as possible.

Each switch group in the set of activities used to create a *nominal schedule* includes only the nominal, or least resource consuming switch case, and all activities take their predicted duration. First, we generate a nominal schedule and find the time at which the nominal switch case is scheduled to complete, as shown in Figure 2.

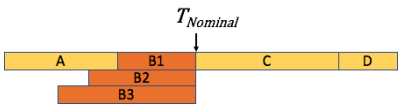


Figure 2: A, B1, C, and D are all mandatory activities in the nominal schedule. $T_{Nominal}$ is the time at which B1 is scheduled to end.

We then manipulate the execution time constraints of the more resource intensive switch cases, B2 and B3 in Figure 2, so that they are constrained to complete by $T_{Nominal}$ as shown in Equation 2. Thus, a more (time) resource consuming switch case will not use up time from any remaining lower priority mandatory activities. If an activity has more than one start time window, then we only alter the one which contains $T_{Nominal}$ and remove the others. If a prior activity ends earlier than expected during execution and frees up some time, then it may be possible to schedule a more consumptive switch case while obeying the time guard given by the altered execution time constraints.

$$T_{B_{ij\_end}} = T_{Nominal} - d_{B_i} \quad (2)$$

Since we found that the above method was quite conservative and heavily constrained the placement of a more resource consuming switch case, we attempted a *preferred time method* to loosen the time guard. In this approach, we set the preferred time of the nominal switch case to its latest start time before generating the nominal schedule. Then, while the nominal schedule is being generated, the scheduler will try to place the nominal switch case as late as possible since the scheduler will try to place an activity as close to its preferred time as possible. As a result, $T_{Nominal}$ will likely be later than what it would be if the preferred time were not set in this way. As per Equation 2, the latest start times, $T_{B_{ij\_end}}$, of the more resource consuming switch cases may be later than what they would be using the previous method where the preferred time was not altered, thus allowing for wider start time windows for higher level switch cases. This method has some risks. If the nominal switch case was placed as late as possible, it could use up time from another mandatory activity with a tight execution window that it would not otherwise have used up if it was placed earlier, as shown in Figure 3.

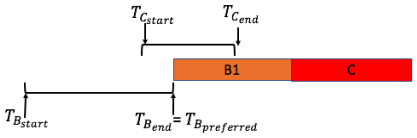


Figure 3: Scheduling B1 at its latest start time prevents C from being scheduled within its start time window.

Guarding for Energy

Fixed Point Minimum State of Charge Guard The Fixed Point method attempts to ensure that scheduling a more resource consuming switch case will not cause the energy to violate the Minimum SOC while scheduling any future mandatory activities by reserving sufficient energy at a single, fixed point in time, $T_{Nominal}$ as shown in Figure 4. The guard value for the Minimum SOC is the state of charge value at $T_{Nominal}$ while constructing the nominal schedule. When attempting to schedule a more resource intensive switch case, a constraint is placed on the scheduler so that the energy cannot fall below the Minimum SOC guard value at time $T_{Nominal}$. If an activity ends early (and uses

fewer resources than expected) during execution, it may be possible to satisfy this guard while scheduling a more consumptive switch case.

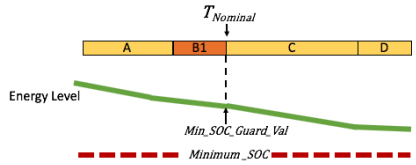


Figure 4: A, B1, C, and D, are mandatory activities in the nominal schedule. A constraint is placed so that the energy cannot dip below $Min\_SOC\_Guard\_Val$ at time $T_{Nominal}$ while trying to schedule a higher level switch case.

Fixed Point Handover State of Charge Guard The Fixed Point method guards for the Minimum Handover SOC by first calculating how much extra energy is left over in the nominal schedule at handover time after scheduling all activities, as shown in Figure 5.

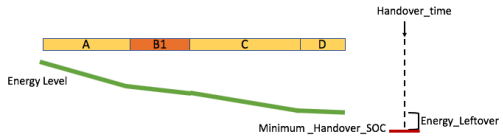


Figure 5: A, B1, C, and D, are mandatory activities in the nominal schedule. A constraint is placed so that the extra energy a higher level switch case consumes cannot exceed *Energy_Leftover*.

Then, while attempting to place a more consumptive switch case, a constraint is placed on the scheduler so that the extra energy required by the switch case does not exceed *Energy_Leftover* from the nominal schedule as in Figure 5. For example, if we have a switch group consisting of three activities, B1, B2, and B3 and $d_{B3} > d_{B2} > d_{B1}$ and each switch case consumes e Watts of power, we must ensure that the following inequality holds at the time the scheduler is attempting to schedule a higher level switch case:

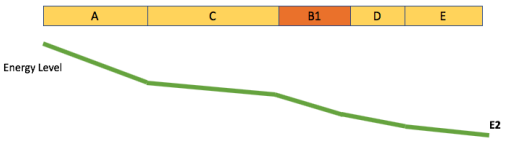
$$(d_{B_i} \times e_{B_i}) - (d_{B_1} \times e_{B_1}) \geq Energy\_Leftover \quad (3)$$

There may be more than one switch group in the schedule. Each time a higher level switch case is scheduled, the *Energy_Leftover* value is decreased by the extra energy required to schedule it. When the scheduler tries to place a switch case in another switch group, it will check against the updated *Energy_Leftover*.

Sol Wide Handover State of Charge Guard The Sol Wide handover SOC guard only schedules a more resource consumptive switch case if doing so will not cause the energy to dip below the Handover SOC at handover time. First, we use the nominal schedule to calculate how much energy is needed to schedule remaining mandatory activities. Having a Maximum SOC constraint while calculating this value may produce an inaccurate result since any energy that would exceed the Maximum SOC would not be taken into account. So, in order to have an accurate prediction of the energy balance as activities are being scheduled, this value is calculated assuming there is no Maximum SOC constraint. 8. The Maximum SOC constraint is only removed while computing the guard offline to gain a clear understanding of the energy balance but during execution it is enforced

As shown in Figure 6, the energy needed to schedule the remaining mandatory activities is the difference between the energy level just after the nominal switch case has been scheduled, call this E1, and after all activities have been scheduled, call this energy level E2.

(a) E1 is the energy level of the nominal schedule with no Maximum SOC constraint after all activities up to and including the nominal switch case (A, D, B1) have been scheduled.



(b) E2 is the energy level of the nominal schedule with no Maximum SOC constraint after all activities in the nominal schedule have been scheduled. The activities were scheduled the following order: A, D, B1, C, E.

Figure 6: Calculating Energy Needed to Schedule Remaining Mandatory Activities.

$$Energy\_Needed = E1 - E2 \quad (4)$$

Then, a constraint is placed on the scheduler so that the energy value after a higher level switch case is scheduled must be at least:

$$Energy\_Level \geq Minimum\_Handover\_SOC + Energy\_Needed \quad (5)$$

By placing this energy constraint, we hope to prevent the energy level from falling under the Minimum Handover SOC by the time all activities have been scheduled.

Sol Wide Minimum State of Charge Guard While we ensure that the energy will not violate the minimum Handover SOC by keeping track of the energy balance, it is possible that scheduling a longer switch case will cause the energy to fall below the Minimum SOC. To limit the chance of this happening, we run a Monte Carlo of execution offline while computing the sol wide energy guard. We use this Monte Carlo to determine if a mandatory activity was

not scheduled due to a longer switch case being scheduled earlier. If this occurs in any of the Monte Carlos of execution, then we increase the guard constraint in Equation 5. We first find the times at which each mandatory activity was scheduled to finish in the nominal schedule. Then, we run a Monte Carlo of execution with the input plan containing the guard and all switch cases. Each Monte Carlo differs in how long each activity takes to execute compared to its original predicted duration in the schedule. If a mandatory activity was not executed in any of the Monte Carlo runs and a more resource consuming switch case was executed before the time at which that mandatory activity was scheduled to complete in the nominal schedule, then we increase the Sol Wide energy guard value in Equation 5 by a fixed amount. We aim to compose a better heuristic to increase the guard value as we continue work on this subject.

Multiple Scheduler Invocation Approach

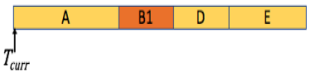
The Multiple Scheduler Invocation (MSI) approach emulates backtracking by reinvoking the scheduler multiple times with the switch cases. MSI does not require any precomputation offline before execution as with the guards and instead reinvokes the scheduler multiple times during execution. During execution, the scheduler reschedules (e.g., when activities end early) with only the nominal switch case as shown in Figure 7a until an MSI trigger is satisfied. At this point, the scheduler is reinvoked multiple times, at most once per switch case in each switch group. In the first MSI invocation, the scheduler attempts to schedule the highest level switch case as shown in Figure 7b. If the resulting schedule does not contain all mandatory activities, then the scheduler will attempt to schedule the next highest level switch case, as in 7c, and so on. If none of the higher level switch cases can be successfully scheduled then the schedule is regenerated with the nominal switch case. If activities have ended early by the time MSI is triggered and resulted in more resources than expected, then the goal is for this approach to generate a schedule with a more consumptive switch case if it will fit (assuming nominal activity durations for any activities that have not yet executed).

There are multiple factors that must be taken into consideration when implementing MSI:

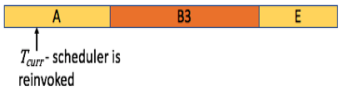
When to Trigger MSI There are two options to trigger the MSI process (first invocation while trying to schedule the switch case):

1. *Time Offset.* Start MSI when the current time during execution is some fixed amount of time, X, from the time at which the nominal switch case is scheduled to start in the current schedule (shown in Figure 8).
2. *Switch Ready.* Start MSI when an activity has finished executing and the nominal switch case activity is the next activity scheduled to start (shown in Figure 9).

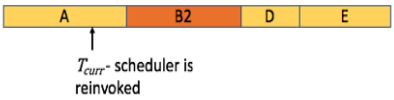
Spacing Between MSI Invocations If the highest level switch case activity is not able to be scheduled in the first invocation of MSI, then the scheduler must be invoked again. We choose to reschedule as soon as possible after the most recent MSI invocation. This method risks over-consumption



(a) MSI has not yet begun. Currently, the nominal switch case, B1, is scheduled.



(b) MSI begins. Scheduling the highest level switch case, B3, prevents D from being scheduled. Therefore, try B2.



(c) B2 is successfully scheduled along with the other mandatory activities so MSI is complete.

Figure 7: Order of MSI Invocations.

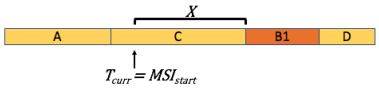


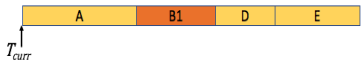
Figure 8: MSI Time Offset.

of the CPU if the scheduler is invoked too frequently. To handle this, we may need to rely on a process within the scheduler called *throttling*. Throttling places a constraint which imposes a minimum time delay between invocations, preventing the scheduler from being invoked at too high of a rate. An alternative is to reschedule at an evenly split, fixed cadence to avoid over-consumption of the CPU; we plan to explore this approach in the future.

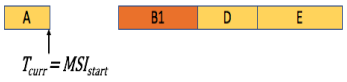
Switch Case Becomes Committed In some situations, the nominal switch case activity in the original plan may become committed before or during the MSI invocations as shown in Figure 10. An activity is *committed* if its scheduled start time is between the start and end of the commit window (Chien et al. 2000). A committed activity cannot be rescheduled and is committed to execute. If the nominal switch case remains committed, the scheduler will not be able to elevate to a higher level switch case.

There are two ways to handle this situation:

1. *Commit the activity.* Keep the nominal switch case activity committed and do not try to elevate to a higher level switch case.
2. *Veto the switch case.* Veto the nominal switch case so that it is no longer considered in the current schedule. When an activity is vetoed, it is removed from the current schedule and will be considered in a future invocation of the scheduler. Therefore, by vetoing the nominal switch case,



(a) B1 is the nominal switch case. Since an activity has not finished executing and B1 is not the next activity, MSI cannot begin yet.



(b) Since A finished executing early, and B1 is the next activity, the MSI process can begin.

Figure 9: MSI Switch Ready.

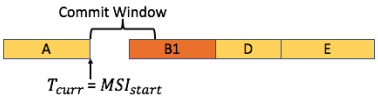


Figure 10: Switch case is committed during MSI. T_{curr} is the current time during execution. MSI_{start} is the time at which MSI begins. The nominal switch case, B1, is committed when MSI begins.

it will no longer be committed and the scheduler will continue the MSI invocations in an effort to elevate the switch case.

Handling Rescheduling After MSI Completes but before the Switch Case is Committed After MSI completes, there may be events that warrant rescheduling (e.g., an activity ending early) before the switch case is committed. When the scheduler is reinvoked to account for the event, it must know which level switch case to consider. If we successfully elevated a switch case, we choose to reschedule with that higher level switch case. Since the original schedule generated by MSI with the elevated switch case was in the past and did not undergo changes from this rescheduling, it is possible the schedule will be inconsistent and may lead to complications while scheduling later mandatory activities. An alternative we plan to explore in the future is to disable rescheduling until the switch case is committed. However, this approach would not allow the scheduler to regain time if an activity ended early and caused rescheduling.

Empirical Analysis

In order to evaluate the performance of the above methods, we apply them to various sets of inputs comprised of activities with their constraints and compare them against each other. 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 *Mars 2020 surrogate scheduler* - an implementation of the same algorithm as the Mars 2020 onboard scheduler (Rabideau and Benowitz 2017), but intended 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 sol type contains between 20 and 40 activities. Data from the Mars Science Laboratory Mission (Jet Propulsion Laboratory 2017b; Gaines et al. 2016a; 2016b) indicates that activity durations were quite conservative and completed early by around 30%. However, there is a desire by the mission to operate with a less conservative margin to increase productivity. In our model to determine activity execution durations, we choose from a normal distribution where the mean is 90% of the predicted, nominal activity duration. The standard deviation is set so that 10 % of activity execution durations will be greater than the nominal duration. For our analysis, if an activity's execution duration chosen from the distribution is longer than its nominal duration, then the execution duration is set to be the nominal duration to avoid many complications which result from activities running long (e.g., an activity may not be scheduled solely because another activity ran late). Detailed discussion of this is the subject of another paper. 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.

We create 10 variants derived from each of 8 sol types by adding one switch group to each set of inputs for a total of 80 variants. The switch group contains three switch cases, $A_{nominal}$, A_{2x}, and A_{4x} where $d_{A_{4x}} = 4 \times d_{A_{nominal}}$ and $d_{A_{2x}} = 2 \times d_{A_{nominal}}$.

In order to evaluate the effectiveness of each method, we have developed a scoring method based on how many and what type of activities are able to be scheduled successfully. 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).

Each mandatory activity that is successfully scheduled, including whichever switch case activity is scheduled, contributes one point to the *mandatory score*. A successfully scheduled switch case that is 2 times as long as the original activity contributes $1/2$ to the *switch group score*. A successfully scheduled switch case that is 4 times as long as the original, nominal switch case contributes 1 to the switch group score. If only the nominal switch case is able to be scheduled, it does not contribute to the switch group score at all. There is only one switch group in each variant, so the maximum switch group score for a variant is 1. Since scheduling a mandatory activity is of much higher importance than scheduling any number of higher level switch case, the mandatory activity score is weighted at a much larger value then the switch group score. In the following empirical results, we average the mandatory and switch groups scores over 20 Monte Carlo runs of execution for each variant.

We compare the different methods to schedule switch cases over varying incoming state of charge values (how much energy exists at the start) and determine which methods result in 1) scheduling all mandatory activities and 2) the highest switch group scores. The upper bound for the theoretical maximum switch group score is given by an *omniscient scheduler*- a scheduler which has prior knowledge of the execution duration for each activity. Thus, this scheduler is aware of the amount of resources that will be available to schedule higher level switch cases given how long activities take to execute compared to their predicted, nominal duration. The input activity durations fed to this omniscient scheduler are the actual execution durations. We run the omniscient scheduler at most once per switch case. First, we try to schedule with only the highest level switch case and if that fails to schedule all mandatory activities, then we try with the next level switch case, and so on.

First, we determine which methods are able to successfully schedule all mandatory activities, indicated by the Maximum Mandatory Score in Figure 11. Since scheduling a mandatory activity is worth much more than scheduling any number of higher level switch cases, we only compare switch group scores between methods that successfully schedule all mandatory activities.

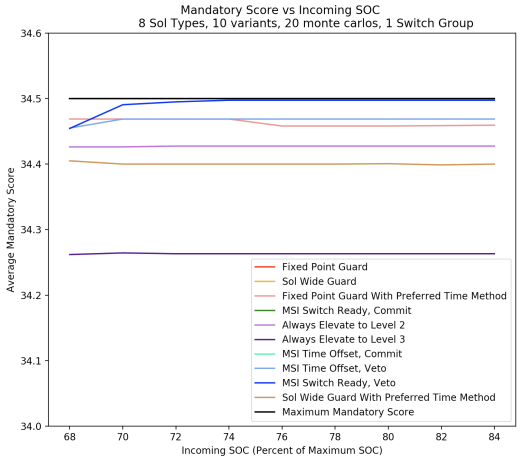


Figure 11: Mandatory score vs Incoming SOC for various Methods to Schedule Switch Cases

In order to evaluate the ability of each method to schedule all mandatory activities, we also compare against two other methods, one which always elevates to the highest level switch case while the other always elevates to the medium level switch case. We see in Figure 11 that always elevating to the highest (3rd) level performs the worst and drops approximately 0.25 mandatory activities per sol, or 1 activity per 4 sols on average while always elevating to the second highest level drops close to 0.07 mandatory activities per sol, or 1 activity per 14 sols on average. For comparison, the study described in (Gaines et al. 2016a) showed that approximately 1 mandatory activity was dropped every 90 sols, indicating that both of these heuristics perform poorly.

We found that using preferred time to guard against time

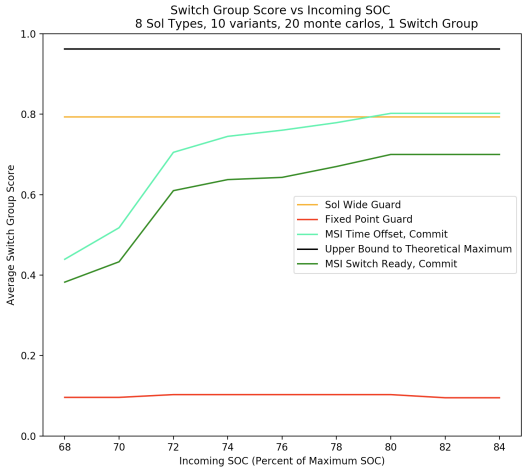


Figure 12: Switch Group Score vs Incoming SOC for Methods which Schedule all Mandatory Activities

caused mandatory activities to drop for both the fixed point and sol wide guard (for the reason described in the Guarding for Time section) while using the original method to guard against time did not. We see in Figure 11 that the preferred time method with the fixed point guard drops on average about 0.04 mandatory activities per sol, or 1 activity every 25 sols while the sol wide guard drops on average about 0.1 mandatory activities per sol, or 1 activity every 10 sols. We also see that occasionally fewer mandatory activities are scheduled with a higher incoming SOC. Since using preferred time does not properly ensure that all remaining activities will be able to be scheduled, a higher incoming SOC can allow a higher level switch case to be scheduled, preventing future mandatory activities from being scheduled.

The MSI approaches which veto to handle the situation where the nominal switch case becomes committed before or during MSI drop mandatory activities. Whenever an activity is vetoed, there is always the risk that it will not be able to be scheduled in a future invocation, more so if the sol type is very tightly time constrained, which is especially true for one of our sol types. Thus, vetoing the nominal switch case can result in dropping the activity, accounting for this method's inability to schedule all mandatory activities. The MSI methods that keep the nominal switch case committed and do not try to elevate to a higher level switch case successfully schedule all mandatory activities, as do the guard methods.

We see that the Fixed Point guard, Sol Wide guard, and two of the MSI approaches are able to successfully schedule all mandatory activities. As shown in Figure 12, the Sol Wide guard and MSI approach using the options Time Offset and Commit result in the highest switch group scores closest to the upper bound for the theoretical maximum. Both MSI approaches have increasing switch group scores with increasing incoming SOC since a higher incoming energy will result in more energy to schedule a consumptive switch case during MSI. The less time there is to complete all MSI

invocations, the more likely it is for the nominal switch case to become committed. Since we give up trying to elevate switch cases and keep the switch case committed if this occurs, fewer switch cases will be elevated. Because our time offset value, X, in Figure 8 is quite large (15 minutes), this situation is more likely to occur using the Switch Ready approach to choose when to start MSI, explaining why using Switch Ready results in a lower switch score than Time Offset.

The Fixed Point guard results in a significantly lower switch case score because it checks against a state of charge constraint at a particular time regardless of what occurs during execution. Even if a switch case is being attempted to be scheduled at a completely different time than $T_{Nominal}$ in Figure 2, (e.g., because prior activities ended early), the guard constraint will still be enforced at that particular time. Since we simulate activities ending early, more activities will likely complete by $T_{Nominal}$, causing the energy level to fall under the Minimum SOC Guard value. Unlike the Fixed Point guard, since the the Sol Wide guard checks if there is sufficient energy to schedule a higher level switch case at the time the scheduler is attempting to schedule it, not at a set time, it is better able to consider resources regained from an activity ending early.

We also see that using the Fixed Point guard begins to result in a lower switch group score with higher incoming SOC levels after the incoming SOC is 80% of the Maximum SOC. Energy is more likely to reach the Maximum SOC constraint with a higher incoming SOC. The energy gained by an activity taking less time than predicted will not be able to be used if the resulting energy level would exceed the Maximum SOC. If this occurs, then since the extra energy cannot be used, the energy level may dip below the guard value in Figure 4 at time $T_{Nominal}$ while trying to schedule a higher level switch case even if an activity ended sufficiently early, as shown in Figure 13.

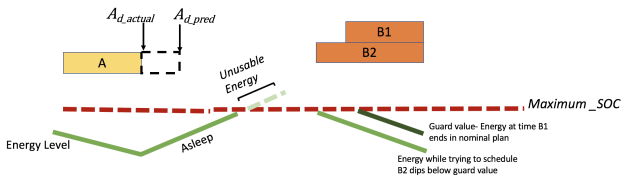


Figure 13: Fixed Point Guard Schedules Fewer Mandatory Activities with Higher Incoming SOC

Related Work

Just-In-Case Scheduling (Drummond, Bresina, and Swanson 1994) uses a nominal schedule to determine areas where breaks in the schedule are most likely to occur and produces a branching (tree) schedule to cover execution contingencies. Our approaches all (re) schedule on the fly although the guard methods can be vewied as forcing schedule branches based on time and resource availability.

Kellenbrink and Helber (Kellenbrink and Helber 2015) solve RCPSP (resource-constrained project scheduling problem) where all activities that must be scheduled are not known in advance and the scheduler must decide whether or not to perform certain activities of varying resource consumption. Similarly, our scheduler does not know which of the switch cases to schedule in advance, using runtime resource information to drive (re) scheduling.

Integrated planning and scheduling can also be considered scheduling disjuncts (chosen based on prevailing conditions (e.g., (Barták 2000))) but these methods typically search whereas we are too computationally limited to search.

Discussion and Future Work

There are many areas for future work. Currently the time guard heavily limits the placement of activities. As we saw, using preferred time to address this issue resulted in dropping mandatory activities. Ideally analysis of start time windows and dependencies could determine where an activity could be placed without blocking other mandatory activities.

Additionally, in computing the guard for Minimum SOC using the Sol Wide Guard, instead of increasing the guard value by a predetermined fixed amount which could result in over-conservatism, binary search via Monte Carlo analysis could more precisely determine the guard amount.

Currently we consider only a single switch group per plan, the Mars 2020 rover mission desires support for multiple switch groups in the input instead. Additional work is needed to extend to multiple switch groups.

Further exploration of all of the MSI variants is needed. Study of starting MSI invocations if an activity ends early by at least some amount and the switch case is the next activity is planned. We would like to analyze the effects of evenly spacing the MSI invocations in order to avoid relying on throttling and we would like to try disabling rescheduling after MSI is complete until the switch case has been committed and understand if this results in major drawbacks.

We have studied the effects of time and energy on switch cases, and we would like to extend these approaches and analysis to data volume.

Conclusion

We have presented several algorithms to allow a very computationally limited, non-backtracking scheduler to consider a schedule containing required, or mandatory, activities and sets of activities called switch groups where each activity in such sets differs only by its resource consumption. These algorithms strive to schedule the most preferred, which happens to be the most consumptive, activity possible in the set without dropping any other mandatory activity. First, we discuss two guard methods which use different approaches to reserve enough resources to schedule remaining mandatory activities. We then discuss a third algorithm, MSI, which emulates backtracking by reinvoking the scheduler at most once per level of switch case. We present empirical analysis using input sets of activities derived from data on expected planetary rover operations to show the effects of using each of these methods. These implementations and empirical evaluation are currently being evaluated in the context of the Mars 2020 onboard scheduler.

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

Barták, R. 2000. Conceptual models for combined planning and scheduling. *Electronic Notes in Discrete Mathematics* 4(1).

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.; Knight, R.; Stechert, A.; Sherwood, R.; and Rabideau, G. 2000. Using iterative repair to improve the responsiveness of planning and scheduling. In *Artificial Intelligence Planning and Schedling*, 300–307.

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.

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.

Kellenbrink, C., and Helber, S. 2015. Scheduling resource-constrained projects with a flexible project structure. *European Journal of Operational Research* 246(2):379–391.

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

Quantum Circuit Compilation: An Emerging Application for Automated Reasoning

Davide Venturelli[‡,*], Minh Do[†,], Bryan O'Gorman[‡,×], Jeremy Frank[†], Eleanor Rieffel[†], Kyle E. C. Booth[+], Thanh Nguyen[⋆], Parvathi Narayan[±], Sasha Nanda[∓]**

[†]Planning and Scheduling Group, NASA ARC; [‡]Quantum Artificial Intelligence Laboratory, NASA ARC;
[*]USRA Research Institute for Advanced Computer Science, USA; [**]Stinger Ghaffarian Technologies, Inc., USA;
[+]Dept. of Mechanical & Industrial Eng., University of Toronto, Canada; [⋆]Computer Science Dept., Dartmouth College, USA;
[±]Computer Science Dept., Washington University in St. Louis, USA;
[∓]Physics Department, Caltech, USA [×]Depts. of Chemistry and EECS, University of California, Berkeley, USA

Abstract

Quantum computing is an information processing paradigm with the potential to solve certain problems faster than any algorithm running on classical computer architectures. In the next few years, new processors will be developed that support quantum computations exceeding the simulation ability of even the largest classical computer systems. A number of academic and industrial groups are developing prototypes of such devices, also known as NISQ (Noisy Intermediate Scale Quantum) processors. Much as software must be compiled to run on classical computers, quantum algorithms must be compiled to take into account the constraints of particular NISQ devices. Especially in these early prototypes, algorithm performance degrades with runtime due to noise; for this reason, minimizing the runtime of the compiled algorithm (which is represented by a "quantum circuit") is critical. We describe a software framework to enable an automated reasoning approach to Quantum Circuit Compilation for NISQ architectures (QCC-NISQ), and our current implementation of it as part of software suite for automated, architecture-aware, compilation for emerging quantum computers. The key components of this suite are a circuit synthesizer, a QCC solver, and a visualizer. These tools provide critical support for the continued development of practical quantum computers and research into quantum algorithms.

1 Introduction

Quantum computing is an emerging computational paradigm with the potential to solve certain problems faster than any algorithm running on classical computer architectures. The breadth of quantum computing applications will become clearer in the next few years as new processors are developed that support quantum computations exceeding the simulation ability of even the largest classical supercomputers. The emerging gate-model *noisy, intermediate scale quantum* (NISQ) processor units, currently in the prototype phase, are *universal* in that, once scaled up, they can run any quantum algorithm.

Much as software must be compiled to run on classical computers, quantum algorithms, also referred to as *logical quantum circuits*, must be compiled to take into account the constraints of particular NISQ devices. Especially in these early prototypes, algorithm performance degrades with runtime due to noise; for this reason, minimizing the runtime of the compiled circuit is critical. Current NISQ architectures have geometric limitations (e.g., connectivity), the specifics of which vary from processor to processor. In this paper, we concentrate on the problem of producing optimally-compiled circuits given the geometric limitations of the processor. Variants of this problem are known as the qubit mapping, qubit routing, qubit allocation, and qubit movement problem. We refer to this problem as Quantum Circuit Compilation for NISQ architectures (QCC-NISQ).

We focus on solid-state architectures based on superconducting quantum bits (qubits), which are among the most advanced NISQ processors. As one example of a geometric limitation, the planar architecture of these processors means that quantum operations can be carried out only between nearest-neighbor locations (qubits). A variety of small superconducting processors, with varying architectures, already exist. Such processors include the 20-qubit IBMQ20 by IBM made available through the Q-Network (IBM Corp. 2018); the 8-, 16-, and 19-qubit chips by Rigetti Computing (Rigetti Computing 2018), and the 72-qubit Bristlecone processor unveiled recently by Google (Google AI Blog 2018). The 50-qubit Intel Tangle Lake chip (Intel Corp. 2018) and a new IBM 50-qubit device are under test and evaluation. Other commercial players in the superconducting NISQ race include Alibaba (Alibaba Cloud 2018) and Quantum Circuits Inc. (Ofek et al. 2016).

In this paper, we detail our software suite based around applying AI Planning, aided by Constraint Programming (CP), to solve the QCC-NISQ problem. Our suite targets: (1) multiple gate-model quantum computing hardware platforms built by different companies (specifically, at the time of writing, Google, Rigetti, and IBM), and (2) different combinatorial optimization problems that can be solved with specific quantum algorithms such as the quantum alternating operator ansatz algorithm (QAOA) (Hadfield et al. 2019) (e.g., Max-Cut, graph coloring, and job shop scheduling). While we have previously published technical details on how we model and solve the QCC-NISQ problem, in this paper we report not only on the actual QCC-NISQ solver, but also on our software suite in its entirety: its components, engineering, and deployment in this promising application area for AI technologies.

2 QCC for NISQ Devices

In the circuit model of quantum computation, a quantum algorithm is expressed conceptually as a *logical quantum circuit*, consisting of a series of quantum operations called *quantum logic gates*. *Quantum processors* are physical devices that implement these quantum logic gates so that the desired quantum operations can be carried out on the quantum states stored in the qubits. In simple cases, the quantum logic gates directly correspond to *physical* quantum gates on the quantum processor, but more typically the processor has physical constraints that prevent a quantum logic circuit, describing the desired algorithm, from being directly implemented.

At a high level, these constraints can be classified into two types: (1) gate set constraints (i.e., those that specify the set of logic gates the processor is capable of applying), and (2) geometric constraints (i.e., those that specify upon which sets of qubits the available logic gates can be applied, limited by, for example, processor connectivity). Although these constraints differ among quantum processors, quantum algorithms can be re-expressed respecting the processor constraints with polynomial overhead in the number of gates (Brierley 2017). As such, for theoretical algorithmic work, the design of logical quantum circuits without concern for the implementation constraints of physical devices is sufficient. However, to implement a quantum algorithm on an actual device, these constraints must be efficiently addressed to take full advantage of NISQ processors.

In this work, we focus on a particular approach to addressing geometric constraints associated with processor connectivity. The approach maps logical qubits to physical qubits on the processor and iteratively updates the mapping through the insertion of additional gates in the course of the computation so as to enable the logical operations to be implemented respecting the physical contraints. This problem is often referred to as "quantum compilation," though quantum compilation usually involves addressing gate set constraints as well as geometric (e.g., connectivity) constraints. Another simple constraint is that gates involving the same qubit cannot be executed in parallel. A generalization of this constraint is a "cross-talk" constraint that may prevent gates in physical proximity from being executed at the same time (Booth et al. 2018).

QCC-NISQ frequently requires adding supplementary operations supported by the hardware to those specified in the idealized circuit. Current superconducting quantum processors have planar architectures with connections only between *nearest-neighbor* locations (qubits), resulting in restrictions as to where gates can be applied. Specifically, a gate can operate only on qubit states located on adjacent qubits on the chip. To compensate for the nearest-neighbor limitation, *swap* gates can move qubit states between connected qubits to reach a configuration where the desired gate, specified in the idealized circuit, can be applied. Current quantum computational hardware suffers greatly from *decoherence* (akin to noise), which degrades the *fidelity* of the computation (Bishop 2017). In NISQ processors, decoherence is intimately linked to the duration of the executed circuit that carries out the quantum computation, so it is critical to minimize the duration of compiled circuits. Thus, compilation is challenging due to: the parallel execution of gates with different durations, the planar or quasi-planar topology of the qubit locations on the chip, the ordering constraints from the original idealized circuit, as well as additional constraints such as cross-talk.

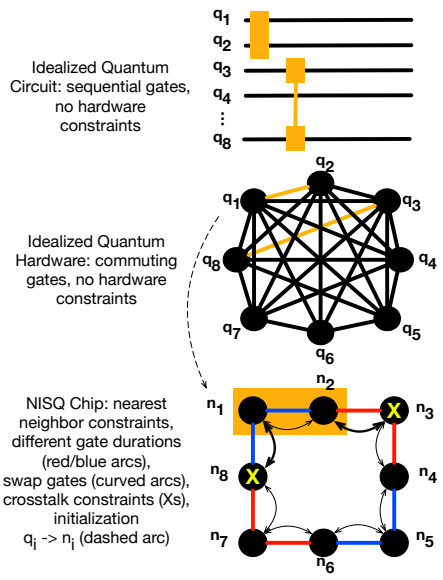


Figure 1: Pictorial view of QCC-NISQ concepts. At the highest level, an *idealized quantum circuit* specifies a sequence of quantum logical gates over *qubit states* that solves a specified problem (top). The idealized quantum circuit could conceptually be implemented on fully-connected quantum hardware in which gates can be carried out between all pairs of *physical qubits*, which is depicted here by a fully-connected graph. Qubit states in an idealized quantum circuit are mapped onto physical qubits in the fully-connected architecture. At this level, gates are specified between physical qubits and can be executed in parallel if they do not involve the same qubit (middle). In an actual NISQ chip, physical gates can only be carried out between a subset of pairs of qubits, usually nearest neighbors in a 1D or 2D array. To carry out 2-qubit gates specified in idealized quantum circuits between qubits that are not connected, *swap* gates are added to route logical qubit states to physical qubits that are connected so that the desired gates can be applied (bottom).

Example: Figure 1 shows a concrete QCC example requiring gate operations $g_A = \mathcal{G}(q_1, q_2)$ and $g_B = \mathcal{G}(q_3, q_8)$. At the top, the algorithm is specified, as is typically done in the gate-model quantum computing literature, as an idealized quantum circuit, with sequential specifications of 2-qubit gates over qubit states. There are no hardware constraints; further, some ordered pairs of gates in the idealized circuit may *commute* (i.e., could execute in arbitrary order, even simultaneously, and still produce correct results). A fully-connected quantum hardware, with no hardware constraints

except that two gates involving the same qubit cannot be carried out at the same time, is represented in the middle as a complete graph connecting all possible pairs of qubits. The gates g_A and g_B are indicated as a subset of this graph (yellow edges). A corresponding real-world NISQ chip has gates between only a small subset of qubit pairs (bottom). For instance, the gates acting respectively on qubit states q_1 and q_2 and on q_3 and q_8 can be executed, even concurrently, when these states are located at pairs qubits connected in the chip. Furthermore, on the actual chip, gate execution durations may differ depending on the actual location they are executed, indicated by red or blue edges in the chip (bottom). Cross-talk constraints preclude operations on qubits that are physically located nearby an activated gate. A 2-qubit gate operating on the quantum state residing on qubit n_1 and n_2 will prevent any other gate operating concurrently on n_3 or n_8; this is shown by the yellow Xs (bottom). In this example, we assume the initial assignment of quantum states to qubits allocates q_i to qubit n_i, shown as a dashed line. We see that g_A can be applied immediately, but g_B involves two qubit states that are not mapped to nearest neighbors. One solution is for the QCC software to add additional "swap" gates. For instance, $\texttt{swap}(n_1, n_8)$ and concurrently $\texttt{swap}(n_2, n_3)$, shown in bold (bottom), will bring the states q_3 and q_8 to qubits where g_B can be applied. This highlights that the QCC procedure can also determine the *initial assignment* task (i.e., decide the initial qubit location for each qubit state on the idealized hardware) to optimize the gate schedule. For example, the QCC solver could decide to initialize q_3 and q_8 on two adjacent qubits on the actual hardware chip, avoiding all swaps for executing the idealized circuit in question.

To summarize, to compile from the idealized quantum circuit illustrated at the top of Figure 1 consisting of two gates $\{\mathcal{G}(q_1, q_2), \mathcal{G}(q_3, q_8)\}$ and no hardware constraints, into the sequence of (parallel) gates that can be executed in the actual NISQ hardware chip at the bottom of Figure 1, the QCC solver will: (1) first find the initial locations for the four qubit states q_1, q_2, q_3, and q_8 on the NISQ chip; then (2) add auxiliary swap gates to bring the two pairs (q_1, q_2) and (q_3, q_8) to adjacent physical qubits that are connected and execute the required operations; and (3) schedule all the gates, each with possibly different duration, to execute in parallel in the shortest amount of time, while obeying all hardware constraints such as cross-talk constraints.

Existing Work on QCC: Since the development of NISQ superconducting processors, there has been development of software libraries to synthesize and compile quantum circuits from algorithm specifications (Wecker and Svore 2014; Smith, Curtis, and Zeng 2016; Steiger, Häner, and Troyer 2018; Barends and others 2016), including work explicitly addressing theoretical bounds on the overhead introduced by swap gates (Beals and others 2013; Brierley 2017; Bremner, Montanaro, and Shepherd 2017). Recently, it was proven that the QCC problem and its common variants are NP-Complete (Botea, Kishimoto, and Marinescu 2018). (Bhattacharjee and Chattopadhyay 2017) investigated approaches with off-the-shelf MILP solvers, such as Gurobi, for solving QCC. (Guerreschi and Park 2018; Zulehner and Wille 2018) developed heuristics that address NISQ constraints while minimizing the number of required swap gates. The initial assignment task has been recently addressed in (Paler 2019). Policies for compilation in chips with variable performance parameters among different qubits and gates have been studied in (Tannu and Qureshi 2018), as well as in (Murali et al. 2019). In (Oddi and Rasconi 2018; Rasconi and Oddi 2019), the authors present heuristics (greedy randomized search, and genetic algorithms) specifically designed to solve the QCC Max-Cut benchmark set that was introduced in (Venturelli et al. 2018). In (Booth et al. 2018), CP is explored as an alternative and complementary approach to the temporal planning methods introduced in (Venturelli et al. 2018). Other methods are proposed in (Li, Ding, and Xie 2018) and in (Childs, Schoute, and Unsal 2019), where the crosstalk-free compilation of circuits is handled heuristically on benchmarks for the IBM chip. In (Khatri et al. 2018), an approach based on iteratively learning a sequence of native gate implementing a target unitary is introduced, solving the problem of compilation with that of gate-synthesis at the same time, for small shallow circuits. Another recent learning approach in (Jones and Benjamin 2018) converts the approximate compilation problem into an auxiliary quantum variational algorithm native on the hardware. In (Nash, Gheorghiu, and Michele 2019) and (Kissinger and Meijer-van de Griend 2019) the QCC problem is solved heuristically for circuits consisting of CNOT gates only.

3 Automated Reasoning for QCC-NISQ

Our approach to solve the QCC-NISQ problem is to deploy a general-purpose software suite that leverages *model-based* approaches, including *temporal planning* and *constraint programming* (CP) for the solution of the actual combinatorial problem. Since quantum computing is still a relatively new paradigm, different quantum hardware architectures and algorithms running on them are introduced and revised frequently, with no current clear winner. Therefore, instead of a machine-specific QCC tool, there is a tremendous benefit in developing a general-purpose QCC software suite that is capable of addressing different hardware architectures, different quantum algorithms running on them, and different optimization problems that can be solved by those algorithms. Furthermore, our approach is a very attractive option because: (1) declarative model adjustment (e.g., through a declarative planning model) can adapt quickly to revised hardware designs and constraints, and can also cover a wide range of hardware architectures; (2) existing hardware from various companies are still limited in size, thus even general-purpose algorithms can quickly find good, sometimes proven optimal, solutions.

Why AI Planning? In planning, a planner searches for a set of actions that can be executed in sequence to achieve the pre-defined set of goals, while satisfying all domain constraints. In model-based temporal planning, specifically PDDL-based planning, actions can have different durations and can be executed in parallel. We choose planning to be the center piece of the QCC solver suite because:

- The action model and plan constraints in PDDL planning can be used to describe gate operation naturally, capturing chip layout, gate duration, and domain constraints such as "cross-talk". This flexibility lets us model various chips from all of the groups mentioned above developing NISQ chips.
- Any new adjustment or hardware updates from the manufacturer can also be reflected easily in planning model updates, without extensive writing of new software.
- The default objective function of optimizing makespan for most temporal planners fits well with quantum decoherence (discussed in the previous section).
- There are multiple off-the-shelf open-sourced PDDL temporal planners that have been tested through multiple International Planning Competitions (IPC), providing a rich set of algorithms, ranging from exact to anytime, to test on the NISQ-QCC problem.

For more details on the AI planning approach to NISQ-QCC, see (Venturelli et al. 2018).

Why Constraint Programming? Constraint Programming (CP) is a paradigm for modeling and solving combinatorial optimization problems, leveraging a diverse set of techniques from fields including operations research and artificial intelligence (Rossi, Van Beek, and Walsh 2006). CP is more general than other discrete optimization paradigms, such as integer programming, as it allows variable types beyond integer and continuous (e.g., interval and set variables), and drops the linearity on the constraints and objective function. CP complements PDDL-based planning for the QCC software suite for a number of reasons, including:

- The scheduling characteristics of QCC (i.e., gate durations, precedence constraints, makespan minimization, and unary qubit capacity) are readily modeled within modern CP solver software.
- Solutions found by PDDL-based planning or a heuristic method can be used as a starting point for the CP search, leading to subsequently better solutions.
- The CP search is an exact algorithm, ensuring that, given enough runtime, the optimal quantum circuit compilation will be found. Additionally, modern CP solvers provide anytime bounds on solution quality in the form of an optimality gap.

For more details on the CP approach to optimize circuits together with planners, see (Booth et al. 2018).

4 System Architecture

The objective of the whole framework is to provide to quantum computing researchers the ability to efficiently deploy executables of specific quantum algorithms by keeping control over the tradeoffs imposed by the different choices related to what strategy to adopt to solve the QCC-NISQ problem. While the key component of our suite is the QCC solver that does the compilation, multiple additional tools complement the software suite.

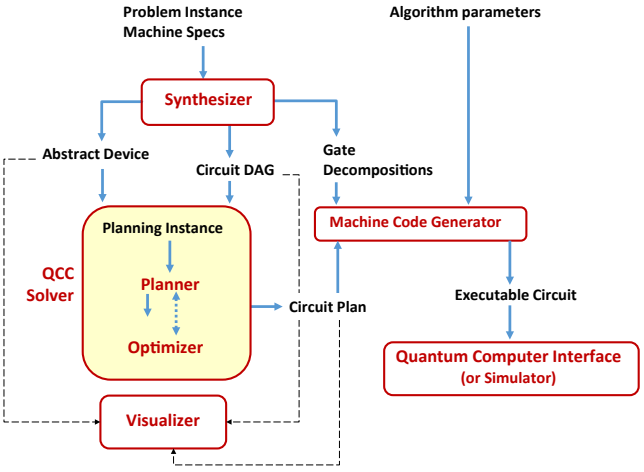


Figure 2: The software suite architecture. Arrows indicate input-output relationship between data structures (black text) and software components (red text). The visualizer can be optionally available (dashed lines) to display the data structure that interacts with the QCC solver in a compelling way. See text of section 4 for details.

Figure 2 shows the architecture of our software suite, with the initial inputs provided by the users consisting of two main components: the *problem instance* containing details on the problem for which we attempt to solve using a quantum algorithm, and the *machine configuration* specifying the details of the gate-model quantum chip (e.g., physical layout, gate durations, constraints). (A third input, the *algorithm parameters*, concerns the setting of the classical parameters of the gates once they are already compiled, and will be discussed in connection with the generation of the executable.)

These inputs are provided to the *Synthesizer* that generates three different outputs (described in later sections), that can be fed into two other components: the *QCC Solver* and the *Machine Code Generator (MCG)*. The *QCC Solver* generates planning problems in the standard Planning Domain Definition Language (PDDL) and generates plans in the standard IPC format, which can then be parsed in a *circuit plan* representation or fed into the CP solver (the "*optimizer*") to generate a better quality solution.

The last two components are: the *Visualizer*, which can show graphically the machine logical layout, the high-level problem specifications, and the QCC solutions; and the *Machine Code Generator*, which maps the QCC solution and information on how to synthesize gates onto a particular hardware architecture to produce the machine-specific executable file (e.g. a python script) that can submit a job on the cloud to a quantum NISQ device through its public API.

In the rest of this section, we will describe in more detail the different components and their inputs and outputs.

(Circuit) Synthesizer (CS): the CS is in charge of two main tasks: (1) generate the low-level gate synthesis (*gate decom-*

positions) to act as one input to the MCG; and (2) generate the inputs for the QCC Solver (*circuit DAG* and *abstract device*).

The first task is meant to decompose the abstract gates, which appear in the idealized circuit of the quantum algorithm, into elementary gates supported by the hardware, whose duration in nanoseconds is known at all possible locations in the chip. This decomposition is known as the *gate-synthesis* problem. It is non-trivial in the general case, but for many quantum algorithms and for standard universal elementary gate sets, optimal decompositions are known. Currently, the CS implementation just consists of a lookup library of known decompositions into elementary gates [1]. This decomposition library has been found by various methods, including proven optimal results (Vatan and Williams 2004). The CS assigns a total duration (in standardized clock units) to each possible logical gate, to inform the second component of the module that will have to generate the input files to the QCC solver.

The second task is to instantiate an *abstract device* software object, which is representing the topology of the hardware but is aware only of the different types of gates that need to be scheduled (including swaps), which are represented as edges, and their duration obtained through synthesis (the edge weight). The abstract device includes information about crosstalks/simultaneity constraints in the form of an extra graph whose vertices are the edges of the hardware graph and whose edges indicate impossibility of concurrent gate operations using the corresponding edges of the hardware graph. Fig. 1 (bottom) is a pictorial representation of some informations contained in the *abstract device*. While the properties described in the Abstract Device instance are tailored to the QCC solver capabilities, the device class should be used by different solvers. For instance, if a non-temporal method is used as a solver instead of temporal planning, the gate durations could be discarded.

Finally, the CS composes a Directed Acyclic Graph (Circuit *DAG*) representation of the problem instance, which takes care of defining the partial ordering rules of the gates composing the circuit. The vertices of the DAG correspond to gates on specified qubits and the arcs correspond to precedence constraints; operations that are incomparable by this relation can be scheduled in any order relative to each other. This freedom arises naturally in quantum computation for quantum gates that "commute" with each other, i.e., produce the same effect regardless of the order in which they are applied. But can also be used in the context of heuristic algorithms that try to balance the effectiveness of the logical circuit and cost of implementing it as a physical circuit. The DAG includes only the synthesized two-qubit gates that are necessary for the logical description of the algorithm.

QCC Solver: The QCC solver takes the SC input and produces internally the PDDL files that represent the *planning instances*. The ultimate objective of this module is to output the *circuit plan*, which is the compiled representation of the target algorithm. The following options are currently implemented:

- *With/without cross-talk constraints:* specifying whether or not the underlying hardware has cross-talk constraints between adjacent qubits; see Figure 1 (bottom).
- *With/without qubit initialization:* specifying whether or not the QCC Solver should decide (as part of the planning objective) the mapping of specific qubits to quantum states at the beginning of the algorithm; see Figure 1 (dashed line, middle-bottom).
- *Single or multiple phases:* specifying if the idealized circuit should be executed multiple times in sequence (see Section 2). This is particularly important for QAOA circuits, which require insertion of alternating "phase-separation" and "mixing" sets of gates to increase the accuracy of the solution returned by the algorithm. Requirements on running multiple phases will lead to CS generating PDDL files with different sets of actions and goals.

The planning instance is processed by a *Planner* to obtain a temporal plan, which is a sequence of gates to be executed on the designated hardware architecture. Specifically, at the moment we use two different macro-approaches to generate the final plan[2]:

- Use off-the-shelf temporal planners that can take the standard PDDL input. The following planners have been used: LPG, TFD, POPF, CPT, and SGPlan; all previous winners at different IPC. The performance of different individual planners is reported on in (Venturelli et al. 2018).
- Use a combination of planning and CP in a hybrid setting where plans found by any temporal planner can then be used to warm-start (i.e., seed) the CP model that in turn is solved by the commercial software CP Optimizer to find a new, better quality plan. The evaluation of this approach is described in (Booth et al. 2018) [3].

Results on the use of the QCC solver with the various variants for MaxCut QAOA are presented in (Venturelli et al. 2018) and (Booth et al. 2018). The QCC solvers have been also configured for tests on Graph Coloring, which are currently being performed. The versatility of our approach utilizing off-the-shelf domain-independent PDDL temporal planners is reflected in the ability to quickly test machine models from different companies (Rigetti, Google, IBM) at different scales and diffferent chip layouts, for different set of gate operations, gate durations, and gate constraints.

Figure 3 shows the visualizer displaying a single phase of QAOA to solve a Maxcut problem (referred to below as Maxcut-QAOA) on the Google Bristlecone chip. The figure shows the following components:

- *Goal Graph:* at the top-left corner, the goal graph shows the gates to be scheduled (colored edges) according to the

[1] Elementary gates might include CZ, CNOT, iSWAP, the PhasedXPowGate of Google's chip and single qubit rotation gates.

[2] The plans generated either by planner or CP Optimizer are validated by the official plan validator software VAL before passing on to the next component.

[3] The CP model is not automatically generated.

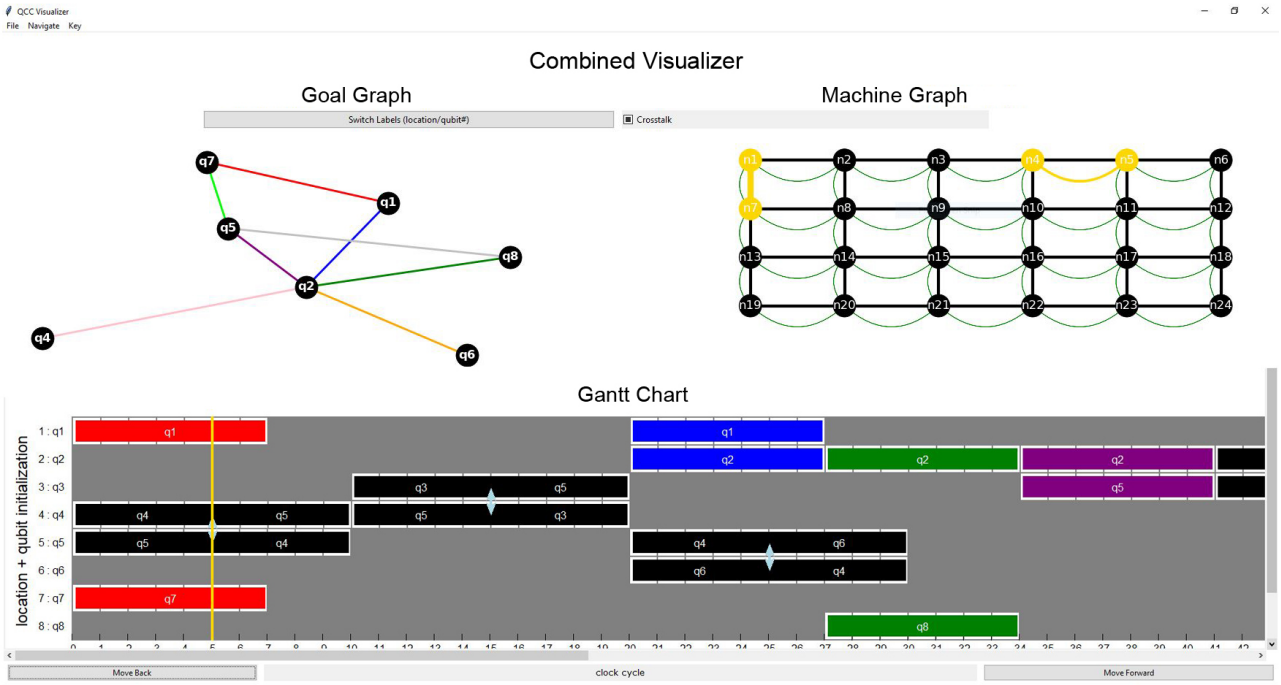


Figure 3: Our visualizer interface with: Goal Graph (top-left), Machine Graph (top-right), and QCC plan (bottom).

idealized quantum circuit. For Maxcut-QAOA, the goal graph is identical to the actual graph that we want to cut. The goal graph contains essentially information related to the *circuit DAG* (Fig. 2).

- *Machine Graph:* at the top-right corner, the machine graph represents the underlying logical layout of the NISQ processor target (in this case, a portion of Google's Bristlecone chip). Multiple connections between a given pair of qubits represent different types of gates that can be applied to the nearest-neighbor connected qubits. Each type of gate has a different duration. The machine graph contains essentially information related to the *abstract device* (Fig. 2).
- *QCC plan:* this shows the temporal schedule of gate operations as a Gantt chart, each with its starting time, duration, and the qubits involved in that particular gate operation. The gates are color coded to match the high-level goals (see the matching edge color on the *Goal Graph* described above). Sliding over the timeline of this Gantt-chart representation will also highlight on the machine graph the qubit set and the active gates operating on that set, at the timepoint selected on the timeline of the plan. For reference, in Figure 3, the slider has been set on time slot number 5. The QCC plan contains essentially the same information related to the circuit plan (Fig. 2).

For the case of graph coloring, the QAOA algorithm logical gates require multiple kind of two qubit gates, as explained in (Hadfield et al. 2019). A more advanced visualizer that is able to show the various steps by using different colors, graphic notations and animations is under development.

Machine Code Generator (MCG): this component outputs the machine instructions that perform the compiled algorithm on the target quantum processor or in a simulator (the *Executable circuit*). The MCG integrates the abstract QCC solution (the *circuit plan*) with the gate synthesis instructions generated by the CS, and sets the parameters that are required for the execution of the algorithm, which are given as inputs (*Algorithm parameters*). More specifically, it unwinds the duration abstraction of the gates that has been scheduled and replaces the composite gate with the code that activates the exact sequence of native gates on the chip with the correct parameters. For instance, for the case of QAOA, these includes the angles for the alternating unitary transformations that compose the algorithm. The algorithm specifications not only set the individual gate parameters, but also many control-loop policies for hybrid classical-quantum computation. Examples include the measurement and accuracy evaluation functions that determine whether the algorithm needs to be iterated one more time with different parameters after execution.

Current deployment and implementation details: The software suite we describe ultimately will be integrated with the general software framework for programming quantum processors QuaSar (NASA's *Quantum user-assisting Software for applied research*. QuaSar is a high-level back-end and frontend system, soon to be released, which support transparent inter-operability between code written for most quantum computers that released APIs, including Cirq, Rigetti Computing PyQuil as well as QASM 2.0.

The current implementation of the CS, and of the QCC Solver is in Cirq (Google AI 2018)[4], an opensource framework that conveniently abstracts several aspects of temporal manipulation of operations in quantum circuits. This excludes the actual implementation of the planner and the CP optimizer, that varies case by case (they are considered external black boxes to be interfaced). The output of the MCG is a Cirq schedule object; such an object can be run directly on Google's hardware or converted by QuaSar to the common quantum circuit format QASM for compatibility with other hardware providers.

Relation to existing work on software frameworks: as reviewed in (LaRose 2019), most quantum computing companies have released software frameworks that allow end-to-end deployment of quantum algorithms. These packages could be as simple as generic wrappers around machine instruction languages, or include suites for handling specific aspects. For instance, Xanadu's PennyLane focuses on facilitating aspects of development of (quantum) machine learning algorithms (Bergholm et al. 2018), Microsoft's Quantum Development Kit (Svore et al. 2018) focuses on resource estimation for fault-tolerant quantum computers, and Zapata's algo2qpu (Sim et al. 2018) focuses attention on the hybridization of variational algorithms with classical optimization techniques.

Our framework focuses on facilitating the optimization of the compilation by allowing the user to use alternative compilation strategies, possibly hybridizing them, and allowing the inspection of the results of the compilation.

5 Conclusions and Future Work

We have introduced a framework for applying AI Planning to the QCC-NISQ problem, and developed a software suite that implements components of that framework. The framework and suite are completely general and can be used to target quantum computing hardware devices of different types and by different companies. Different planning algorithms, complemented by hybrid algorithms using Constraint Programming, can be used to compile a qunatum circuit to a specified hardware device. The compiled circuit can be visualized, enabling users to understand how the compilation of the circuit interacts with the constraints of the device. The resulting compiled circuit can then be run on actual hardware, or simulators therof, by different companies. The flexibility of declarative AI planning models allows us to solve the QCC-NISQ problem for these diverse hardware architectures, and to evolve our solutions as the details of the hardware evolve.

The suite integrates several planners (LPG, POPF, TFD, CPT, and SGPlan), together with IBM's CP Optimizer suite, the best performing commercial software of its kind. We have generated tens of thousands of instances of the QCC-NISQ problem based on pairing QAOA for MaxCut on random graphs with quantum computer architectures of varying size and constraints. Performance comparisons of different planners and hybrid approaches are reported in (Booth et al. 2018; Venturelli et al. 2018) and benchmark instances are available online[5] and have already been used to test other QCC-NISQ solver approaches (e.g. in (Oddi and Rasconi 2018)). Future hybrid efforts are under development, including the use of different planners hybridized with LPG, which can take as input a "seed" plan. While CP improves on seed plans, it requires a separate modeling effort; hybridizing multiple planners uses multiple algorithms but requires no additional modeling effort.

We claim that a general-purpose software suite built on declarative planning algorithms and constraint programming is a promising approach to addressing the constraints of NISQ devices. A highly optimized problem-specific tool may perform better in some cases, but at the cost of significant engineering effort that cannot be reused for new problems. Our model-based, automated reasoning approach is very flexible with respect to features of the hardware graph, including irregular structures, as often arise from manufacturing imperfections. The ease and expressiveness of PDDL modeling facilitates the inclusion of additional features that are characteristic of quantum computer architectures, such as the ability to *quantum teleport* quantum states across the chip (Copsey et al. 2003), providing more flexibility than mere nearest-neighbor swap.

The modularity of the software suite we introduced allows it to be improved iteratively, giving a foundation for future work on the application artificial intelligence methods to quantum computing. Here, we focused on the makespan of the compiled circuit as the objective function to minimize, but data from ongoing experimental work will likely yield more sophisticated quantities to optimize. We are also actively looking at compiling circuits corresponding to QAOA for different combinatioral optimization problems (e.g. job shop scheduling), many of which involve new types of multi-qubit gates with different characteristics from the simple ones used in illustrative examples in this paper and in other introductory publications. Another avenue of extension of our work is to generalize the NISQ-QCC problem and our software suite to the specificities of quantum processors that are dramatically different than superconducting NISQ devices. For instance, soon to made available Ion-trap processors such as the one of IonQ (Nam et al. 2019) or of Honeywell International Inc. feature a number fully-connected cells of qubits (where qubits interact through a different set of native gates than the ones of superconducting processors) which are in communinication to each other through the ability to swap quantum information via photonic interfaces. In a future work, we intend to provide examples for different architectures including sample code. We plan to make the architecture available (to be interfaced with a planner and optionally a CP optimizer) as an opensource package.

Finally we believe that our approach should be of great interest to the community developing low-level quantum compilers for generic architectures (Steiger, Häner, and Troyer 2018; Häner et al. 2018),

[4] https://github.com/quantumlib/Cirq

[5] https://ti.arc.nasa.gov/m/groups/asr/planning-and-scheduling/QCC_ICAPS18.zip.

to designers of machine-instructions languages for quantum computing (Smith, Curtis, and Zeng 2016; Bishop 2017), and to developers of unifying frameworks for quantum computing software toolchains and interface to solvers (McCaskey et al. 2018).

Acknowledgement: The authors would like to acknowledge support from the NASA Advanced Exploration Systems program, NASA Academic Mission Services (NNA16BD14C) and the NASA Ames Research Center. B.O. was supported by a NASA Space Technology Research Fellowship.

References

Alibaba Cloud. 2018. *Alibaba Cloud and CAS Launch One of the Worlds Most Powerful Public Quantum Computing Services*. http://engineering.purdue.edu/~mark/puthesis.

Barends, R., et al. 2016. Digitized adiabatic quantum computing with a superconducting circuit. *Nature* 534(7606):222–226.

Beals, R., et al. 2013. Efficient distributed quantum computing. *Proceedings of the Royal Society A*. 469(2153):767 – 790.

Bergholm, V.; Izaac, J.; Schuld, M.; Gogolin, C.; and Killoran, N. 2018. Pennylane: Automatic differentiation of hybrid quantum-classical computations. *arXiv:1811.04968*.

Bhattacharjee, D., and Chattopadhyay, A. 2017. Depth-optimal quantum circuit placement for arbitrary topologies. *arXiv:1703.08540*.

Bishop, L. S. 2017. QASM 2.0: A quantum circuit intermediate representation. *Bulletin of the American Physical Society* 62.

Booth, K. E. C.; Do, M.; Beck, J. C.; Rieffel, E.; Venturelli, D.; and Frank, J. 2018. Comparing and integrating constraint programming and temporal planning for quantum circuit compilation. In *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling (ICAPS2018)*, 366–374.

Botea, A.; Kishimoto, A.; and Marinescu, R. 2018. On the complexity of quantum circuit compilation. In *Proceedings of the Eleventh International Symposium on Combinatorial Search (SoCS2018)*, 138–142.

Bremner, M. J.; Montanaro, A.; and Shepherd, D. J. 2017. Achieving quantum supremacy with sparse and noisy commuting quantum computations. *Quantum* 1:8.

Brierley, S. 2017. Efficient implementation of quantum circuits with limited qubit interactions. *Quantum Information & Computation* 17(13-14):1096–1104.

Childs, A. M.; Schoute, E.; and Unsal, C. M. 2019. Circuit transformations for quantum architectures. *arXiv:1902.09102*.

Copsey, D.; Oskin, M.; Impens, F.; Metodiev, T.; Cross, A.; Chong, F. T.; Chuang, I. L.; and Kubiatowicz, J. 2003. Toward a scalable, silicon-based quantum computing architecture. *IEEE Journal of Selected Topics in Quantum Electronics* 9(6):1552–1569.

Google AI Blog. 2018. *A Preview of Bristlecone, Googles New Quantum Processor*. https://ai.googleblog.com/2018/03/a-preview-of-bristlecone-googles-new.html.

Google AI. 2018. *Cirq, a python framework for creating, editing, and invoking Noisy Intermediate Scale Quantum (NISQ) circuits.* https://github.com/quantumlib/Cirq.

Guerreschi, G. G., and Park, J. 2018. Two-step approach to scheduling quantum circuits. *Quantum Science and Technology* 3(4):045003.

Hadfield, S.; Wang, Z.; O'Gorman, B.; Rieffel, E. G.; Venturelli, D.; and Biswas, R. 2019. From the quantum approximate optimization algorithm to a quantum alternating operator ansatz. *Algorithms* 12(2):34.

Häner, T.; Steiger, D. S.; Svore, K.; and Troyer, M. 2018. A software methodology for compiling quantum programs. *Quantum Science and Technology* 3(2):020501.

IBM Corp. 2018. *IBM Q Network*. https://www.research.ibm.com/ibm-q/network/.

Intel Corp. 2018. *The future of quantum computing is counted in qubits*. https://newsroom.intel.com/news/future-quantum-computing-counted-qubits/.

Jones, T., and Benjamin, S. C. 2018. Quantum compilation and circuit optimisation via energy dissipation. *arXiv:1811.03147*.

Khatri, S.; LaRose, R.; Poremba, A.; Cincio, L.; Sornborger, A. T.; and Coles, P. J. 2018. Quantum assisted quantum compiling. *arXiv:1807.00800*.

Kissinger, A., and Meijer-van de Griend, A. 2019. CNOT circuit extraction for topologically-constrained quantum memories. *arXiv:1904.00633*.

LaRose, R. 2019. Overview and comparison of gate level quantum software platforms. *Quantum* 3:130.

Li, G.; Ding, Y.; and Xie, Y. 2018. Tackling the qubit mapping problem for NISQ-era quantum devices. *arXiv:1809.02573*.

McCaskey, A.; Dumitrescu, E.; Liakh, D.; and Humble, T. 2018. Hybrid programming for near-term quantum computing systems. *arXiv:1805.09279*.

Murali, P.; Baker, J. M.; Abhari, A. J.; Chong, F. T.; and Martonosi, M. 2019. Noise-adaptive compiler mappings for noisy intermediate-scale quantum computers. *arXiv:1901.11054*.

Nam, Y.; Chen, J.-S.; Pisenti, N. C.; Wright, K.; Delaney, C.; Maslov, D.; Brown, K. R.; Allen, S.; Amini, J. M.; Apisdorf, J.; et al. 2019. Ground-state energy estimation of the water molecule on a trapped ion quantum computer. *arXiv preprint arXiv:1902.10171*.

Nash, B.; Gheorghiu, V.; and Michele, M. 2019. Quantum circuit optimizations for NISQ architectures. *arXiv:1904.01972*.

Oddi, A., and Rasconi, R. 2018. Greedy randomized search for scalable compilation of quantum circuits. In *Proceedings of the Fifteenth International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR2018)*, 446–461.

Ofek, N.; Petrenko, A.; Heeres, R.; Reinhold, P.; Leghtas, Z.; Vlastakis, B.; Liu, Y.; Frunzio, L.; Girvin, S.; Jiang, L.; et al. 2016. Extending the lifetime of a quantum bit with error correction in superconducting circuits. *Nature* 536(7617):441.

Paler, A. 2019. On the influence of initial qubit placement during NISQ circuit compilation. In *Proceedings of the First International Workshop on Quantum Technology and Optimization Problems (QTOP2019)*, 207–217.

Rasconi, R., and Oddi, A. 2019. An innovative genetic algorithm for the quantum circuit compilation problem. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence (AAAI2019)*, In press.

Rigetti Computing. 2018. *Acorn QPU Properties*. `http://docs.rigetti.com/en/latest/qpu.html`.

Rossi, F.; Van Beek, P.; and Walsh, T. 2006. *Handbook of constraint programming*. Elsevier.

Sim, S.; Cao, Y.; Romero, J.; Johnson, P. D.; and Aspuru-Guzik, A. 2018. A framework for algorithm deployment on cloud-based quantum computers. *arXiv:1810.10576*.

Smith, R. S.; Curtis, M. J.; and Zeng, W. J. 2016. A practical quantum instruction set architecture. *arXiv:1608.03355*.

Steiger, D. S.; Häner, T.; and Troyer, M. 2018. ProjectQ: An open source software framework for quantum computing. *Quantum* 2:49.

Svore, K. M.; Geller, A.; Troyer, M.; Azariah, J.; Granade, C.; Heim, B.; Kliuchnikov, V.; Mykhailova, M.; Paz, A.; and Roetteler, M. 2018. Q#: Enabling scalable quantum computing and development with a high-level domain-specific language. *arXiv:1803.00652*.

Tannu, S. S., and Qureshi, M. K. 2018. A case for variability-aware policies for NISQ-era quantum computers. *arXiv:1805.10224*.

Vatan, F., and Williams, C. 2004. Optimal quantum circuits for general two-qubit gates. *Physical Review A* 69(3):032315.

Venturelli, D.; Do, M.; Rieffel, E.; and Frank, J. 2018. Compiling quantum circuits to realistic hardware architectures using temporal planners. *Quantum Science and Technology* 3(2):025004.

Wecker, D., and Svore, K. M. 2014. LIQ$Ui|>$: A software design architecture and domain-specific language for quantum computing. *arXiv:1402.4467*.

Zulehner, A., and Wille, R. 2018. Compiling SU (4) quantum circuits to IBM QX architectures. *arXiv:1808.05661*.

Advantages and Challenges of Using AI Planning in Cloud Migration

Hongtan Sun, Maja Vukovic, John Rofrano, Chen Lin
IBM T.J. Watson Research Center
1101 Kitchawan Rd,
Yorktown Heights, New York 10598
hongtan.sun@ibm.com, { maja, rofrano } @us.ibm.com, liana.lin@ibm.com

Abstract

Cloud Migration transforms customer's data, application and services from original IT platform to one or more cloud environment, with the goal of improving the performance of the IT system while reducing the IT management cost. The enterprise level Cloud Migration projects are generally complex, involves dynamically planning and replanning various types of transformations for up to 10k endpoints. Currently the planning and replanning in Cloud Migration are generally done manually or semi-manually with heavy dependency on the migration expert's domain knowledge, which takes days to even weeks for each round of planning or replanning. As a result, automated planning engine that is capable of generating high quality migration plan in a short time is particularly desirable for the migration industry. In this short paper, we briefly introduce the advantages of using AI planning in Cloud Migration, a preliminary prototype, as well as the challenges the requires attention from the planning and scheduling society.

Introduction

Automated planning and AI planning have been investigated extensively by researchers and successfully applied in many areas for decades, for example, health care (Cardoen, Demeulemeester, and Beliën 2010), semiconductor manufacturing (Uzsoy, Lee and Martin-Vega 1992), and aviation (Bazargan, M. 2010), to name a few. Meanwhile, attracted by the promise of the scalability, flexibility and potentially lower cost of the resources, more and more enterprises are considering moving their IT infrastructure and applications to Cloud or Hybrid Cloud service platforms, which is called *Cloud Migration* in general (Armbrust et. al. 2010, Khajeh-Hosseini, Greenwood, and Sommerville 2010). Noticing that the discussions of using AI planning in the Cloud Migration are limited both in academia and in industry, in this short paper we identify the advantages and challenges of applying AI planning to Cloud Migration by (i) introducing Cloud Migration and its planning problem; (ii) demonstrate problem feasibility by showing a prototype AI planning model; and (iii) discuss the limits of current model and future research.

Planning in Cloud Migration

Cloud Migration transforms customer's data, application and services from original IT platform, hosted on servers hosted in-house or cloud environment, to one or more cloud environment, with the goal of improving the performance of the IT system while reducing the IT management cost. Generally speaking, enterprise-level Cloud Migration is a complex and usually long running process that requires careful planning.

Cloud Migration includes four major steps: Discovery, Planning, Execute and Validation (Vukovic and Hwang 2016). In the *Discovery* stage, migration experts investigate the current IT system, collect data and identify customer's migration goals. Then migration experts allocate resources and schedule the executions activities, refer to as the *Planning* stage. Next the migration are executed as planned, which is called *Execute* stage. In the last *Valication* stage, all the applications are tested that they are running as expected on the new cloud environment. Due to the complexity of the IT system and IT infrastructure, there may not be clear boundaries between the major steps. It could happen that in the Planning stage some data inconsistency were observed and additional discoveries are performed and the migration execution needed to be re-scheduled or re-planned.

Due to the complexity of migration projects, low tolerance on errors and its heavy dependence on the migration expert's domain knowledge, current practitioners mostly perform migration planning either manually. or using tools manually created runbooks (Transition Manager, Velostrata). For example, in Transition Manager, the user has to upload manually scripts, e.g. groovy scripts (The Apache Groovy Programming Language), and ask the tool to generate runbook for existing wave bundles. Meanwhile, Velostrata Manager would create a .csv format template for the user to manually put tasks in and create the runbook (Creating and modifying runbooks). These planning and replanning approaches rely heavily on the practitioner's previous migration experience and domain knowledge, hence is not scalable.

With the fast evolution of computing speed and machine learning technologies, domain-independent AI planners are becoming more and more powerful (Ghallab, Nau,

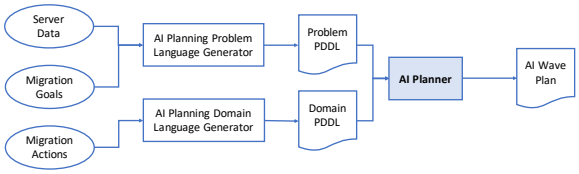


Figure 1: Overview for the prototype AI planner

and Traverso 2004). Better planners emerge and demonstrate their performance and capability every year on the International Planning Competition (ICP) and the International Conference on Automated Planning and Scheduling (ICAPS). As result, taking advantage of the domain-independent planners to automate the planning of Cloud Migration is extremely desirable for migration practitioners.

Prototype AI Planner

In a Cloud Migration planning problem, there are N assets to be migrated. Assets may communicate with each other, for example, an application reads/writes a database, hence causes dependencies between assets and enforces precedence constraints for migration tasks. For instance, if asset A depends on asset B, the migration of asset A has to be done before asset B's migration. The goal of migration planning is to allocate resources and create sequence of tasks to be executed. In the case of enterprise level migration, the execution should be performed in a limited time window to minimize potential business disruption.

From application point of view, the main step in developing an AI planner based on domain-independent planner is to formulate the planning problem in Cloud Migration as a planning problem for the planner. In one of our prototype AI planner, a simplest scenario, in which only migration of physical servers and virtual machines are considered, is modeled as Domain file and Problem file using Planning Domain Definition Language (PDDL). The objects in the domain file are server and wave. Each server has is assigned a numeric value called 'effort hours', which represents the cost of migrating this server. Each wave is assigned a numeric value called 'effort hour limit', which enforces a capacity constraint for the number of servers to be migrated in each wave. The goal is to migrate all the servers without violating the capacity constraint.

A planner supported by Metric-FF planner is developed to test the performance and a graphical UI is created for users to upload spreadsheet containing server's information (Jackson, Rofrano, Hwang, and Vukovic 2018). In particular, translation engines are developed to generate the Domain.pddl file and Problem.pddl file automatically. When there are limited number of servers, the planner finds solution in a few seconds. However, when tested with 500 servers, the planner did not find any solution in 2 hours. Figure 1 shows an overview of the prototype planner.

Challenges and Future Research Directions

In conclusion, in this short paper, the Cloud Migration process is investigated and a prospective research direction is identified around using domain-independent AI planner in Cloud Migration Planning. Automate migration planning is desirable for practitioners from both the cost perspective and the quality consideration. It also brings in new research topics. Some of them are listed as following.

- Optimize the modeling of migration planning problem so that the domain and problem file can be generated faster, shorten the auto-planning time.
- Noticing that many top-performed planners in ICP does not support metric feature, efficient algorithms that removes the metric requirements in the resource planning part of a migration planning problem needs to be developed.
- Improve planner's computational speed or develop algorithms so it can generate migration plan for thousands assets and more complicated migration scenarios.

References

Cardoen, B., Demeulemeester, E., and Beliën, J. 2010. Operating Room Planning and Scheduling: A Literature Review. *European Journal of Operational Research,* 201: 921–932.

Uzsoy, R., Lee, C., and Martin-Vega, L.A. 1992. A Review of Production Planning and Scheduling Models in the Semiconductor Industry Part 1: System Characteristics, Performance Evaluation and Production Planning. *IIE Transactions,* 24(4): 47–60.

Bazargan, M. eds. 2010. *Airline Operations and Scheduling* London, UK.: Ashgate Publishing Co.

Armbrust, M., Fox, A., Griffith R., Joseph A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., and Zaharia, M. 2010. A View of Cloud Computing. *Communications of the ACM,* 53(4): 50–58.

Khajeh-Hosseini, A., Greenwood, D., and Sommervile, I. 2010. Cloud Migration : A Case Study of Migrating and Enterprise IT System to IaaS. In *Proceedings of 2016 IEEE/IFIP Network Operations and Management Symposium,* 96–103. Istanbul, Turkey.

Vukovic, M., and Hwang J. 2016. Cloud Migration using Automated Planning. In *Proceedings of 2010 IEEE 3rd International Conference on Cloud Computing,* 450–457. Miami,FL.

Transition Manager, https://www.transitionaldata.com/transitionmanager−data−center−migration−tools/

Velostrata, https://cloud.google.com/velostrata/

The Apache Groovy Programming Language, http://groovy−lang.org

Creating and modifying runbooks, https://cloud.google.com/velostrata/docs/how−to/organizing−migrations/creating−and−modifying−runbooks

Ghallab, M., Nau D., and Traverso, P. 2004. *Automated Planning Theory and Practice* San Francisco, CA.: Elsevier.

Jackson, M., Rofrano, J., Hwang, J., and Vukovic M. 2018. BluePlan: A Service for Automated Plan Construction using AI. In *International Conference on Service-Oriented Computing,* Demo Session. Hangzhou, China.

Evaluating the Cost of Employing LPs and STPs in Planning: Lessons Learned From Large Real-Life Domains

Elad Denenberg and **Amanda Coles** and **Derek Long**
Department of Informatics, King's College London, UK.
email: {elad.denenberg,amanda.coles,derek.long}@kcl.ac.uk

Abstract

When solving real-life problems we often encounter issues that are not captured by academic benchmark domains. In this paper we consider an application problem, representative of a class of real-world problems that have interesting properties: long solution plans with many temporal/numeric constraints. We identify a number of limitations of a popular family of planners in solving these problems. This family of planners perform Forward Search and call a Linear Programming (LP) solver multiple times at every state to check for consistency, and to set bounds on the numeric variables in order to determine action applicability. These checks during search allow the pruning of branches; however, they do carry computational cost. In this paper we investigate and analyse this trade-off, with particular reference to our class of application problems, and show that adapting the planners to call the LP solver less often, and using a cheaper consistency check at each state, can improve performance.

1 Introduction

Automated Planning is concerned with using a planner to formulate a sequence of actions that transforms a given initial state into a desired goal state. One strength of planning is domain-independence: a single general planner can plan in a wide range of different application domains. For example, space (Chien et al. 2000), battery usage (Fox, Long, and Magazzeni 2011) and software penetration testing (Obes, Sarraute, and Richarte 2013). In order to facilitate their application in realistic problems planners need to reason with expressive models of the world. Such models can be temporal: finding a plan with timestamped actions, taking into account action durations and concurrency; as well as numeric: considering numeric variables that change discretely, or in hybrid problems also continuously, over time (Fox and Long 2003).

When planning in expressive domains a planner needs a mechanism to schedule the plan, i.e. assign timestamps to actions to meet the temporal and numeric constraints. One option is the decision epoch mechanism of SAPA (Do and Kambhampati 2001) and Temporal Fast Downward (Eyerich, Mattmüller, and Röger 2009); alternatively, one can use simple temporal networks (STNs) (Dechter, Meiri, and Pearl 1991) e.g. as in the planner Crikey 3 (Coles et al. 2009). Approaches to hybrid planning employ more complex mechanisms include compiling the problem into SAT Modulo Theories (Cashmore et al. 2016), interval relaxation (Scala et al. 2016), time discretization (Piotrowski et al. 2016) and convex optimization (Fernández-González, Karpas, and Williams 2017).

In this work we focus on a family of planners that make use of LP solvers to schedule the plan. This family includes COLIN (Coles et al. 2012), POPF (Coles et al. 2010) and OPTIC (Benton, Coles, and Coles 2012). These planners perform forward state-space search starting from the initial state. At each state in the search a LP solver is used once to determine whether there exists a consistent schedule for the plan (adhereing to temporal and numeric constraints). If a no consistent schedule exists for the plan to the current state, the search branch can be pruned. If the state is consistent, the LP is then used multiple more times to bound the numeric variables and thus prune the space of applicable actions in this state, narrowing the search space ahead.

In this work, we analyse the performance of these planners on a particular interesting class of real-world application problems, illustrated in a domain provided by our industrial partner. This domain highlights issues they encountered in using this family of planners in their deployed applications. Specifically, the problems require long solution plans, with large numbers of numeric variables and temporal constraints. An important observation made in this class of application problems is that whilst the traditional planning benchmarks, on which the planners were initially evaluated, lead typically to small LPs being generated at each state (Coles et al. 2012); LP solving in this class of application domains is much more expensive as the long plans, hence large numbers of variables, lead to larger LPs, which hinders planner performance.

In this paper we consider the trade-offs involved in solving LPs to check for consistency and bound variables at every state: theoretically, since we only require the final plan to be valid, we could simply complete the consistency check on states the planner believes to be goal states. The conventional wisdom has been that checking for inconsistent plans at each state allows pruning of the search space thus expanding fewer nodes to reach the goal; however, this does come at the cost of a higher per-node expansion overhead, especially if the LPs to be solved are large and complex. Our contribution here is propose and evaluate a range of strategies for using schedulers to prune, ranging from the traditional use of a LP for consistency and bounds checking at every state; though to using just an STN to check only tem-

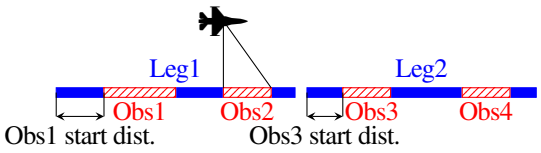


Figure 1: The flying observer

poral consistency at each state to checking only for consistency in goal states. We thoroughly analyse planner performance with respect to different configurations on our representative application problems and show that we can obtain better performance in these domains with an approach that performs cheaper STN consistency checking on a per-state basis, whilst stil guaranteeing plan consistency using a LP in the goal state.

2 Problem Definition

A temporal planning problem with discrete and linear continuous numeric effects is a tuple:

$$\langle I, G, \mathbf{A}, \mathbf{P}, \mathbf{V} \rangle \quad (1)$$

where $\mathbf{P}$ is a set of propositions and $\mathbf{V}$ a set of numeric variables. I is a set of value assignments to these propositions and numeric values, representing the initial state of the problem, G is the goal: a conjunction of propositions in P and linear numeric conditions over the variables in V, of the form $w^1v^1+w^2v^2+...+w^iv^i\{<,\leq,=,\geq,>\}c$ ($w^1...w^i$ and c are constants $\in \mathbb{R}$). $\mathbf{A}$ is a set of actions defined by the tuple:

$$\langle d, pre_{\vdash}, eff_{\vdash}, pre_{\leftrightarrow}, eff_{\leftrightarrow}, pre_{\dashv}, eff_{\dashv} \rangle \quad (2)$$

where d is the duration of the action constrained by a conjunction of numeric conditions. $pre_{\vdash}$ and $pre_{\dashv}$ are conjunctions of preconditions (facts and numeric conditions) that must be true at the start and end of the action, $pre_{\leftrightarrow}$ are invariant conditions (preconditions that must hold throughout the action's duration), $eff_{\vdash}$ and $eff_{\dashv}$ are instantaneous effects that occur at the start and end of the action. Such effects may add or delete a proposition $p \in \mathbf{P}$ (eff_{+}, eff_{-}) or update a numeric variable $v^i \in V$ according to a linear instantaneous change (eff_{num}). In this work all effects in eff_{num} are assumed to be linear and of the form: $v^k\{+=,=,-=\}w^1v^1+w^2v^2+...+w^iv^i+w^j*dur_A+c$ where dur_A is a special variable representing the duration of the action A of which this is an effect ($w^1...w^j$ and $c \in \mathbb{R}$). $eff_{\leftrightarrow}$ is a conjunction of continuous effects operating throughout the duration of the action. In this work each continuous effect is linear, i.e. of the form $\frac{dv}{dt}\{+=,=,-=\}c$ where $c \in \mathbb{R}$ is a constant.

3 Example Representative Application Domain

This example was supplied by our industrial partner. It encapsulates the structure of problems that arise when using an LP planner in their target application domains.

In this domain, named flying observer, the planner is required to plan a Unmanned Aerial Vehicle (UAV) observation mission. The UAV is required to fly *leg*s over a desired streach of land containing objects to be observed. Each leg is of different length. Each observation has a different duration and requires a different type of equipment. To mark the area within the leg in which the observation must take place a target-start distance is defined. The observation can take place only when the UAV has flown more than the target-start distance of that leg (flown$_l$ $?leg \geq$ target-start$_o$). A continous numeric effect of the fly$_l$ action updates the distance flown so far in a leg: $\frac{\mathrm{dflown}_l}{\mathrm{dt}} = 1$.

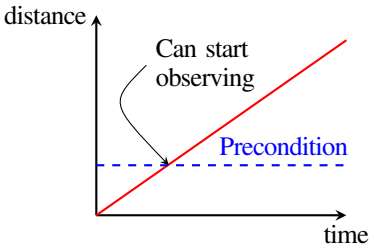
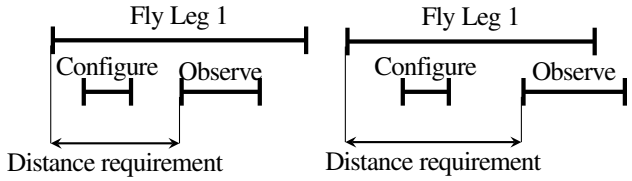


Figure 2: Distance Requirement



(a) Possible Plan (b) Impossible Plan

Figure 3: Durative Meaning of Distance Requirement

Fig 1 illustrates an instance of this domain: in this instance two legs are defined (marked in solid blue lines), in each leg two observations are required (marked in red, pattern filled lines). All observations have a target-start distance defined, but for clarity only the starting distance of the first and third observations are shown.

In order to perform an observation a defined piece of equipment needs to be calibrated and configured for a specific observation. Once the observation is done the equipment needs to be released to become available for future observations. The domain comprises the following actions:

- take-off$_l$: dur = 5; $pre_{\vdash}$ ={on-ground,first-leg$_l$};
  $eff_{\vdash}$ ={¬on-ground, flown$_l$ = 0}; $eff_{\dashv}$ ={flying$_l$};
- set-course$_{l1,l2}$: dur = 1; $pre_{\vdash}$ ={done$_{l1}$,next$_{l1,l2}$};
  $eff_{\vdash}$ ={¬done$_{l1}$}; $eff_{\dashv}$ ={flying$_{l2}$, flown$_{l2}$ = 0};
- fly$_l$: dur=$^{\text{distance}_l}/_{\text{speed}_l}$;
  $pre_{\vdash}$ ={flying$_l$}; $pre_{\leftrightarrow}$ ={flown$_l \leq$ distance$_l$};
  $eff_{\dashv}$ ={done$_l$, ¬flying$_l$}; $eff_{\leftrightarrow}$ ={$^{\text{dflown}_l}/_{\text{dt}}$ += 1};
- configure$_{o,e}$: dur = 1; $pre_{\vdash}$ ={available$_e$, optionfor$_{o,e}$};
  $eff_{\vdash}$ ={¬available$_e$}; $eff_{\dashv}$ ={configuredfor$_o$, pending$_{o,e}$};
- observe$_{l,o}$: dur = time-for$_o$; $pre_{\vdash}$ ={configuredfor$_o$, contains$_{l,o}$, awaiting$_o$, target-start$_o \leq$flown$_l$};
  $pre_{\leftrightarrow}$ ={flying$_l$}; $eff_{\vdash}$ ={¬awaiting$_o$}; $eff_{\dashv}$={observed$_o$}
- release$_{o,e}$: dur = 1; $pre_{\vdash}$ ={pending$_{o,e}$};
  $eff_{\vdash}$ ={¬configuredfor$_o$, ¬pending$_{o,e}$}; $eff_{\dashv}$ ={available$_e$};

The target distance precondition and temporal constraints of this problem force the configure/observe actions to fit within the fly action. The meaning of the precondition is illustrated in Fig 2: The red line is a depiction of the distance change as the UAV flies over the leg. The dashed blue line is the precondition signifying the distance required for the start of the observation.

Algorithm 1: Overview of OPTIC's Search

Data: Planning Problem $\langle I,G,A,P,V\rangle$
Result: A solution plan
1 $Q \leftarrow [I]$;
2 **while** Q *is not empty* **do**
3 $S \leftarrow$pop the next state from Q;
4 $app \leftarrow a \in A_{snap} \cdot S \models pre(A)$;
5 **foreach** $a_i \in app$ **do**
6 $S' =$ apply(a,S);
7 **if** $\neg$ *isConsistent(S')* **then continue**;
8 ;
9 **if** $S' \models G$ **then return** *plan to* S';
10 ;
11 $UpdateNumericVariableRanges(S')$;
12 $h(S') = ComputeHeuristicValue(S')$;
13 **if** *h(S')* $\neq \infty$ **then** $Q.enqueue(S')$;
14 ;
15 **return** *problem unsolvable;*

When the distance reaches the value required in the precondition the observation can start. The meaning of the numeric constraint as it is manifested in the temporal state can be seen in Fig 3a.

Notice that by defining contains$_{l,o}$ for multiple legs the planner can be given a choice of multiple legs in which it can decide to perform observation o. Further, notice that the legs must be flown in the order defined by the predicates next$_{l1,l2}$, therefore, a leg may not be skipped even if it does not contain an observation.

4 Linear Programming (LP) based planners

The mechanism described here allows planners to reason with continuous numeric change. It was first described in COLIN (Coles et al. 2012), then used in POPF and OPTIC. An overview of this search is given in Algorithm 1.

4.1 Search In OPTIC

OPTIC performs forward search from the initial state and branching over applicable actions, exploring partially-ordered but un-time-stamped sequences of snap-actions. Snap-actions are instantaneous actions marking the start ($A_\vdash$) and end ($A_\dashv$) of a durative action A: $A_\vdash$ has preconditions $pre_\vdash$ A A and effects $eff_\vdash$ A; $A_\dashv$ has preconditions $pre_\dashv$ A and effects $eff_\dashv$ A. For brevity of notation we define the set A_{snap} to contain all snap actions corresponding to the start and end of actions in A ($A_\vdash$, $A_\dashv$ such that $a \in A$) and all instantaneous actions in A.

Each state S in search comprises the set of propositions ($S.p \subseteq P$) that are true in S and optimistic upper ($S.max(v)$) and lower ($S.min(v)$) bounds on the value each variable in V can hold in S. In the initial state all variables have $max(v) = min(v) =$ the value of v specified in the initial state; $max(v)$ and $min(v)$ will only differ from each other in subsequent states if/when a variable has been subject to continuous change in the plan so far, as the value of v will then change as time elapses in S. Search proceeds by popping the first state from the openlist: in our work, we use WA* (W=5) so sort the openlist by $h(S)+5.g(s)$, using the temporal-numeric RPG heuristic of COLIN (Coles et al. 2012).

At line 4 the planner identifies the actions applicable in S, i.e. those whose preconditions are satisfied in S (and do not lead to a state in which the invariants of the currently executing actions are violated). A propositional precondition p is satisfied if it is true in the state, i.e. $p \in S.p$. Numeric preconditions, of the form $w^1v^1 + w^2v^2 + ... + w^iv^i\{\geq,>,=,<,\leq\}c$, are deemed satisfied if the values of $max(v)$ and $min(v)$ optimistically satisfy them: that is, for example, we consider $v^1 - v^2 >= 5$ to be satisfied if $ub(v^1) - lb(v^2) \geq 5$. Next (line 6) we use the applicable actions to generate all successors S' of S by adding/deleting all propositions in eff_a^+ and eff_a^- respectively, and applying all discrete numeric effects to both $max(v)$ and $min(v)$ for all $v \in V$ affected by eff_a^{num}. At this point OPTIC also adds the necessary *ordering* constraints to the plan: the action that has just been applied is ordered after the last actions to add each of its preconditions, after actions whose preconditions it deletes, and after actions with numeric effects on variables it updates or refers to in preconditions/effects: all such constraints are of the form $t_j - t_i \geq \epsilon$, where t_j/t_i are the times at which the new and existing action must occur respectively and ϵ is a small constant enforcing separation.

4.2 Checking Temporal/Numeric Consistency

Whilst the plan generated by the above search is guaranteed to be *propositionally consistent* (all propositional preconditions are satisfied when an action is applied) it might not be *temporally or numerically consistent*. Temporal/numeric consistency of plans must be explicitly checked, this is done at Line 7, using either an STN or LP as appropriate. To illustrate why this in the case, consier the partial plan: take-off,fly$^\vdash_{l0}$, configure$^\vdash_{o1,e2}$, configure$^\dashv_{o1,e2}$,fly$^\dashv_{l0}$ in our application domain. This plan is propositionally sound, but if the duration of configure exceeded the duration of fly then this would not be a *temporally consistent* plan.

First, consider the case where there are no continuous or duration dependednt numeric effects in the plan (like the one given above). In this case we can update the values of all numeric variables in each state according to discrete numeric effects, and know the exact values of v_i in each state, allowing us to enforce numeric preconditions correctly during search (max(v) = min(v) = the known value of v in S). The only remaining constraints we have are temporal constraints: ordering constraints of the form $t_j - t_i \geq \epsilon$ and duration constraints of the form $t_{a\dashv} - t_{a\vdash}\{\leq,=\geq\}c$, where c is a constant (without loss of generality, c can be computed from the known values of variables $v^i \in v$ in the state in which $a_\vdash$ was applied). It is well known that such a set of constraints constitutes a simple temporal network (STN) (Dechter, Meiri, and Pearl 1991): a solution to this represents a valid schedule for the snap-actions in the plan. This can be solved (or proven unsolvable) in polynomial time using an all-pairs shortest path algorithm.

Consider now the more complex case where we have a partial plan that has a precondition on a variable that has undergone continuous numeric change, e.g.: take-off,fly$^\vdash_{l0}$, configure$^\vdash_{o1,e2}$, configure$^\dashv_{o1,e2}$, observe$^\vdash_{l0,o1}$, observe$^\dashv_{l0,o1}$,fly$^\dashv_{l0}$. We can as above, equally check that the plan is temporally consistent (the duration of configure and observe actions are less than the fly action) using a simple temporal network; if the plan were temporally inconsistent we could prune it using only an STN. However, we cannot be certain using an STN that this plan is valid as the

STN does not take into account the precondition of observe$_{l0,o1}$: target-start$_{o1} \leq$ flown$_l$. It might be that there is insufficient time within the leg $l0$ action to wait until this constraint is met (e.g. if leg $l0$ has duration 15 and observe$_{l0,o1}$ duration 10 and target-start distance 11) then the plan would not be temporally and *numerically* sound (this is illustrated in Fig 2–3). In this case we need to use an LP to encode both the temporal and numeric constraints of the problem. Section 5 details how this LP is built in such a way that a solution is a valid assignment of time stamps to the snap actions in the partial plan that satisfies the temporal and numeric constraints of the problem.

At line 7 OPTIC intelligently selects the scheduler used based on the constraints in the plan and will use the cheaper STN by default, using the LP only when constraints that necessitate it are present. If the LP, or STN, determines that there is no consistent schedule for the plan reaching a state, the plan is temporally or numerically invalid and so the state is pruned. If the LP determines there is a valid schedule of the actions that satisfies the temporal/numeric constraints then S' is a valid successor of S. If S' satisfies the goal, we have as solution plan. Otherwise, the LP is used again to find the range on each state variable (line 11) prior to heuristic evaluation, using the standard temporal/numeric relaxed planning graph heuristic of Colin (Coles et al. 2012). It is then inserted into the openlist, providing $h(S') \neq \infty$, i.e. the heuristic does not indicate S' is a dead-end.

To understand this use of the LP consider the partial plan: take-off,fly$^{\vdash}_{l0}$, fly$l0^{\vdash}$, configure$^{\vdash}_{o1,e1}$, configure$^{\dashv}_{o1,e1}$. As soon as the action fly$^{\vdash}_{l0}$ is applied we have a continuous effect increasing the value of the numeric variable flown$_{l0}$, so we can now no longer say flown is in the range [0,0] (as in the initial state) but it could be anywhere in the range $[0,\infty]$ as an effect increasing it has started but not yet ended. In the general case we can exploit the same LP we used for consistency checking to tighten the bounds on numeric variables in each state: for each $v \in V$ that has been subject to any continuous or duration-dependent change in the plan to reach S' we set the LP objective function to maximise (and then minimise) v to find out the maximum possible value of v admitted by the plan. Again, we explain how the LP is used to do this in Section 5.

We make two observations about this use of the LP to tighten variable bounds. First it is effectively optional: even if we assumed the bounds on v were $[0,\infty]$ then when S' is later expanded, and isConsistent() is used on its successors, this would prune any reached by an action whose preconditions were not satisfiable in S'. Though, the use of the LP to generate tighter bounds on the value of v helps prune the list of applicable actions: if an action can be pruned because its numeric preconditions are unsatisfied according to the tighter bounds on v, this saves building another LP and using isConsistent for the corresponding successor: this is a key trade-off that we investigate in this paper. The second observation is that these bounds are optimistic: that is, we ask the LP to maximise v and separately to minimise w however, it might be the case that the maximum value of v and the minimum value of w are not achievable by the same schedule; so the precondition $v-w>=5$ might still not be satisfiable even if the bounds state that it is; thus we always need the is consistent check upon expanding S' to confirm its preconditions are satisfied; although this would be needed anyway, to ensure any other temporal/numeric constraints are satisfied when applying the action.

Step	Action	variables	constraints	comment
0	TakeOff	t_0	≥ 0	
1	Fly$^{\vdash}_{l0}$	t_1	$-t_0 \geq \epsilon$	Step1 afer Step0
		$flown\_l0_1$	$=0$	Initial Assignlemt
		$flown\_l0'_1$	$flown\_l0_1$	Value after action
			$\leq distance\_l0$	Invariant
2	Configure$^{\vdash}_{o1,e1}$	t_2	$-t_1 \geq \epsilon$	Step2 after Step1
		$flown\_l0_2$	$=flown\_l0'_1+1*(t_2-t_1)$	Value before action
			$\leq distance\_l0$	Invariant
		$flown\_l0'_2$	$=flown\_l0_2$	Value before Action
			$\leq distance\_l0$	Invariant
3	Configure$^{\dashv}_{o1,e1}$	t_3	$-t_2 \geq \epsilon$	Step2 after Step1
		$flown\_l0_3$	$=flown\_l0'_2+1*(t_3-t_2)$	Value before action
			$\leq distance\_l0$	Invariant
		$flown\_l0'_3$	$=flown\_l0_2$	Value before Action
			$\leq distance\_l0$	Invariant
4	Observe$^{\vdash}_{o1,l0}$	t_4	$-t_3 \geq \epsilon$	Step3 after Step2
		$flown\_l0_4$	$=flown\_l0'_3+1*(t_4-t_3)$	Value before action
			$\geq target\text{-}start\_o1$	Start precondition
			$\leq l0\_dist$	Invariant
		$flown\_l0'_4$	$=flown\_l0_4$	Value after action
			$\geq Target\_dist\_o1$	Strat precondition
			$\leq l0\_length$	Invariant
5	Observe$^{\dashv}_{o1,l0}$	t_5	$-t_4 \geq \epsilon$	Step4 after Step3
			$-t_4 \leq time-for$	Action duration
		$flown\_l0_5$	$=flown\_l0'_4+1*(t_5-t_4)$	Value before action
			$\leq distance\_l0$	Invariant
		$flown\_l0'_5$	$=flown\_l0_5$	Value after action
			$\leq l0\_length$	Invariant
6	now	t_{now}	$-t_5 \geq \epsilon, -t_4 \geq \epsilon, -t_3 \geq \epsilon,$ $-t_2 \geq \epsilon, -t_1 \geq \epsilon, -t_0 \geq \epsilon$	After All Steps
		$flown\_l0_{now}$	$=flown\_l0'_5+1*(t_{now}-t_5)$	Value Now

Table 1: LP Equations of a Partial Plan

5 Building LPs in OPTIC

In this section we detail how the LP to check plan consistency is formulated at each state. We refer throughout to Table 1 which shows an example LP for the partial-plan: take-off, fly$^{\vdash}_{l0}$, configure$^{\vdash}_{o1,e1}$, configure$^{\dashv}_{o1,e1}$, observe$^{\vdash}_{o1,l0}$, observe$^{\dashv}_{o1,l0}$.

To represent the planning problem as an LP the following LP variables are defined for each step i of the partial plan: t_i defining the time at which the step is to be taken, v_i the value of variable $v \in \mathbf{V}$ just before the application of the action and $v'_i \in \mathbf{V}$ the value of variable v just after the application of the action.

Temporal Constraints: Ordering constraints can be enforced exactly as written, when step j must occur after step i we write:

$$t_j - t_i \geq \epsilon \tag{3}$$

Duration constraints can also be formulated directly:

$$t_j - t_i \{\geq, \leq, =\} w_i^1 v_i^1 + w_i^2 v_i^2 + ...c \tag{4}$$

For example, in Table 1 ordering constraints enforce that observe$^{\vdash}_{o1,l0}$ occurs at least ϵ after configure$^{\dashv}_{o1,e1}$ as the latter achieves a precondition of the former; and duration constraints enforce that observe$^{\dashv}_{o1,l0}$ occurs exactly the defined duration after observe$^{\vdash}_{o1,l0}$ ($t_{Obs1_{\dashv}} - t_{Obs1_{\vdash}} = dur_{Obs1}$). Recall that in the absence of continuous numeric effects and duration-dependent effects, the LP need only contain these temporal constraints, and the right hand sides of all duration constraints are known constants, therefore an STN solver suffices.

Numeric Constraints: To manage linear continuous change an additional variable δv_i stores the sum of change acting on

variable v after step i. If A is a durative action with a continuous linear effect, and c_A the constant defining said linear change (i.e. $\frac{dv_i}{\mathrm{dt}} += c_A$). Then δv_i is calculated thus:

$$\delta v_i = \begin{cases} \delta v_{i-1} + c_A & \text{if} A_i = A_{\vdash} \\ \delta v_{i-1} - c_A & \text{if} A_i = A_{\dashv} \end{cases} \quad (5)$$

This, if A starts at $step_i$, c_A will be added to δv_i. If A ends at $step_j$, c_A will be removed from δv_j. Thus δv_i is the sum of all effects currently active on $v \in \mathbf{V}$ after step$_i$. We use δv_i to compute the value of variables undergoing continuous numeric change:

$$v_i = v'_{i-1} + \delta v_{i-1}.(t_i - t_{i-1})$$

This can be seen in Table 1: the only continuous effect is acting on $\mathit{flown\_l0}_i$ from step 2 onward. Thus $\delta \mathit{flown\_l0}_i = 1$ (for $i \geq 2$) and the value of $\mathit{flown\_l0}_i$ is calculated according to this at each step (e.g. $\mathit{flown\_l0}_2 = \mathit{flown\_l0}'_1 + 1.(t_2 - t_1)$).

Numeric preconditions $pre_{\vdash}$ and $pre_{\dashv}$ are formulated over the respective variables v_i. We formulate the invariant conditions $pre_{\leftrightarrow}$ of actions at the start and end steps of the action and at every step between them. This is sound because all effects are linear so no turning points can be present between the steps.

In the example LP for the flying observer (Table 1) the distance-flown (flown$_{l1}$) precondition on the observe$^{\vdash}_{o1,l1}$ action is encoded at step3 over the variable ($\mathit{flown\_l0}_3$); and the invariant ($\mathit{flown}_{leg_j} \leq \mathit{distance}_{l0}$) of fly_{l0} can be seen enforced immediately after step$_1$ (i.e. on $\mathit{flown\_l0}'_1$) and before and after all subsequent steps that refer to $\mathit{flown\_l0}$ (over $\mathit{flown\_l0}_i$ and $\mathit{flown\_l0}'_i$).

A solution to this LP gives us an assignment to each t_i representing a valid time stamp for each $step_i$ in the partial plan; if the LP solver reports that no solution exists then the plan is inconsistent and we can prune the resulting state. If the state is deemed consistent the formulated LP problem is used again (Algorithm 1, Line 11) to determine bounds on the numeric variables. To do this we create new step now, with associated timestamp variable t_{now} ordered after all existing steps; then for each variable $v \in \mathbf{V}$ we create v_{now}, and calculate its value at t_{now} in the usual way. The LP is solved with an objective minimise (then again to maximise) v_{now} yielding the minimal (maximal) possible value v may hold in the current state. Table 1 illustrates this for the variable $\mathit{flown\_l0}$. t_{now} is constrained to come after all other plan steps, and the value of $\mathit{flown\_l0}_{now}$ is computed as $\mathit{flown\_l0}'_5 + 1.(t_{now} - t_5)$; using the objective miminise (maximise) $\mathit{flown\_l0}_{now}$ will tell us the maximum and minimum feasible values of $\mathit{flown\_l0}$ we can expect to rely on for the precondition of any action to be applied in this state.

Prior work observed that the formulation of the LP was expensive and solving was cheap (Coles et al. 2012). Therefore since the LP is not reformulated, but simply resolved for different v_{now}s, computing bounds was relatively inexpensive. However, this did not match our observations when examining our complex real-life problems with large LPs[1].

6 LP vs STN Scheduling in OPTIC

It is well known that the scalability of planners is affected by both the number of applicable actions per state (branching factor) and the length of the required solution plan (depth to which the search tree must be explored). The latter of these factors is magnified in planners using LP schedulers because the size of the LP being solved increases with the length of the plan being scheduled. In this section we explore the scalability of LP based planners in our complex real-life observer domain, which involves continuous numeric change and requires long solution plans compared to conventional benchmark domains.

Instance	Observations	Legs	Observations Required in Goal
1	10	28	4
2	15	38	6
3	20	48	8
4	25	58	10
5	30	68	12

Table 2: Single Observation Per-Leg Instances

6.1 Standard OPTIC Performance

To explore this behaviour we ran the flying observer domain on 5 instances increasing in difficulty. In this domain, there is never more than one observation that can be chosen to happen in each leg (i.e. if contains$_{l1,o1}$ is defined then there does not exist any other observation o$_i$ such that contains$_{l1,oi}$ is defined). Further, there is no choice over which leg each observation can occur in (i.e. if contains$_{l1,o1}$ is defined then there does not exist any other leg l_i such that contains$_{li,o1}$ is defined). Each observation requires one piece of equipment out of the three available. The number of observations defined, the number of legs, and the number of observations required in the goal differs between the instances as specified in Table 2.

Note that to emulate a real life scenario where the instance of the problem may contain many optional actions, not all of which are required for the goal, the number of observations and legs here is much greater than is required for the goal. For instance, leg number 25 and higher is not required, yet is a possibility that the planner might consider. The same goes for observation number 20 - it is defined, but not required for the goal. Solving the hardest instance of this domain was slow and took OPTIC about 260 seconds on an Intel i7 2.80GHz.

A second variant of the domain, again representing an additional challenge encountered in our application, adds a global variable counting the total distance flown, and a precondition to the "configure" action requiring that this is not greater than 1000 units. The changed actions would therefore be:

1. fly$_l$: dur=$^{\text{distance}_l}/_{\text{speed}_l}$; $pre_{\vdash}$ ={flying$_l$}; $pre_{\leftrightarrow}$ ={flown$_l$ $\leq$ distance$_l$}; $\mathit{eff}_{\dashv}$ ={done$_l$, $\neg$flying$_l$}; $\mathit{eff}_{\leftrightarrow}$ ={$^{\text{dflown}_l}/_{\text{dt}}$ += 1, $^{\text{dtotalFlown}}/_{\text{dt}}$ += 1};
2. configure$_{o,e}$: dur = 1; $pre_{\vdash}$ ={available$_e$, optionfor$_{o,e}$, totalFlown$\leq$1000}; $\mathit{eff}_{\vdash}$ ={$\neg$available$_e$}; $\mathit{eff}_{\dashv}$ ={configuredfor$_o$, pending$_{o,e}$};

Before adding this variable the "configure" action was entirely propositional, and therefore consistency of plans containing it could be confirmed using an STN. Now it has a precondition that inspects a variable affected by continuous numeric change so, as discussed in Section 4, any plan containing this action requires a LP to confirm consistency.

[1] OPTIC's LP building code is also more optimized than COLIN's

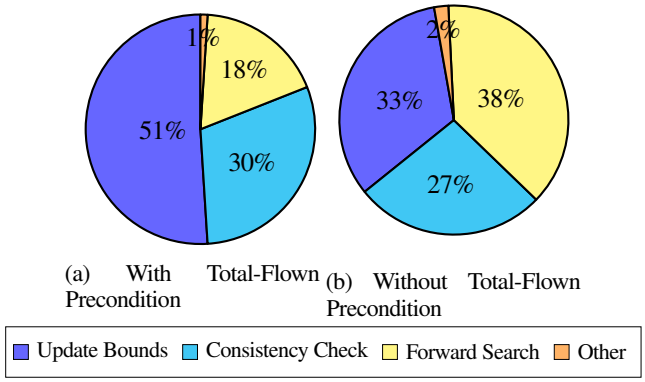


Figure 4: Single Observation Per Leg Profiling (Instance 5)

		Nodes			Nodes STN		Nodes LP	
	#	Generated	Expanded	Evaluated	Checked	Pruned	Checked	Pruned
Normal	1	413	41	407	309	0	309	2
	2	4976	395	4750	4855	0	4855	222
	3	28465	1570	28032	28315	0	28315	429
	4	92358	6683	91115	92195	0	92195	1239
	5	135043	10479	133251	134879	0	134879	1788
LP for Goals	1	413	41	409	409	0		
	2	4976	395	4972	4972	0		
	3	28465	1570	28461	28461	0		
	4	92358	6683	92354	92354	0		
	5	135043	10479	135039	135039	0		

Table 3: Nodes in Search, Single Observation Per Leg

Solving the hardest instance without the total-flown variable (the initially described domain) took about 260 seconds 60% of which were spent solving the LP. However, when solving the hardest instance with the total-flown variable and precondition, planning took about 560 seconds 81% of which were spent in the LP. Fig 4 shows the profiling results of the 5th instance in the domain with the total-flown precondition and without. STN solving takes negligible time, and is included in the 'Forward Search' measurement; LP takes the most time, particularly with 'total-flown'. Inspecting further, the more nodes generated during search (the more LPs solved) the slower it is. For instance, in general it was seen that running the domain lacking the total-flown requirement was faster, however, in instance 3 of the domain, more nodes were visited during search, and the runtime on that instance specifically was slower.

Also notable is that in our domain, more time is spent computing variable bounds than consistency checking. This suggests these LP calls are expensive, so avoiding them may be beneficial.

6.2 Proposals to Improve Performance

Since the LP solving is computationally expensive, we try to reduce the number of times the LP solver is called during search. Four options for reducing the amount of calls for the solver are presented here:

1. Call updateVariableRanges (Algorithm 1 Line 11) only for the variables which have an active continuous effect acting on them in the partial plan (i.e. an action with a continuous effect on v has started, but not yet finished);
2. Solve the LP only to check for consistency (Line 7), and don't call updateVariableRanges for any variables (i.e. skip Line 11);
3. Solve an STN to check only *temporal consistency* (at Line 7) even if an LP would normally be used. Solve the LP to check for consistency only in possible goal states (i.e. at Line 9);
4. Solve neither the LP nor an STN at each state (remove Line 7), and solve the LP only in possible goal states.

The first option, updating only the active variables, reduces the number of LPs solved in each state by only increasing bounds due to effects, but not always tightening them based on the preconditions. A continuous change can increase the bounds on a variable, whereas preconditions may only tighten the bounds. In the default configuration OPTIC updates bounds on all variables that have ever been subject to continuous numeric change; here we suggest optimizing only the variables that are acted upon by currently executing actions. This is a compromise: by checking the effect we guarantee the most optimistic bounds, while reducing the number of LPs solved.

The second option is to solve the LP only for consistency checks (i.e. do not use the LP to update variable bounds). This means fewer actions will be marked as inapplicable, and the pruning of branches is done based on consistency only. Updating bounds is done to narrow the search space; however, if the cost of this is more expensive than the search effort saved this is not worthwhile.

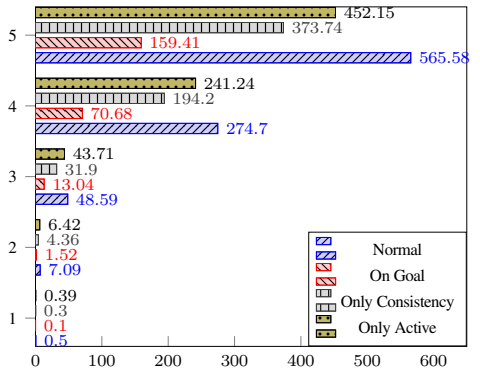
The third option is to solve the LP only on the states which the forward search finds as possible goal states. This means that fewer actions are marked as inapplicable during the search and the states are only tested for temporal consistency, not for numeric consistency, until branching reaches a goal state. Search finds an ordering of actions that transform the initial into a goal state, here the LP solver is called to schedule the ordered actions. If a schedule is found then this indeed is a goal state. If a schedule cannot be found than this is an invalid branch and the planner needs to backtrack.

The fourth option is ploughing through forwards search disregarding any temporal or numerical constraints, then, when a possible goal state is reached, solve a LP to check whether it can be scheduled. When this method finds a solution, it is not much faster than method 3. Sadly though, it almost never finds one. In the domains we checked this method was only able to solve the first and second instances of the single observation-per-leg domain, and did so quite quickly. However, it timed out on all other instances and domains we checked. Because hardly any pruning is done it is quite easy for search to get stuck in a fruitless subspace when this method is employed. It is therefore not recommended, and is not shown in the results.

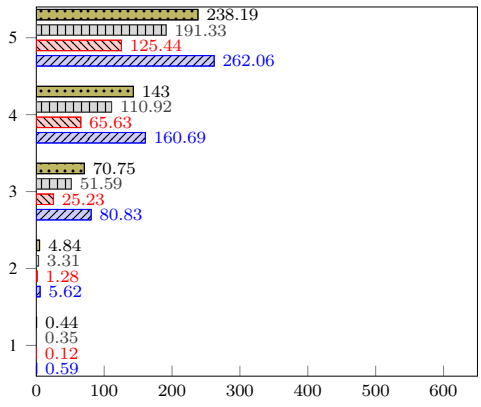
6.3 Comparison of Proposals

Fig 5 presents the results of the first three options on the 5 instances. Fig 5a lists the results on the domain with the total-flown precondition on the "configure" action, and Fig 5b without.

The slowest method, named "Normal" (blue, northeast to southwest pattern) is running OPTIC with the default configuration: solving the LP for consistency, then several more times to find the bounds on the numeric variables, and is always the most computationaly expensive. Next (olive, dotted pattern) is the

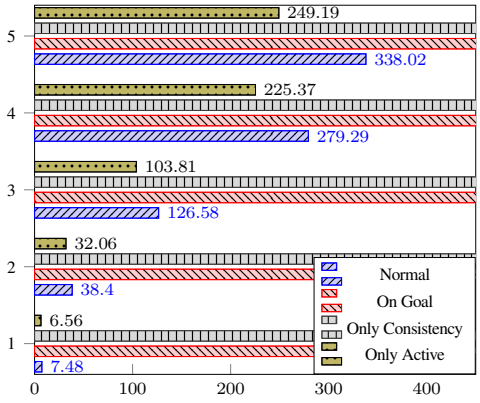


(a) With Total Flown Constraint

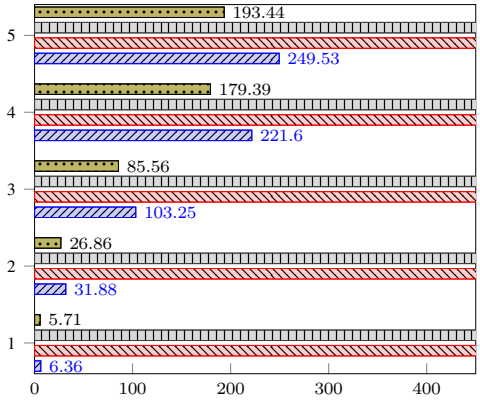


(b) Without Total Flown Constraint

Figure 5: Single Observation Per Leg Runtime (seconds)



(a) With Total Flown Constraint



(b) Without Total Flown Constraint

Figure 6: Multiple Observation Per Leg Runtime (seconds)

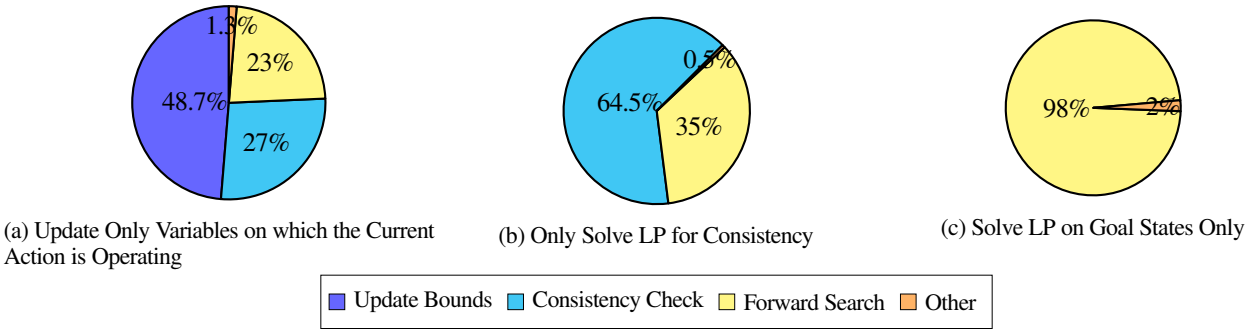
option that solves the LP for consistency checking, but updates only the bounds of the variables currently subject to continuous numeric change. Faster still is the option that solves the LP only for consistency checking (gray, horizontal pattern) but does not update bounds. Finally, the fastest option on these instances is the third option: solving the LP only at at the goal state while using STN in non-goal states for the consistency check (marked red, with northwest to southeast pattern). As stated before the 4th option performance was too poor to include here.

Table 3 shows the number of nodes generated, expanded and evaluated during the search, as well as the number of times STN and LP solvers were called for consistency checks and the number of times each found a branch to be inconsistent. The number of nodes examined in the default configuration was identical to those in the update of the active variables only and when no update took place, and therefore are not shown here. The fact the number of nodes were similar suggests that setting and updating the bounds on the variables did not help flagging actions as invalid, and therefore, in these domains and problem instances slowed the planning process down. In addition, the number of nodes examined when calling the LP only on the goal state was not greatly different to the normal run, suggesting the consistency check of the LP also did little to prune branches.

These results show that in this case, whilst the small cost of the STN is outweighed by significant search pruning; it is most beneficial to avoid LPs as much as possible using them only when necessary, to check whether a goal is valid or not. However, not solving the LP might have a hindering effect on the search.

Fig 7 supports the above claim. It presents the profiling of the different methods. It shows the correlation between the total runtime and the call for the LP solver. The less time is invested in the solving of the LP the faster the planner reaches the goal state.

The above claim, however, cannot be said to be generally true. We tested the same two domains with a different set of problem instances which also represent a problem that may arise in real life. Here, instance n-1 contains n legs (0..n-1) with 6 observations that must take place in each leg (i.e. for $k \in 1...n$, i $\in [6k, 6k+5]$ contains$_{lk,oi}$) all of which are required in the goal (and require a different one of 6 pieces of equipment). There is no choice over which leg each observation can occur in (i.e. if contains$_{l1,o1}$ is defined then there does not exist any other leg l_i such that contains$_{li,o1}$ is defined); except for o_5, which can be performed in either leg$_0$, or leg$_{n-1}$ (contains$_{l1,o5}$ and contains$_{ln-1,o5}$). To compensate for this leg_{n-1} has only 5 additional observations (o_{6n} to o_{6n+4}) rather than the usual 6. The instances are engineered such that the duration of leg$_0$ is too short for o_6 to fit inside: target-start$_{o6}$ is defined such that waiting long enough for (target-start$_{o6} \leq$ flown$_{leg1}$) to be satisfied, means o_6 cannot finish within leg_0 as illlustrated in



(a) Update Only Variables on which the Current Action is Operating

(b) Only Solve LP for Consistency

(c) Solve LP on Goal States Only

Figure 7: Profiling Suggested Improvements, Single Observation Per-Leg, Instance 5

	Nodes			Nodes STN		Nodes LP	
Instance	Generated	Expanded	Evaluated	Checked	Pruned	Checked	Pruned
1	3510	1316	3393	3470	112	3358	0
2	11170	3631	10956	11111	208	10903	0
3	24639	7778	24280	24523	352	24171	0
4	38244	11309	37812	38108	425	37683	0
5	38981	11336	38549	38840	425	38415	0

Table 4: Nodes in Search, Multi Observation Per Leg

Fig 3b. leg_n is, however, long enough to accomodate o_6 with the precondition satisfied.

Fig 6 presents the results of these multiple-observations per leg instances. Fig 6a on the domain with the precondition for the total-flown in the "configure" action, and Fig 6b without. Table 4 presents the number of nodes using option 1 and the normal configuration, as did Table 3 in the previous set of instances.

As can be seen in Fig 6, on these instances when solving without checking for consistency at each state (option 3) the planner timed out (1000 seconds). Inspection of the states the planner visited during the forward search revealed that this is because the planner added the invalid action to the first leg, and went on searching down the branch. When it reached the goal state it found that it was inconsistent, and started backtracking, but never backtracked enough to find the valid solution. This can also be inferred from Table 4, unlike in the previous case, search makes use of the updated bounds on the variables to prune some branches, similarly actions are being marked as innaplicable in the search (specificially it can infer $target - start_{o6} \leq flown_{leg1}$ will never be satisfied so does not consider putting o_6 in leg_0).

When solving using the second option, the planner timed out as well. This is because the planner kept trying to add the non-valid action, and found it inconsistent. Since there are ordering constraints on the observations, the planner would try to fit the inconsistent observation as the first observation, then, when failed, it would try it as the second, then the third, and so on. This is an expensive process, and it timed out as well.

In this instance updating the bounds was beneficial. Updating only for the active variables was more efficient, as this guaranteed that each variable would be updated at least once, and therefore narrowed down the search space by marking the problematic observation as a non valid action for the first leg.

7 Discussion and Conclusions

The profiling results presented here suggest that the Simple Temporal Network (STN) is much faster than the LP solver. In the domain discussed, which was supplied by an industrial partner, it was shown that the time to solve many LPs which grow in size may not be negligible, and may lead to the planner having difficulties reaching a solution.

Four options were proposed for reducing planning time: updating the bounds on fewer variables by selecting only those currently undergoing continuous numeric change; not updating the bounds at all (solving the LP only for consistency); using an STN on non goal states (while solving the LP only on possible goal states) and not using any solver on a non-goal state (calling the STN and LP only at a goal state).

Solving the STN on non-goal states allows a large number of states to be explored quickly. However, this is only useful in the cases in which actions cannot be marked as inapplicable by their numerical preconditions (they can only be marked as such by their propositions). Using this option in problems with non-cosmetic numeric precondition might cause the planner to search down a branch that is not valid, and to remain in that branch for too long to practically be able to reach a solution.

Solving the LP only to check consistency speeds search up by avoiding LP calls to determine variable bounds. But, again, this increases the branching factor, generating more states, thus slowing the search down.

The last option, not calling any solver on non-goal, was shown to be inefficient, it carries no advatages over the others.

Finally, updating only the active variables was found to be a good compromise. It reduces the per-state LP overheads compared to the default configuration of OPTIC, with a net reduction in planning time; in principle, it has a higher branching factor, so it is not guaranteed to pay off, but we did not encounter such a case in this work.

These four options have been implemented in an updated version of OPTIC, allowing the user to choose from them if needs be. Future research would involve the automatic identification of cases in which per-node LP solving is non beneficial, and the selective update of the variables to facilitate faster search.

Acknowledgements

This work was supported by the UK Engineering and Physical Sciences Research Council (EPSRC) grant EP/R511559/1

(Deployment of Expressive Continuous Numeric Planners in Large Scale Applications).

References

Benton, J.; Coles, A. J.; and Coles, A. 2012. Temporal planning with preferences and time-dependent continuous costs. In *ICAPS*, volume 77, 78.

Cashmore, M.; Fox, M.; Long, D.; and Magazzeni, D. 2016. A Compilation of the Full PDDL+ Language into SMT. In *Proceedings of ICAPS*.

Chien, S.; Rabideau, G.; Knight, R.; Sherwood, R.; Engelhardt, B.; Mutz, D.; Estlin, T.; Smith, B.; Fisher, F.; Barrett, T.; Stebbins, G.; and Tran, D. 2000. Aspen - automated planning and scheduling for space mission operations. In *in Space Ops*.

Coles, A. I.; Fox, M.; Halsey, K.; Long, D.; and Smith, A. J. 2009. Managing concurrency in temporal planning using planner-scheduler interaction. *Artificial Intelligence* 173.

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

Coles, A. J.; Coles, A. I.; Fox, M.; and Long, D. 2012. Colin: Planning with continuous linear numeric change. *Journal of Artificial Intelligence Research* 44:1–96.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49.

Do, M. B., and Kambhampati, S. 2001. Sapa: a domain-independent heuristic metric temporal planner. In *Proc. European Conf. on Planning (ECP'01)*.

Eyerich, P.; Mattmüller, R.; and Röger, G. 2009. Using the context-enhanced additive heuristic for temporal and numeric planning. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS 2009)*.

Fernández-González, E.; Karpas, E.; and Williams, B. C. 2017. Mixed discrete-continuous planning with convex optimization. In *AAAI*, 4574–4580.

Fox, M., and Long, D. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of artificial intelligence research* 20:61–124.

Fox, M.; Long, D.; and Magazzeni, D. 2011. Automatic construction of efficient multiple battery usage policies. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Three*, IJCAI'11, 2620–2625. AAAI Press.

Obes, J. L.; Sarraute, C.; and Richarte, G. 2013. Attack planning in the real world. *CoRR* abs/1306.4044.

Piotrowski, W.; Fox, M.; Long, D.; Magazzeni, D.; and Mercorio, F. 2016. *Heuristic planning for PDDL+ domains*, volume 2016-January. International Joint Conferences on Artificial Intelligence. 3213–3219.

Scala, E.; Haslum, P.; Thiébaux, S.; and Ramirez, M. 2016. Interval-based relaxation for general numeric planning. In *ECAI*, 655–663.

Author Index