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. $\forall t \in T, \forall i \in \{1, \dots, N\}$:

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

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):

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

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

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

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

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:

$\forall t: d_t \leq p_t$ (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.

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

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

$((l_{n_i}^{dur} = l_n^{durMax}) \wedge (l_{n_i}^{isShed} = 0))\ )$

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:

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

Separation for non-periodic loads

$l_n^{minOff} + 1 \leq l_{n_{i+1}}^{start} - l_{n_i}^{start} - l_{n_i}^{dur} \leq l_n^{maxOff}$ (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