Proceedings of the 7th

Scheduling and Planning Applications woRKshop

# SPARK 2013

Rome, Italy - June 11, 2013

*Edited By:*

**Luis Castillo Vidal, Steve Chien, Riccardo Rasconi**

## Organizing Commitee

**Luis Castillo Vidal**
IActive, Spain

**Steve Chien**
Jet Propulsion Laboratory, USA

**Riccardo Rasconi**
ISTC-CNR, Italy

## Program committee

**Laura Barbulescu**, Carnegie Mellon University, USA
**Mark Boddy**, Adventium, USA
**Luis Castillo**, IActive, Spain
**Steve Chien**, Artificial Intelligence Group, Jet Propulsion Laboratory, USA
**Gabriella Cortellessa**, ISTC-CNR, Italy
**Minh Do**, SGT Inc., NASA Ames, USA
**Patrik Haslum**, NICTA, Australia
**Russell Knight**, Artificial Intelligence Group, Jet Propulsion Laboratory, USA
**Jana Koehler**, Hochschule Luzern, Lucerne Univ. of Applied Sciences and Arts, Switzerland
**Nicola Policella**, ESA-ESOC, Germany
**Riccardo Rasconi**, ISTC-CNR, Italy
**Bernd Schattenberg**, University of Ulm, Germany
**Tiago Vaquero**, University of Toronto, Canada
**Ramiro Varela**, Universityof Oviedo, Spain
**Gèrard Verfaillie**, ONERA, France
**Neil Yorke-Smith**, American University of Beirut, Lebanon, and SRI International, USA
**Terry Zimmerman**, SIFT, USA

## Foreword

Application domains that entail planning and scheduling (P&S) problems present a set of compelling challenges to the AI planning and scheduling community, from modeling to technological to institutional issues. New real-world domains and problems are becoming more and more frequently affordable challenges for AI. The international Scheduling and Planning Applications woRKshop (SPARK) was established to foster the practical application of advances made in the AI P&S community. Building on antecedent events, SPARK'13 is the seventh edition of a workshop series designed to provide a stable, long-term forum where researchers and practitioners can discuss the applications of planning and scheduling techniques to real-world problems. The series webpage is at http://decsai.ugr.es/~lcv/SPARK/

We are once more very pleased to continue the tradition of representing more applied aspects of the planning and scheduling community and to perhaps present a pipeline that will enable increased representation of applied papers in the main ICAPS conference.

We thank the Program Committee for their commitment in reviewing. We thank the ICAPS'13 workshop and publication chairs for their support.

Edited by
Luis Castillo Vidal, Steve Chien and Riccardo Rasconi

## Table of Contents

**Dynamic Scheduling of Electric Vehicle Charging under Limited Power and Phase Balance Constraints.........................................................1**
*Alejandro Hernandez Arauzo, Jorge Puente Peinador, Miguel A. González, Ramiro Varela and Javier Sedano*

**Integrating Planning, Execution and Diagnosis to Enable Autonomous Mission Operations.........................................................9**
*Jeremy Frank, Gordon Aaseng, Michael Dalal, Charles Fry, Charles Lee, Rob McCann, Sriram Narasimhan, Lilijana Spirkovska, Keith Swanson, Lui Wang, Arthur Molin and Larry Garner*

**Scheduling a Multi-Cable Electric Vehicle Charging Facility.........................................................20**
*Tony T. Tran, Mustafa K. Dogru, Ulas Ozen and Chris Beck*

**A Steady-State Model for Automated Sequence Generation in a Robotic Assembly System.........................................................27**
*Kimitomo Ochi, Alex Fukunaga, Chikako Kondo, Munehiko Maeda, Fumio Hasegawa and Yukihiro Kawano*

**Generating Alternative Plans for Scheduling Personal Activities.........................................................35**
*Anastasios Alexiadis and Ioannis Refanidis*

**Adapting Planetary Rover Plans via Action Modality Reconfiguration.........................................................41**
*Enrico Scala, Roberto Micalizio and Pietro Torasso*

**Designing quiet rotorcraft landing trajectories with probabilistic road maps.............................49**
*Bob Morris, Michele Donini, Kristen Brent Venable and Matthew Johnson*

**Synthesizing Fractionated Spacecraft Designs as a Planning Problem.........................................55**
*Minh Do, Martin Feather, David Garcia, Kenneth Hicks, Eric Huang, Dave Kluck, Ryan Mackey, Tuan Nguyen, Jami Shah, Yorgos Stylianos, Raffi Tikidjian, Serdar Uckun, Zihan Zhang and Rong Zhou*

**Compilation of Maintenance Schedules based on their Key Performance Indicators for Fast Iterative Interaction.........................................................63**
*Dara Curran, Roman Van Der Krogt, James Little and Nic Wilson*

**Adaptive Route Planning for Metropolitan-Scale Traffic.........................................................69**
*David Wilkie, Jur van den Berg, Ming Lin and Dinesh Manocha*

**Constraint based Berth Allocation and Ship Scheduling with Upstream Supply Planning.........................................................77**
*Kameshwaran Sampath, Alfiya Tezabwala and Vinayaka Pandit*

**Query Optimization Revisited: An AI Planning Perspective.........................................................84**
*Nathan Robinson, Sheila McIlraith and David Toman*

**A generic constraint-based local search library for the management of an electromagnetic surveillance space mission.........................................................92**
*Cédric Pralet, Guillaume Infantes and Gérard Verfaillie*

# Dynamic Scheduling of Electric Vehicle Charging under Limited Power and Phase Balance Constraints

**Alejandro Hernández**[1] and **Jorge Puente**[1] and **Miguel A. González** and **Ramiro Varela**[1] and **Javier Sedano**[2]
[1]Department of Computer Science,
University of Oviedo, 33271 Gijón (Spain)
e-mail: {alex, puente, ramiro}@uniovi.es
[2]Instituto Tecnológico de Castilla y León (ICTL)
e-mail: javier.sedano@itcl.es

## Abstract

We confront the problem of scheduling the charge of electric vehicles, under limited electric power contract, with the objective of maximizing the users' satisfaction. The problem is motivated by a real life situation where a set of users demand electric charge while their vehicles are parked. Each space has a charging point which is connected to one of the lines of a three-phase electric feeder. We first define the problem as a Dynamic Constraint Satisfaction Problem (DCSP) with Optimization. Then, we propose a solution method which requires solving a number of CSPs over time. Each one of these CSPs requires in its turn solving three instances of a one machine sequencing problem with variable capacity. We evaluated the proposed algorithm by means of simulation across some instances of the problem. The results of this study show that the proposed scheduling algorithm is effective and produces much better results than some classic dispatching rules.

## Introduction

It is well known that the use of Electric Vehicles (EVs) may have a positive impact on the economies of the countries and on the environment, due to promoting the use of alternative sources of energy and relieving the dependency of foreign petrol. At the same time, the emerging fleet of EVs introduces some inconveniences such as the additional load on the power system. However, the charge of EVs is usually more flexible than the conventional load of oil vehicles as in many cases the owners require charging while their vehicles are parked during large time periods. This flexibility may be exploited to design appropriate algorithms for charging control (Wu, Aliprantis, and Ying 2012).

In this paper we consider a real life problem that requires scheduling the charging intervals of a set of EVs that demand power while they are parked in their own spaces within a community car park. A charging station is installed in the car park so that each space has an independent charging point. However, if the power demand is very large during a given time period, not all the requiring vehicles can be charged simultaneously, as the contracted power is limited. So, in these situations, an appropriate scheduling policy is necessary to organize and control the charging intervals of the vehicles along the time they are in the car park (Sedano et al. 2012).

We propose modeling the problem of computing such a schedule in the framework of Dynamic Constraint Satisfaction Problems (DCSP) with Optimization. As it is usual, one problem of this class requires solving a number of CSPs over time. In order to solve each one of these CSPs, we propose an algorithm that requires solving a number of instances of a one machine scheduling problem with variable machine capacity. The scheduling algorithm is evaluated by means of simulation and compared with some dispatching rules such as First Come First Scheduled (FCFS) or Latest Starting Times (LST).

The rest of the paper is organized as follows. In the next section we summarize the aspects of the charging station that are relevant from the point of view of the scheduling algorithm. Then, we define the problem and describe the proposed algorithm to solve it. Finally, we report the results of the experimental study and give some conclusions and ideas for future research.

## Description of the charging station

In this section we summarize the main characteristics of the electrical structure and the operation mode of the charging station. These elements are detailed in (Sedano et al. 2012). Figure 1 shows a schema of the distribution net of the charging station. The net is feeded by a three-phase source of electric power. In the model considered here, each line feeds a number of charging points. The station has about 180 spaces, each one having a charging point which may be in two states: active or inactive. When it is inactive, the charging point is not connected to the electric net, while in active state it is connected to the net and transfers energy at a constant rate (2.3Kw) in the so called mode 1 (Sedano et al. 2012).

The operation of the station is controlled by a distributed system comprising a master and a number of slaves. Every two consecutive charging points in the same line are under the control of the same slave. The master has access to the database where the vehicles' data and the charging schedule are stored. It receives information about the state of the charging vehicles from the slaves, and transmits to the slaves starting times and durations of charging intervals. So the slaves are responsible for activating and deactivating charg-

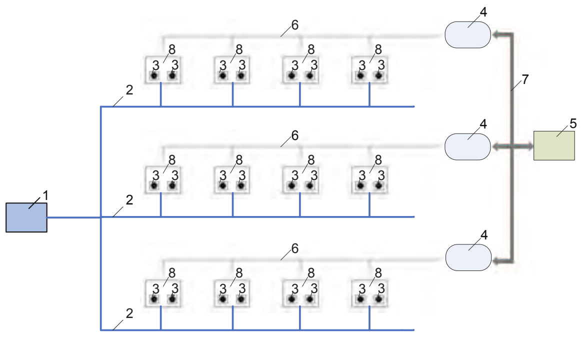


Figure 1: General structure of the distribution net of charging stations. It is formed by different parts such as: (1) power source, (2) three-phase electric power, (3) charging points, (4) masters, (5) server with database, (6) communication RS 485, (7) communication TCP/IP, (8) slaves.

ing points as well as registering asynchronous events such as a new vehicle arriving to the system.

The operating mode of the station is as follows. Each user has one vehicle and one space assigned. These are concrete spaces as each user has to be the owner or the renter of its stall and he cannot use the space of another user. This restriction makes the scheduling problem harder to solve as each stall is connected to one of the three lines of the three-phase feeder and so keeping the balance constraints may not be easy. So, for a vehicle and user can use the station, they have to be registered in the system. When entering in the station, the user has to check in and identify himself and the vehicle. At this time, the user has to connect the vehicle to the charging point and provide the charging time, as well as an expected time, or due date, for taking the vehicle away. These values are then used by the control system to schedule the vehicle, i.e., to establish a starting time. In principle, the vehicle will start to charge at this starting time unless the schedule is modified before it.

There are some constraints that must be satisfied for the station to work properly. For example, although there are 180 spaces available, not all the charging points in these spaces can be activated at the same time. In practice there is a maximum number of vehicles $N$ that can be in charge (actives) at the same time in a line which depends on the contracted power. Also, due to electro-technical and economical reasons, the current consumption in the three lines must be balanced. This condition is considered as a maximum imbalance between any two lines.

In this paper we consider a simplified model of the charging station which make the following assumptions: the user never takes the vehicle away before the declared due date and the battery does not get completely charged before the charging time indicated by the user. Even though they are unrealistic assumptions, the model may be adapted to deal with these situations with nothing more than introducing new asynchronous events.

In principle, each time a new vehicle requires charging, the current schedule may get unfeasible and so a new schedule should be built. However, in order to avoid the system to collapse if many of such events are produced in a very short period of time so that a new schedule cannot be obtained from one event to the next, new schedules are computed at most at time intervals of length $\Delta T$. In order to do that, the protocol is the following: every $\Delta T$ time units a supervisor program, running on the server, checks for the events produced in the last interval. If at least one event was produced that could make the current schedule unfeasible, then the scheduler is launched to obtain a new feasible schedule which is applied from this time on.

## Modeling frameworks

Given the characteristics of the charge scheduling problem, maybe the most appropriate framework for modeling is the dynamic constraint satisfaction problem framework (DCSP) introduced in (Dechter and Dechter 1988). A DCSP is a sequence of CSPs, $\langle P_1, P_2, \ldots, P_n \rangle$, where each $P_i, 1 < i \leq n$ is derived from $P_{i-1}$ by adding and removing a limited number of constraints. Some variants of the DCSP framework has been proposed that capture other characteristics such as dynamic domains of the variables, state variables which are controlled by the physical system and not by the decision maker, or the uncertainty about the presence of some constraints. However, none of these characteristics appears in the version of the charging scheduling problem considered here. All of these and other frameworks are surveyed in (Verfaillie and Jussien 2005).

There are two main types of methods to solve a DCSP: reactive and proactive. A reactive algorithm does not use knowledge of the possible changes, so it may not produce robust solutions, but at the same time it may react better to any kind of change. On the contrary, a proactive method is able to exploit any available knowledge and so it may produce robust or flexible solutions. In both cases, the algorithms may either reuse the solutions of the previous CSP or compute a new solution from scratch. Each of these options has its own advantages and drawbacks. Solution reuse may speed up the calculation of a new solution, but at the same time may prevent the algorithm from obtaining a better one.

A particular case of proactive method is the online stochastic optimization framework (Chang, Givan, and Chong 2000), (Bent and Van Hentenryck 2004), which has been applied to a variety of problems where a set of requests are given over time. Online stochastic optimization algorithms rely on two main components: an offline solver for a set of requests, and a sampler that generates fictitious requests with a given distribution along a time horizon. The online stochastic algorithms solve instances which are composed by subsets of requests including real and fictitious ones, then they take decisions from the solutions of these fictitious instances. So, the application of online stochastic optimization to the charge scheduling problem may make it difficult to maintain the balance constraints if the processing times of the fictitious operations differ much from the charging periods required for the incoming vehicles.

From all the above, we have opted to model the electric vehicle charging scheduling problem, termed PI in the sequel, in the framework of DCSP with optimization and to use a reactive method to solve it. So, in the next sections we give the formal definition of the problem and describe the proposed algorithms. Before this, we also give a formal definition of the problem as a static CSP, which assumes a complete knowledge in advance about the vehicle arrivals. This helps to understand the subsequent dynamic definition.

## Definition of the PI problem as a static CSP

As we have pointed, if the problem data, i.e., the arrival times of the vehicles and their charging times and due dates were known in advance, the problem could be formalized as a static CSP. Even though this is not the case for our problem, we consider here a static version of it. The purpose is twofold, firstly to clarify the overall problem and then to define the simulation framework which will be used in our experimental study. In the next subsections we give the problem data, the goal, the problem constraints and the evaluation function to be optimized.

**Problem data.** In an static instance $\mathbf{P}$ of the PI problem there are 3 charging lines $L_i$, $1 \leq i \leq 3$, each one having $n_i$ charging points. $N > 0$ is the maximum number of charging points that can be active at the same time in each one of the three lines. The line $L_i$ receives a number of $M_i$ vehicles $\{v_{i1}, \ldots, v_{iM_i}\}$ from a time 0 up to a planning horizon. Each vehicle $v_{ij}$ is characterized by an arrival time $t_{ij} \geq 0$, a charging time $p_{ij}$ and a time at which the user is expected to take the vehicle away, or due date, $d_{ij}$ by which the battery of the vehicle should be charged.

There is also a parameter $\Delta \in [0, 1]$ which controls the maximum imbalance among the lines.

**Goal.** The goal is to get a feasible schedule for $\mathbf{P}$, i.e., assigning starting times to the decision variables $st_{ij}$ for each vehicle $v_{ij}$ satisfying the constraints and optimizing the evaluation function.

### Constraints

I. For all vehicle $v_{ij}$, $st_{ij} \geq t_{ij}$.

II. No preemption is allowed, so a vehicle $v_{ij}$ cannot be disconnected before its charging time $C_{ij}$ is reached, i.e., $C_{ij} = st_{ij} + p_{ij}$.

III. The number of active charging points in a line at a given time cannot exceed $N$, i.e.,

$$\max_{(t \geq 0; i=1,2,3)} N_i(t) \leq N \tag{1}$$

where $N_i(t)$ denotes the number of charging points of line $L_i$ which are active during the time interval $[t, t+1)$.

IV. The maximum imbalance between any two lines $L_i$ and $L_j$ is controlled by the parameter $\Delta$ as

$$\max_{(t \geq 0; 1 \leq i,j \leq 3)} (|N_i(t) - N_j(t)|/N) \leq \Delta \tag{2}$$

**Evaluation function.** The evaluation function is the total tardiness defined as

$$\sum_{i=1,2,3; j=1,\ldots,M_i} \max(0, C_{ij} - d_{ij}) \tag{3}$$

which must be minimized.

## Definition of the PI problem as a DCSP

The PI problem may be naturally considered as a dynamic problem due to the fact that the arrival of vehicles is not known in advance. For this reason, an instance $\mathbf{P}$ can be defined as a sequence of instances, $\mathbf{P_1}, \mathbf{P_2}, \ldots$ of a static CSP termed PII. Each $\mathbf{P_k}$ is defined (see the next Section) from the set of vehicles in the system which have not yet completed their charging periods.

To solve this problem, we adopted here a similar strategy to that used in (Rangsaritratsameea, Ferrell, and Kurzb 2004) for the dynamic Job Shop Scheduling problem where the jobs are unknown until they arrive. In that paper, the authors propose to build a new schedule at each ”rescheduling point” combining all previous operations that have not started processing together with operations arriving after the previous rescheduling point.

Due to technological restrictions, we do not consider rescheduling each time a new vehicle arrives. Instead, we consider rescheduling each time the Supervisor is activated. The new schedule involves the vehicles which have arrived from the previous point together with all the vehicles in the system which have not yet started to charge.

### Solving the dynamic PI problem

Algorithm 1 shows a simulation of the actual algorithm to solve a dynamic PI problem. In the simulation, the problem data and the sequence of times for the Supervisor to be executed are given to the algorithm. The algorithm iterates on this sequence of times $T_1, T_2, \ldots$. In the iteration $k$, i.e., at

**Algorithm 1** Solving the PI problem.

```
Require: The data of a P instance of the PI problem: $t_{ij}$, $p_{ij}$ and
    $d_{ij}$ for all vehicles $v_{ij}$; and the sequence of times $T_1, T_2, \ldots$ at
    which the Supervisor is activated.
Ensure: A schedule S for P defined by the time each vehicle starts
    to charge $st_{ij}$ and the total tardiness produced by this schedule.
  $S = \emptyset$;
  for all $k = 1, 2, \ldots$ do
    if a new vehicle $v_{ij}$ has arrived at a time $t = t_{ij} \in (T_{k-1}, T_k]$
    then
      Generate a new instance $\mathbf{P_k}$ of the problem PII with all
      vehicles $v_{ij}$ s.t. $t_{ij} \leq T_k$ and that have not started charging
      yet;
      Calculate a solution $S$ for the instance $\mathbf{P_k}$; {A solution $S$
      defines starting times $st^*_{ij} \geq T_k$ to schedule all vehicles
      $v_{ij}$ that are not active at $T_k$}
    end if
    Establish $S$ as the current solution along $[T_k, T_{k+1})$; i.e., for
    each $st^*_{ij} \in S$ such that $T_k \leq st^*_{ij} < T_{k+1}$, set $st_{ij} = st^*_{ij}$
    in the final schedule S, so that $v_{ij}$ starts charging at $st^*_{ij}$;
  end for
  return the schedule S for P and its total tardiness;
```

time $T_k$, a new instance $\mathbf{P_k}$ of PII is created if some vehicle arrived from the previous instant $T_{k-1}$. This instance is solved and the new solution replaces the current one from $T_k$ on. If no vehicle has arrived from $T_{k-1}$ to $T_k$ then the current solution can remain active until the next iteration. The current solution is applied during the time it is active. This means that the vehicles start charging at the times $st^*_{ij}$ given in the current solution, so $st_{ij}$ is fixed to $st^*_{ij}$ in the solution to the $\mathbf{P}$ instance, and disconnected at their expected times $C_{ij} = st_{ij} + ps_{ij}$.

## Definition of the PII problem

The PII problem can be defined as a static CSP as follows: In an instance $\mathbf{P_k}$, we are given a set of vehicles $\{v_{i1}, \ldots, v_{il_i}, \ldots, v_{im_i}\}$ at time $T_k$ in each line $L_i$, $1 \leq i \leq 3$. Each vehicle $v_{ij}$ requires a charging time $p_{ij}$ and has a due date $d_{ij}$. The vehicles $v_{i1}, \ldots, v_{il_i}$ are already active, as they started to charge at a time $t < T_k$ and have not yet finished, i.e., $C_{ij} = st_{ij} + p_{ij} > T_k$. While the vehicles $v_{il_i+1}, \ldots, v_{im_i}$ have not yet started to charge. So, in the iteration $k$, the capacity of the line $L_i$ to charge new vehicles, denoted $M^k_i(t)$ is given by

$$M^k_i(t) = N - \sum_{1 \leq j \leq l_i} X_{ij}(t), \;\; t \geq T_k \tag{4}$$

where

$$X_{ij}(t) = \begin{cases} 1, & t < C_{ij} \\ 0, & t \geq C_{ij} \end{cases} \tag{5}$$

The objective is to obtain a feasible schedule for all vehicles in the system such that all of them can be sorted out, even if no new vehicles arrive after $T_k$. This requires assigning starting times $st^*_{ij}$ to all vehicles unscheduled at time $T_k$, which are compatible with the starting times of the vehicles already scheduled by $T_k$. This means that all the constraints naturally derived from the static PI problem must be satisfied.

Also, the evaluation function will be, in principle, minimizing the total tardiness. However, as the solution to the instance $\mathbf{P_k}$ is expected to be useful along a short time period (until the arrival of a new vehicle), we will try to maximize the number of charging vehicles at the beginning. So, we could consider a time horizon $t_h$ at which a new event may be expected and try to maximize the charge along the interval $[T_k, T_k + t_h]$. This new objective may be expressed as maximizing

$$\int_{T_k}^{T_k+t_h} (N_1(t) + N_2(t) + N_3(t))dt \tag{6}$$

where $N_i(t); i = 1, 2, 3$ denotes the number of active vehicles in line $L_i$ at time $t$.

## Solving the PII problem

The PII problem is really hard to solve because of the constraint derived from constraint (IV) of the static PI problem, which for the instance $\mathbf{P_k}$ may be expressed as

$$\max_{(t \geq T_k; 1 \leq i,j \leq 3)} (|N_i(t) - N_j(t)|/N) \leq \Delta \tag{7}$$

It is not easy to build a schedule satisfying this constraint and at the same time maximizing expression (6). To solve this problem, we propose to use two simple dispatching rules and a more sophisticated algorithm based on problem decomposition.

**Solving PII with a dispatching rule.** We propose to use a simple dispatching rule termed LTS (Latest Starting Time) to solve each $\mathbf{P_k}$ instance. This rule works as follows: the unscheduled vehicles at time $T_k$ in the system are sorted in accordance with their latest starting times given by $d_{ij} - p_{ij}$, then these vehicles are scheduled in this order and each one is given the earliest starting time such that the scheduled vehicles satisfy all the constraints. This rule can be easily implemented, but the restriction that the balance constraint must be satisfied after scheduling each vehicle is very hard and may prevent the algorithm from reaching near optimal solutions.

**Solving PII by problem decomposition.** We also propose a method that uses problem decomposition in the following way. First of all, we establish a profile of maximum charge, $N^{max}_i(t), i = 1, 2, 3$, for each one of the three lines; so that these profiles satisfy the constraint

$$\max_{(t \geq T_k; 1 \leq i,j \leq 3)} (|N^{max}_i(t) - N^{max}_j(t)|/N) \leq \Delta \tag{8}$$

Then, we try to obtain a schedule for each one of the lines, so that $N_i(t)$ is as close as possible to $N^{max}_i(t)$ for $t \geq T_k$ while it satisfies the constraint

$$N_i(t) \leq N^{max}_i(t), \;\; t \geq T_k \tag{9}$$

Hence, combining the solutions to the three lines may give rise to a feasible solution to the PII instance. If not, the profiles $N^{max}_i(t)$ are adjusted and new schedules are computed for one or more lines.

The problem of calculating a schedule for a line subject to a maximum load is denoted PIII herein and the instance of this problem which consist in scheduling the vehicles in the line $L_i$, subject to the profile $N^{max}_i(t)$ at time $T_k$, is denoted $\mathbf{P_{ki}}$. So, our proposed method starts from some initial profiles and then these profiles are updated as long as the solutions obtained to the $\mathbf{P_{ki}}$ instances, $1 \leq i \leq 3$, derived from a $\mathbf{P_k}$ instance, do not make up a solution to the $\mathbf{P_k}$ instance.

Algorithm 2 describes the calculation of a solution to a $\mathbf{P_k}$ instance. The algorithm starts from trivial profiles $N^{max}_i(t)$ and then iterates until a solution is reached. In each iteration, it solves the three $\mathbf{P_{ki}}$ instances subject to the profiles $N^{max}_i(t)$. If these solutions make up a solution for $\mathbf{P_k}$, the algorithm finishes; otherwise, some profile is adjusted from the earliest time $t'$ at which an imbalance is detected onwards. In this way, the profiles are maintained as large as possible at the beginning and so, hopefully, the evaluation function given in expression (6) is maximized. The adjustment of the profiles is the most controversial operation in this algorithm. We will reconsider this issue later.

**Algorithm 2** Solving the PII problem.

```
Require: The data of an instance $\mathbf{P_k}$: $p_{ij}$ and $d_{ij}$ for all unscheduled vehicles $v_{ij}$ that arrived by $T_k$; and the values $st_{ij}$ and $p_{ij}$ for all vehicles scheduled before $T_k$ such that $st_{ij} + p_{ij} \geq T_k$.
Ensure: A schedule $\mathbf{S}$ for $\mathbf{P_k}$ defined by the time each vehicle starts to charge $st^*_{ij}$ and the total tardiness produced by this schedule.
  Set the initial profiles to $N_i^{max}(t) = N, t \geq T_k, 1 \leq i \leq 3$;
  while $\mathbf{P_k}$ remains unsolved do
    Solve the instances $\mathbf{P_{ki}}$ under the current profiles $N_i^{max}(t)$;
    {The three PIII instances get solved with charge profiles $N_i(t)$}
    Let $t' \geq T_k$ be the earliest time such that an imbalance exists, i.e., $N_i(t') - N_j(t') > \Delta \times N$ for some $1 \leq i, j \leq 3$;
    if there exists such a time $t'$ then
      Adjust the profile of maximum load for the line $L_i$ so that $N_i^{max}(t) \leq N_j(t) + \Delta \times N, t \geq t'$;
    else
      The solutions to the $\mathbf{P_{ki}}$ instances, $1 \leq i \leq 3$, make up a solution to $\mathbf{P_k}$;
    end if
  end while
  return the schedule $\mathbf{S}$ for $\mathbf{P}$ and its total tardiness;
```

## Definition of the PIII problem

In an instance $\mathbf{P_{ki}}$ of the PIII problem we are given the set of vehicles $\{v_{i1}, \ldots, v_{il_i}, \ldots, v_{im_i}\}$ at time $T_k$ in the line $L_i$. Additionally, we are given a maximum charge profile for the line $L_i$, $N_i^{max}(t), t \geq T_k$.

The objective is to obtain a schedule for the vehicles, i.e., starting times $st^*_{ij} \geq T_k$ for the inactive vehicles $v_{il_i+1}, \ldots, v_{im_i}$, such that the following two constraints, derived from the PII instance, are satisfied:

i. $st^*_{ij} \geq T_k$, for each inactive vehicle.

ii. $N_i(t) \leq N_i^{max}(t)$, for all $t \geq T_k$.

The evaluation function is the total tardiness, defined as

$$\sum_{j=l_i+1,\ldots,m_i} \max(0, C_{ij} - d_{ij}) \qquad (10)$$

which must be minimized.

### Solving the PIII problem

The $\mathbf{P_{ki}}$ problem can be viewed as that of scheduling a number of $m_i - l_i$ jobs, all of them available at time $T_k$, on a machine whose capacity varies along the time, and the objective is minimizing the total tardiness. The processing time of the jobs are the charging times of the vehicles $v_{il_i+1}, \ldots, v_{im_i}$, respectively. Each job can use only one slot of the machine at a time. In other words, the machine is a cumulative resource with variable capacity. Cumulative scheduling has been largely considered in the literature, mainly in the context of the Resource Constrained Project Scheduling Problem (RCPSP). However, to the best of our knowledge, cumulative resources with time dependent capacity has not been considered yet.

In our case, the capacity of the machine is defined by the profile $N_i^{max}(t)$ and the vehicles already scheduled $v_{i1}, \ldots, v_{il_i}$, which complete charging at times $C_{ij} \geq T_k$. So, the capacity of the machine may be expected to be increasing at the beginning, as long as the scheduled vehicles complete charging, and decreasing at the end, as the profiles $N_i^{max}(t)$ are non increasing along time. To be concrete, the capacity of the machine at time $t$ for the line $L_i$, $Cap_i^k(t)$, is calculated as

$$Cap_i^k(t) = \min(M_i^k(t), N_i^{max}(t)), \;\; t \geq T_k \qquad (11)$$

We denote this problem as $(1, Cap(t)||\sum T_i)$ following the conventional notation $(\alpha|\beta|\gamma)$ proposed in (Graham et al. 1979).

**Solving the $(1, Cap(t)||\sum T_i)$ problem.** In the simple case where the capacity $Cap(t)$ is non decreasing, the problem is equivalent to the problem of identical parallel machines with variable availability denoted $(P, NC_{inc}||\sum T_i)$ following the notation used in (Schmidt 2000), where $P$ is the number of parallel machines and $N_{inc}$ denotes that the availability of machines is non decreasing along the time. Scheduling problems with machine availability appear in many situations, for example when maintenance periods are considered, with different profiles of machine availability. This kind of problems are surveyed in (Ma, Chu, and Zuo 2010).

In (Koulamas 1994), the $(P||\sum T_i)$ problem, in which all the machines are continuously available, is proved to be at least binary NP-hard. An efficient simulated annealing algorithm for this problem is proposed in (Sang-Oh Shim and Kim 2007). In this algorithm, the starting solution is obtained by means of the apparently tardiness rule. This rule was adapted for similar problems in (Kaplan and Rabadi 2012), to deal with ready times and due date constraints. In this paper, we propose to adapt this rule to solve the $(1, Cap(t)||\sum T_i)$ problem as follows: let $\Gamma(\alpha)$ the earliest starting time for an unscheduled job in the partial schedule $\alpha$ built so far. Then for all unscheduled jobs that can start at $\Gamma(\alpha)$ a selection probability is calculated as

$$\Pi_j = \frac{1}{p_j} \exp\left[\frac{-max(0, d_j - \Gamma(\alpha) - p_j)}{g\overline{p}}\right] \qquad (12)$$

where $\overline{p}$ is the average processing time of the jobs and $g$ is a look-ahead parameter to be fixed empirically. These probabilities may be applied deterministically, i.e., the job $j$ with the largest probability is selected to be scheduled next, or probabilistically. In principle, we will consider the first option in the experimental study.

## Profiles of maximum load

As we have pointed out, the balance among the lines is the most critical issue of the whole charge scheduling problem. In order to deal with it, we propose to use the following model for the profiles of maximum load. A profile $N_i^{max}(t)$ is given by a stepwise non increasing function of the form:

$$N_i^{max}(t) = \begin{cases} \delta_j & \tau_j \leq t < \tau_{j+1}, 1 \leq j < k \\ \delta_k & \tau_k \leq t \end{cases} \qquad (13)$$

where $\delta_1 > \cdots > \delta_k$ and $\tau_1 < \cdots < \tau_k$, $k \geq 1$. We represent this profile as a sequence of tuples as: $\langle(\delta_1, \tau_1), (\delta_2, \tau_2), \ldots, (\delta_k, \tau_k)\rangle$.

In Algorithm 2, the initial profiles are $N_i^{max}(t) = \langle(N, 0)\rangle$ for all three lines. Then, these profiles are adjusted as long as new imbalances are found after the solutions of the three PIII instances. In particular, when an imbalance of the form $N_i(t') - N_j(t') > \Delta \times N$ is detected, then the profile $N_i^{max}(t)$ is modified so as a new element $(\delta, \tau) = (\Delta \times N + N_j(t'), t')$ is inserted and all tuples $(\delta_j, \tau_j)$ with $\delta_j > \delta$ and $\tau_j > \tau$ are removed from $N_i^{max}(t)$.

This is a very simple model which helps to keep the load in the three lines as large as possible at the beginning, hopefully along the interval $[T, T + t_h]$. However, it may have some inconvenience as well. For example, a new imbalance may be produced at a time just after to $t'$. To avoid this drawback, we could adjust the new tuple as $(\delta - \delta_H, \tau - \tau_H)$, where $\delta_H \geq 0$ and $\tau_H \geq 0$ are parameters to be established empirically. Also, the next imbalance may be at a time lower than $t'$ due to the non-preemption constraint. In any case, the value of $N_i^{max}(t)$, for each time $t$, is non increasing along the subsequent adjustments. This guarantees that Algorithm 2 terminates after a finite number of steps.

## Experimental study

As it was pointed out, we evaluated the scheduling algorithm by simulation. To do that, we have firstly defined a set of instances of the PI problem and then we implemented a simulator to run the Algorithm 1. In the next two subsections, we give the details of the benchmark defined and summarize the results of the experimental study.

### Benchmark set

We consider that the charging station is installed in a car park with 180 spaces distributed uniformly in the three lines. We have generated some benchmarks[1] considering a time horizon of one day and a profile for arrival times which are based on the expected behavior of the users in some weekdays. Also, we have considered different demand and due date profiles.

Figure 2 shows the arrival profile of the vehicles along the day. As we can observe, there are peaks of arrivals at four different times of the day. The processing times $p_{ij}$ and due dates $d_{ij}$ follow the profiles represented in Table 1. We consider four different profiles depending on the state of the battery at the arrival time. The second column of the table represents the probabilities of each case and the two last columns represent the probability distributions for the values $p_{ij}$ and $d_{ij}$, which are given by means of normal distributions.

We have defined two types of instances, in the first one (type 1), 60 vehicles arrive at each line $L_i$ along the day and demand charging, while in the second (type 2) the vehicles are 108 in $L_1$, 54 in $L_2$ and 18 in $L_3$, i.e., 60%, 30% and 10% respectively. So, in the later case we may expect that the scheduling algorithm has to control many situations

[1]These instances are available at http://www.di.uniovi.es/tc (Repository).

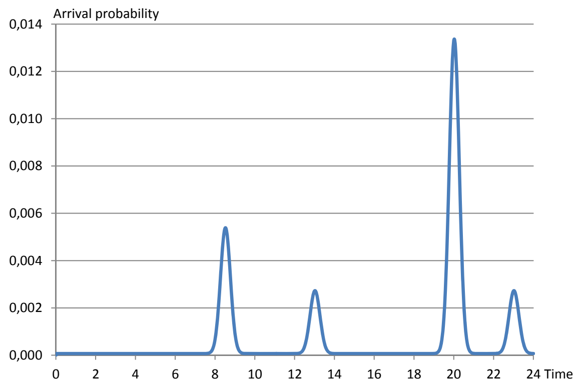


Figure 2: Arrival profile of vehicles along a day. x-axis represents the time of the day from 0 to 24 hours and the y-axis represents the arrival probability.

Table 1: Charging time and due date profiles used to generate PI instances. $N(x, y)$ denotes a normal distribution with mean $x$ and standard deviation $y$. Time is given in hours.

| Case | Prob. | $p_{ij}$ | $d_{ij}$ |
|---|---|---|---|
| 1 | 0.1 | $N(2, 1)$ | $t_{ij} + \max(p_{ij}, N(4, 2))$ |
| 2 | 0.3 | $N(5, 1.5)$ | $t_{ij} + \max(p_{ij}, N(6, 2))$ |
| 3 | 0.3 | $N(6.5, 0.75)$ | $t_{ij} + \max(p_{ij}, N(8, 2))$ |
| 4 | 0.3 | $N(8.8, 0.6)$ | $t_{ij} + \max(p_{ij}, N(11, 2))$ |

of imbalance among the lines in order to build a feasible schedule. Also, we will consider different values for the imbalance parameter $\Delta$ (0.2, 0.4, 0.6 and 0.8) and for the maximum number of vehicles that can be charging at the same time in a line $N$ (20, 30 and 40). 30 instances were generated for each combination of type, $\Delta$ and $N$, so we have 720 instances in all.

### Evaluation of the proposed algorithm

Our main purpose is to evaluate the proposed algorithm to solve the problem PI, termed PD (Problem Decomposition) herein, under different demand conditions. We also compare it with two algorithms: a single dispatching rule FCFS (First Come First Scheduled) that could be implemented by a human operator, and the aforementioned scheduling algorithm that uses the LST rule. In FCFS, the vehicles are scheduled in the order they arrive to the car park and then each one is scheduled at the earliest time such that all the constraints of the problem PI are satisfied. After this, the starting time is never changed.

Table 2 summarizes the results of these experiments. Each line of the table shows the average tardiness obtained for the 30 instances of the same type and the same values of $\Delta$ and $N$. The parameter $\Delta T$ was set to 120 s. Regarding the time taken by the algorithms, in the case of PD it depends on the number of adjustments required to reach a solution, and it is larger for this algorithm than it is for the dispatching rules FCFS and LST. However, in no case the time required by

PD was larger than 1 s., which is negligible w.r.t. to 120 s. between two consecutive executions of the scheduler. As we can observe the total tardiness is lower with PD than it is with FCFS and LST in almost all the cases. In average, the total tardiness obtained by FCFS and LTS in this benchmark is very similar and it is about $33,4\%$ larger than that obtained by PD.

Analyzing the schedules obtained for the algorithms, we have observed that PD is able to adjust the imbalance of the schedules up to the limit allowed by the parameter $\Delta$, at difference of the dispatching rules, and this is the very reason for its superior performance. As we have conjectured, one of the reasons of the poor performance of FCFS and LST is that they have to keep the imbalance constraint after each operation is scheduled, what requires delaying the starting time of many operations.

Therefore, from these results, we can conclude that the proposed algorithm PD is effective to solve the problem PI.

**Adjustments of the maximum charge profiles $N_i^{max}(t)$.** It is also worth analyzing the number of adjustments required to reach a solution to a PII instance, as it may have an important impact on the time required to reach a solution to the whole PI problem.

Table 2: Summary of results from PD, LST and FCFS on two instances of types 1 and 2 with different values of the parameters $\Delta$ and $N$. The values of tardiness are given in minutes.

| PI Instance | | | Total Tardiness | | |
|---|---|---|---|---|---|
| Type | $N$ | $\Delta$ | FCFS | LST | PD |
| 1 | 20 | 0.2 | 2,03E+06 | 2,02E+06 | 9,26E+05 |
| | | 0.4 | 7,97E+05 | 7,96E+05 | 5,09E+05 |
| | | 0.6 | 5,73E+05 | 5,70E+05 | 4,62E+05 |
| | | 0.8 | 5,45E+05 | 5,42E+05 | 4,57E+05 |
| | 30 | 0.2 | 6,80E+05 | 6,94E+05 | 2,14E+05 |
| | | 0.4 | 1,08E+05 | 1,04E+05 | 5,87E+04 |
| | | 0.6 | 6,73E+04 | 6,69E+04 | 5,50E+04 |
| | | 0.8 | 6,61E+04 | 6,56E+04 | 5,50E+04 |
| | 40 | 0.2 | 2,21E+05 | 2,32E+05 | 7,76E+04 |
| | | 0.4 | 9,62E+03 | 9,01E+03 | 3,35E+03 |
| | | 0.6 | 1,50E+03 | 1,49E+03 | 1,04E+03 |
| | | 0.8 | 1,44E+03 | 1,43E+03 | 1,04E+03 |
| 2 | 20 | 0.2 | 2,00E+07 | 2,02E+07 | 1,53E+07 |
| | | 0.4 | 7,56E+06 | 7,53E+06 | 5,55E+06 |
| | | 0.6 | 3,98E+06 | 3,97E+06 | 2,76E+06 |
| | | 0.8 | 2,41E+06 | 2,42E+06 | 1,78E+06 |
| | 30 | 0.2 | 1,15E+07 | 1,15E+07 | 8,70E+06 |
| | | 0.4 | 3,44E+06 | 3,43E+06 | 2,57E+06 |
| | | 0.6 | 1,40E+06 | 1,40E+06 | 9,69E+05 |
| | | 0.8 | 7,21E+05 | 7,20E+05 | 5,40E+05 |
| | 40 | 0.2 | 7,37E+06 | 7,37E+06 | 5,53E+06 |
| | | 0.4 | 1,71E+06 | 1,72E+06 | 1,31E+06 |
| | | 0.6 | 5,99E+05 | 6,00E+05 | 4,22E+05 |
| | | 0.8 | 2,60E+05 | 2,59E+05 | 1,99E+05 |
| | Average | | 9,94E+06 | 9,95E+06 | 7,46E+06 |

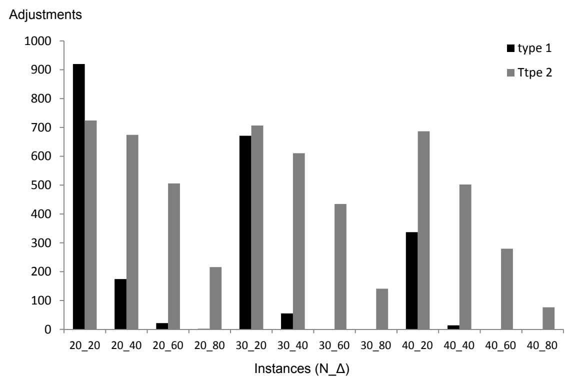


Figure 3: Number of adjustments of the maximum charge profiles depending on the type of the problems (type 1 or type 2), the maximum imbalance $\Delta$ (0.2, 0.4, 0.6, 0.8) and the maximum number of active vehicles in one line N (20, 30, 40). The x-axis represents the group of instances for each type of problem ($\Delta \times 100\_N$) and the y-axis the average number of profile adjustments made to solve the PI instances of each class.

Figure 3 shows the average number of adjustments required to solve each one of the 24 groups of instances defined by the same type and values of $N$ and $\Delta$. For the instances of type 2, the number of adjustments depends on the allowed imbalance $\Delta$ and, as can be expected, this number is in inverse ratio with $\Delta$. The adjustments also depend weakly on the maximum number of active vehicles in a line $N$, this dependency being also in direct ratio.

Regarding the instances of type 1, where the three lines receive the same proportion of vehicles, these dependencies are much more stronger than they are in the type 2. For the largest values of $\Delta$ the number of adjustments is negligible. However, for $\Delta = 0.2$, i.e., when the allowed imbalance is very low, the number of adjustments is really large. The reason for this is that for the instances of type 1, when the allowed imbalance is very low, the scheduling algorithm has to do adjustments in the profiles of maximum load, $N_i^{max}(t)$, in more than one line, while for the problems of type 2 the adjustments are almost restricted to the line with the largest number of vehicles.

## Conclusions and future work

We have seen that scheduling the charging of electric vehicles may be formulated as a Dynamic Constraint Satisfaction Problem (DCSP) with Optimization. In this paper, we have given a formal definition for one problem of this family. This problem is termed PI and it is motivated by a real environment in which a number of vehicles may require charge from an electric system installed in a garage where each vehicle has a preassigned space. This problem is hard to solve due to the imbalance constraints among the three lines of the three-phase electric feeder. We have proposed an algorithm that reduces the calculation of a solution for the dynamic scheduling problem to solving a number of instances of the one machine sequencing problem with variable capacity, denoted $(1, Cap(t)||\sum T_i)$. As far as we know, the $(1, Cap(t)||\sum T_i)$ problem was not yet considered in the literature.

The overall charge scheduling algorithm was evaluated by simulation over a benchmark set inspired in some real scenarios. The results of this study shown that the proposed algorithm is better than some dispatching rules such as First Come First Scheduled, which could be followed by a human operator, and a more sophisticated scheduling algorithm that uses the Least Staring Time rule. In our opinion, the performance of our algorithm relies on how it deals with the imbalance constraints. Instead of keeping this constraint after each operation is scheduled, as it is done by the other two algorithms, we define profiles of maximum load in the three lines and then adapt the schedules to these profiles. Even though these profiles may require a number of adjustments, the algorithm produces much better schedules than the other two algorithms.

This work leaves some issues open for future research. Firstly, we will make a more comprehensive experimental study considering instances derived from different expected scenarios to that considered here. For instance weekend scenarios or situations derived from some contingencies. In this study we will consider some variants of the algorithm that solves the $(1, Cap(t)||\sum T_i)$ problem. As we have mentioned, the apparently tardiness rule may be used probabilistically to obtain a variety of solutions.

Then, we will have to consider a number of characteristics of the real situations that have been skipped here. For example, the users may pick up the vehicle before the declared due date $d_{ij}$, or the battery may get fully charged before the expected charging time $p_{ij}$. In both cases an imbalance may be produced in the system. To deal with these and other situations, we will have to add new asynchronous events to the model.

Another important characteristic that must be considered is the fact that the charging time of the vehicles may be reduced in situations of saturation in order to reduce the tardiness of the vehicles. Furthermore, if the tardiness for some vehicle is too large in situations of very high demand, the vehicle may be discarded from the schedule and so not served.

Another line for future research will be devoted to generalize the problem formulation and the solution methods to situations where, for example, the contracted power changes over the time or the vehicles can be charged at non constant rate. As it is pointed in (Sedano et al. 2012), the later is technically possible under certain restrictions and offers much more flexibility to organize the charging of vehicles over time. At the same time, it may make the scheduling problem harder to solve.

## Acknowledgments

This research has been supported by the Spanish Government under research project TIN-20976-C02-02. We would like to thank the anonymous referees for their useful comments and suggestions.

## References

Bent, R., and Van Hentenryck, P. 2004. Regrets only! online stochastic optimization under time constraints. In *Proceedings of the 19th national conference on Artifical intelligence*, AAAI'04, 501–506. AAAI Press.

Chang, H. S.; Givan, R.; and Chong, E. K. P. 2000. Online scheduling via sampling. In *In Artificial Intelligence Planning and Scheduling (AIPS)*, 62–71.

Dechter, R., and Dechter, A. 1988. Belief maintenance in dynamic constraint networks. In *Proceedings of the Seventh Annual Conference of the American Association of Artificial Intelligence*, 37–42.

Graham, R.; Lawler, E.; Lenstra, J.; and Kan, A. 1979. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics* 5:287 – 326.

Kaplan, S., and Rabadi, G. 2012. Exact and heuristic algorithms for the aerial refueling parallel machine scheduling problem with due date-to-deadline window and ready times. *Computers & Industrial Engineering* 62(1):276–285.

Koulamas, C. 1994. The total tardiness problem: Review and extensions. *Operations Research* 42:1025–1041.

Ma, Y.; Chu, C.; and Zuo, C. 2010. A survey of scheduling with deterministic machine availability constraints. *Computers & Industrial Engineering* 58(2):199–211.

Rangsaritratsameea, R.; Ferrell, W. G.; and Kurzb, M. B. 2004. Dynamic rescheduling that simultaneously considers efficiency and stability. *Computers & Industrial Engineering* 46:1–15.

Sang-Oh Shim, S.-O., and Kim, Y.-D. 2007. Scheduling on parallel identical machines to minimize total tardiness. *European Journal of Operational Research* 177(1):135–146.

Schmidt, G. 2000. Scheduling with limited machine availability. *European Journal of Operational Research* 121:1–15.

Sedano, J.; Portal, M.; Hernández-Arauzo, A.; Villar, J. R.; Puente, J.; and Varela, R. 2012. Sistema de control autónomo para distribución de energía en una estación de carga de vehículos eléctricos: diseño y operación. *Technical Report. Instituto Tecnológico de Castilla y León ITCL.*

Verfaillie, G., and Jussien, N. 2005. Constraint solving in uncertain and dynamic environments: A survey. *Constraints* 10(3):253–281.

Wu, D.; Aliprantis, D.; and Ying, L. 2012. Load scheduling and dispatch for aggregators of plug-in electric vehicles. *IEEE Transactions on Smart Grid* 3(1):368–376.

# Integrating Planning, Execution and Diagnosis to Enable Autonomous Mission Operations

**Jeremy Frank and Gordon Aaseng and K. Michael Dalal and Charles Fry and Charles Lee and Rob McCann and Sriram Narasimhan and Lilijana Spirkovska and Keith Swanson**
NASA Ames Research Center
Mail Stop N269-1
Moffett Field, CA 94035-1000
{firstname.lastname}@nasa.gov

**Lui Wang and Arthur Molin and Larry Garner**
NASA Johnson Space Center
Mail Code ER61
2010 NASA Parkway
Houston, TX 77085
{firstname.lastname}@nasa.gov

**Abstract**

NASA's Advanced Exploration Systems Autonomous Mission Operations (AMO) project conducted an empirical investigation of the impact of time delay on today's mission operations, and of the effect of processes and mission support tools designed to mitigate time-delay related impacts. Mission operation scenarios were designed for NASA's Deep Space Habitat (DSH), an analog spacecraft habitat, covering a range of activities including nominal objectives, DSH system failures, and crew medical emergencies. The scenarios were simulated at time delay values representative of Lunar (1.2-5 sec), Near Earth Object (NEO) (50 sec) and Mars (300 sec) missions. Each combination of operational scenario and time delay was tested in a Baseline configuration, designed to reflect present-day operations of the International Space Station, and a Mitigation configuration in which a variety of software tools, information displays, and crew-ground communications protocols were employed to assist both crews and Flight Control Team (FCT) members with the long-delay conditions. This paper describes the mitigation configuration, with specific attention on the plan and procedure execution tracking and fault detection, isolation and recovery software.

## Introduction

NASA is now investigating a range of future human spaceflight missions that includes a variety of Martian destinations and a range of Near Earth Object (NEO) targets. These possibilities are summarized in Table 1. Table 1 shows the approximate distance between the destination and the Earth, where the control center will be located, and the one-way light-time delay between the destination and Earth.

| Destination | Earth Distance (km) | 1-way Time delay (s) |
|---|---|---|
| Lunar | 38,400,000 | 1.3 |
| NEOs (close) | 100Ks | 10s |
| Mars (close) | 545,000,000 | 181.6 |
| Mars (opposition) | 4,013,000,000 | 1337.6 |

**Table 1. One-way light-time delay (seconds) to candidate destinations.**

On next-generation deep-space missions, crews will have to operate much more autonomously than they do today. A higher degree of crew autonomy represents a fundamental change to mission operations. Enabling this new operations philosophy requires a host of protocol and technology development to answer the following question: How should mission operations responsibilities be allocated between ground and the spacecraft in the presence of significant light-time delay between the spacecraft and the Earth?

### Human Spaceflight Mission Operations Today

Current International Space Station (ISS) operations are conducted with significant reliance on ground monitoring, control, and planning capability; some of which is by design to maximize crew time available for onboard science. Nearly instantaneous feedback from ground commands combined with a computer architecture designed with more software control capability than previous vehicles provides Flight Control Team (FCT) personnel the ability to conduct critical mission operations while minimizing, or in some cases eliminating, the need for onboard crew intervention.

Nearly continuous communication coverage is maintained with ISS for voice, telemetry, commanding, and video transfer with the various control centers during crew wake periods. Procedures are designed for Crew, Ground, or Multi-Center execution. Crew procedures depend on existing spacecraft displays for commanding references and data telemetry checks. Ground procedures may rely on additional displays, as well as references to command instances that are not readily available on the spacecraft. There is no existing data path that can join telemetry and commanding with the procedure viewer. Further, there is no existing indication of the current step in progress transferred from crew to ground. Voice call or telemetry indications showing that equipment was affected as intended are used to view progress through a procedure by another user. Execution of a procedure by a ground Flight Controller requires approval from the Flight Director. Upon proceeding into the execution steps, the Flight Controller enables command uplink capability and executes the commands called out in the procedure steps.

Off-nominal events, such as system failures, may create a need to deviate from the original mission plan. Such deviations typically have downstream impacts to plans later in the week or even further in the future. The rest of the FCT works closely in these cases with the Ops Planner to coordinate plan impacts and reschedule events to later opportunities, while still meeting mission objectives and priorities wherever possible. Off-nominal events may also change the environment around the ISS by changing the orientation or configuration of the vehicle. These unplanned and unanalyzed changes are corrected as soon as possible, and post-event analysis is conducted to determine if damage was done to the ISS structure. Future operations may be subjected to additional constraints should analysis indicate that increased protection is necessary. In depth troubleshooting and analysis efforts are a coordinated effort between the FCT and mission analysists in the post-event timeframe.

### The Challenge of Distant Destinations

For the last 50 years, NASA's crewed missions have been confined to the Earth-Moon system, where speed-of-light communications delays between crew and ground are practically nonexistent. The close proximity of the crew to the Earth has enabled NASA to operate human space missions primarily from the ground. This "ground-centered" mode of operations has had several advantages: by having a large team of the people involved on the ground, the on-board crew could be smaller, the vehicles could be simpler and lighter, and the mission performed for a lower cost.

The roles and responsibilities of the crews of the future will differ fundamentally from those of the past. Crewmembers will be the primary "doers" for more and more activities, responsible for performing most of the procedures associated with their assigned activities, and completing troubleshooting procedures in response to system failures and medical emergencies. While FCT members are expected to play an active role in some of these procedures as well, overall their role will be more supportive, advising and guiding crewmembers as they went about their activities.

Accompanying this change in role and responsibility is a necessary change in the tools used by crews to manage the mission. With fewer crewmembers onboard spacecraft comes the need to redesign tools used for the FCT, who may have more training and more time to understand the systems. As responsibility for executing the mission shifts to the crew, the technology used to support them must evolve to suit the available time and resources, both computational and cognitive, that spacecraft and crews have to manage the tasks.

## The Autonomous Mission Operations Experiment

NASA conducted an experiment assessing crew-ground interaction and operational performance was performed in May and June of 2012 in NASA Johnson Space Center's Deep-Space Habitat (DSH) (Kennedy, 2010; Tri, et al.,2011) an Earth-analog of a workspace and living area that might house a crew during the transport and surface phases of a deep-space crewed mission. Crews consisting of a commander and three flight engineers followed a two-hour mission timeline populated with activities representative of those that might occur during a typical day in the quiescent (cruise) phase of a long-duration space mission. Crews were supported by a small Flight Control Team (FCT) consisting of eight console positions located in the Operations Technology Facility (OTF) in the Christopher Kraft Mission Control Center at Johnson Space Center. The two-hour mission timeline was performed repeatedly under varying conditions:

- A simulated time delay between the ground and the vehicle of low (1.2 or 5 seconds), medium (50 seconds), or long (300 seconds) duration.
- Either no unexpected events (nominal), multiple spacecraft systems failures (off-nominal systems), or a crew medical emergency (off-nominal medical).
- One of two mission operations configurations. In the Baseline configuration, conducted first, the flight control team and crew performed their nominal and off-nominal tasks with support tools, interfaces, and communications protocols similar to those in use for International Space Station operations today. In the Mitigation configuration, crews and FCT members had access to an advanced suite of operations support tools and mission support technologies that we hypothesized would enable the crew to carry out nominal and off-nominal mission operations with greater autonomy and with enhanced crew-ground coordination capability under time delay.

The AMO study complements and extends previous studies (Bleacher et al. 2011; Chappelle et al. 2011; Chappelle et al. 2012; Hurst et al. 2011; Kanas et al. 2010; Kanas et al. 2011) of time delay in ground-based analog environments in a variety of ways. The AMO study is the first of the studies in NASA's Earth-analog environments to examine the effects of time delay in an operational environment that:

- Exclusively utilized highly experienced NASA flight controllers and astronauts as study participants.
- Achieved at least a medium level of mission operational fidelity (as rated by the participants).
- Exclusively employed operations products (plans and procedures) like those used in crewed missions today.

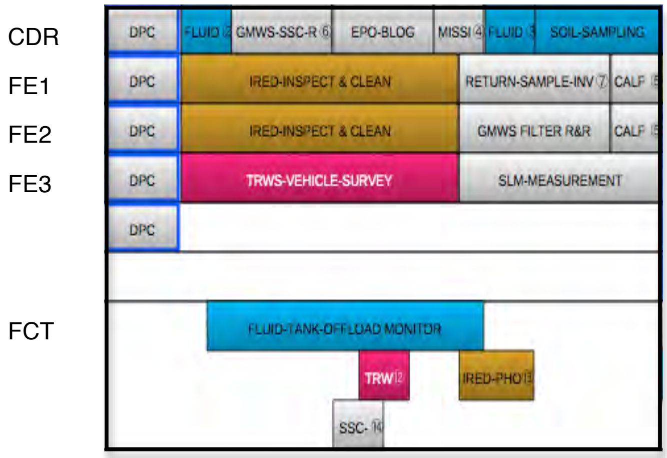


Figure 1. Mission Timeline.

## Mission Timeline

The experiment employed variations of a timeline of activities that the crew needed to complete. For the simulation "initial conditions", the vehicle was returning form an asteroid and was in a "quiescent" operational mode, meaning there are no significant, complex or dynamic operations scheduled (i.e. no burns or other maneuvers were planned for the day). The vehicle was in a nominal configuration except for some designated conditions listed below, and there were no previous major systems failures. This timeline was built by hand prior to the experiments and was unchanged during the experiments (even in response to system failures).

The crew's timeline consisted of 12 activities of varying duration during a two-hour period, and is shown in Figure 1. In the Baseline configuration, these activities were preceded by a 10 minute schedule-prepwork activity and a 15 minute Daily Planning Conference (DPC) activity, in which the flight control team briefed the crew on the specifics of the day's timeline. A total of 31 procedures accompanied these activities. These procedures included

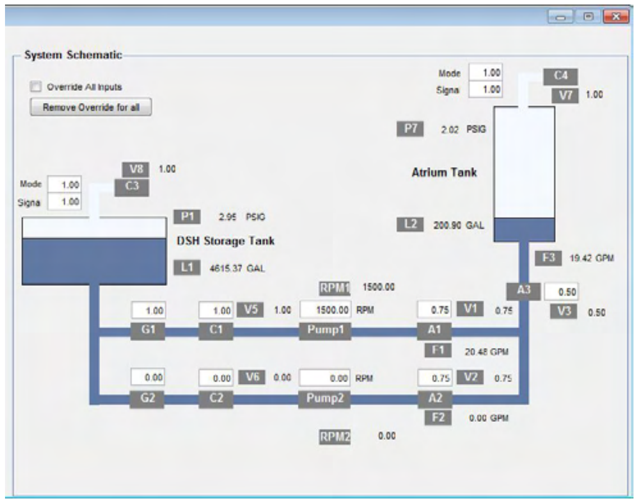


Figure 2. (Simulated) Fluid Transfer System.

both nominal and off-nominal procedures for operation of spacecraft subsystems and crew activities. The activities and simulated failures were designed so that coordination was needed between the FCT and the crew, thereby magnifying the impact of time delay.

The focus of this paper is on the technology used by flight controllers and crew to manage the Atrium Tank Fluid Fill activity, and in handling failures in the spacecraft Electrical Power System (EPS). These subsystems had rich electronic interfaces and were the most amenable to fault injection, and were thus best suited to applying procedure execution and fault management technology; they are described in the next section.

## Fluid Transfer Activity

The Atrium Tank Fluid Fill activity employed a software simulation of a spacecraft water tank and valve system; the schematic of the system is shown in Figure 2. It consists of a DSH storage tank on the left and Atrium tank on the right. The fluid transfer activity involves transferring fluid from the storage tank to the atrium tank. This is achieved by using redundant transfer lines through a combination of valves and pumps. The G valves represent gate valves that can only be opened or closed manually and can only be controlled to be fully open or fully closed. The C valves represent control valves that can be commanded remotely and can also only be fully open or fully closed. The A valves represent annin valves that can be remotely controlled to any partially open status between 25% and 100%. The pumps can be operated at different RPMs ranging from 0 to 3000. For a nominal fluid transfer operation the main transfer line on the top will be used while the auxiliary transfer line in the bottom is only used in case of contingencies. This activity was planned to take roughly an hour and a half in total.

The Fluids system had associated thresholds, which if exceeded, would produce Caution and Warning messages:

| C&W | Threshold |
|---|---|
| FLOW_HIGH | > 26 GPM |
| FLOW_LOW | < 10 GPM |
| FLOW_CHECK | < 12 GPM or > 24GPM |
| TANK_FULL | >= 100% |
| TANK_HIGH | > 93% |
| TANK_LOW | < 10% |
| TANK_EMPTY | < 3% |

Only the flows at the outlet of annin valves and tank levels are measured and simulation is configured to publish only these values to the DSH communication infrastructure. These subsets of sensor locations were chosen to increase the diagnosis ambiguity, which was driven by the experiment design to increase ground/crew interaction. The simulation includes the capability to inject faults. The faults considered were valves stuck in fixed positions, pumps failed or operating at lower efficiency, and sensor faults. Only one fault was introduced in the system at any point in time.

A total of 8 procedures were developed, including both nominal and off-nominal procedures.

## Electrical Power System and Wireless Sensors

The DSH EPS system consists of an interconnection of 120Vac, 28Vdc and 24Vdc power sources. These power sources are distributed throughout the inside of the DSH through six Power Distribution Units (PDUs), each of which has 16 outlets. These can be remotely commanded on and off.

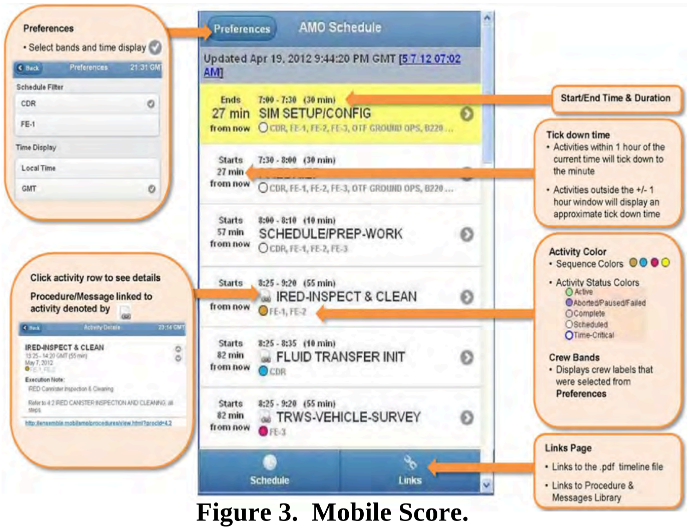
DSH data (temperature and humidity) was collected through a network of Wireless Sensor Nodes (WSNs); these sensors were powered via the DSH power system. They reported data via a Compact Remote I/O (cRIO) card, also powered by the DSH power system.

Failure injection included the ability to fail the 24V converter or part of the cRIO. These failures would also eliminate data delivery via the WSNs, leading to a typical ‘C&W storm’ for both loss of sensor data as well as loss of power on the various power channels, requiring diagnosis. Individual WSNs also proved unreliable and caused unplanned failures. In the event the cRIO needed to be rebooted, this would take between 15 and 25 minutes, during which no WSN data is available.

A total of 7 EPS procedures were developed, all of which were off-nominal procedures.

## Technology Enabling Crew Autonomy

The AMO experiment included a wide range of technologies enabling autonomy; see (Frank et al. 2013) for a more complete discussion. In this paper we focus attention on three key technologies that aided the FCT and crew in executing the plan: Mobile Score, WebPD, and Advanced Caution and Warning (ACAWS). These tools are described in the next section.


Preferences
• Select bands and time display
Back
Preferences
21:31 GMT
Schedule Filter
CDR
FE-1
Time Display
Local Time
GMT
Preferences
AMO Schedule
Start/End Time & Duration
Tick down time
• Activities within 1 hour of the current time will tick down to the minute
• Activities outside the +/- 1 hour window will display an approximate tick down time
Activity Color
• Sequence Colors
• Activity Status Colors
Active
Aborted/Paused/Failed
Complete
Scheduled
Time-Critical
Crew Bands
• Displays crew labels that were selected from Preferences
Click activity row to see details
Procedure/Message linked to activity denoted by
Back
Activity Details
23:14 GMT
Links Page
• Links to the .pdf timeline file
• Links to Procedure & Messages Library
Schedule
Links

Updated Apr 19, 2012 9:44:20 PM GMT [5 7 12 07:02 AM]

Ends 27 min from now 7:00 - 7:30 (30 min) SIM SETUP/CONFIG CDR, FE-1, FE-2, FE-3, OTF GROUND OPS, B220 ...

Starts 27 min from now 7:30 - 8:00 (30 min) CDR, FE-1, FE-2, FE-3, OTF GROUND OPS, B220 ...

Starts 57 min from now 8:00 - 8:10 (10 min) SCHEDULE/PREP-WORK CDR, FE-1, FE-2, FE-3

Starts from now 8:25 - 9:20 (55 min) IRED-INSPECT & CLEAN FE-1, FE-2

Starts 82 min from now 8:25 - 8:35 (10 min) FLUID TRANSFER INIT CDR

Starts 82 min from now 8:25 - 9:20 (55 min) TRWS-VEHICLE-SURVEY FE-3

IRED-INSPECT & CLEAN
13:25 - 14:20 GMT (55 min)
May 7, 2012
FE-1, FE-2
Execution Note:
IRED Canister Inspection & Cleaning
Refer to 4.2 IRED CANISTER INSPECTION AND CLEANING, all steps
http://ensemble.mobi/amo/procedures/view.html?procid=4.2

Figure 3. Mobile Score.

## Mobile Score

Mobile Score is a browser and server based application to provide lightweight display of timeline information, and to provide easy access to procedures and other experiment documentation; an overview of Mobile Score is provided in Figure 3. The FCT and crew used Mobile Score to display the plan, filter plan activities based on assigned crew performing the activity or activity time, show which activities were slated to occur soon, and quickly access procedure references, messages, and other information needed to perform activities. The Mobile Score UI was accessible via modern versions of web browsers like Firefox, or Google Chrome on desktop machines, and using Mobile Safari on the Apple iPads used by the AMO crew members in the DSH.

As mentioned, previously, the timeline was not altered during the experiment; no activities were reordered, added or removed. This was driven by the shortness and simplicity of our experiment timeline, and typical practice for ISS operations today is to limit plan updates to once per day.

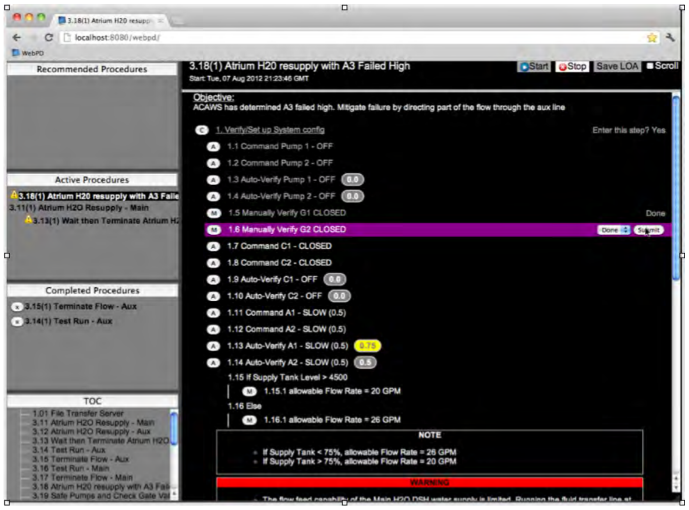
Easy access to procedures, and the AMO Message Library, was available via Mobile Score by selecting Links in the lower right corner. Mobile Score was used by the crew members while they were performing procedures that were being viewed on one of the four crew iPads. During Baseline experiments, crew members would use Mobile Score to navigate to PDF versions of their procedures. During Mitigation, they would use Mobile Score to provide convenient access to the desired procedures within WebPD (see the next section). After selecting a procedure from the index, the crew member could select either the PDF or WebPD version of the procedure. Note also that two separate procedure table-of-content (TOC) lists were available – one accessible from Mobile Score, and another available directly within WebPD. This is because the WebPD TOC also contained engineering procedures that were only intended for the DSH engineering team.

## WebPD

The procedures for operating spacecraft systems and performing tasks were presented using an electronic interface called WebPD, shown in Figure 4. These resources were accessible to all team members from their browser, and from the DSH iPads. WebPD incorporated a focus bar, allowing the crew to track their place in a procedure. The crew could issue commands to spacecraft systems from WebPD. Procedure instructions that verify telemetry readings display the current reading along with an indication of whether or not it is in range of the desired value(s). Procedure steps often required reading system data values or checking limits; WebPD receives system data, and these are incorporated in the WebPD interface.

The WebPD allows many users to monitor the execution of all procedures simultaneously. However, only one client, the one that started the procedure, has control of the procedure's execution (e.g. takes input from the user); the others simply track execution and do not allow interaction.

The WebPD presents a list of all available procedures, any of which can be selected for execution at any time, by any user. When a procedure from the list is selected, it is displayed, and can be started with the mouse-click (or finger touch on the iPad) of a button. The WebPD also maintains lists of procedures that are active, completed, and those that have been recommended by the AMO



3.18(1) Atrium H20 resupply with A3 Faile
3.11(1) Atrium H2O Resupply - Main
3.13(1) Wait then Terminate Atrium H2

3.15(1) Terminate Flow - Aux
3.14(1) Test Run - Aux

1.01 File Transfer Server
3.11 Atrium H2O Resupply - Main
3.12 Atrium H2O Resupply - Aux
3.13 Wait then Terminate Atrium H2O
3.14 Test Run - Aux
3.15 Terminate Flow - Aux
3.16 Test Run - Main
3.17 Terminate Flow - Main
3.18 Atrium H20 resupply with A3 Fai
3.19 Safe Pumps and Check Gate Val

3.18(1) Atrium H20 resupply with A3 Failed High
Start: Tue, 07 Aug 2012 21:23:46 GMT

Objective:
ACAWS has determined A3 failed high. Mitigate failure by directing part of the flow through the aux line

```
C 1. Verify/Set up System config                              Enter this step? Yes
  A 1.1 Command Pump 1 - OFF
  A 1.2 Command Pump 2 - OFF
  A 1.3 Auto-Verify Pump 1 - OFF 0.0
  A 1.4 Auto-Verify Pump 2 - OFF 0.0
  M 1.5 Manually Verify G1 CLOSED                             Done
  M 1.6 Manually Verify G2 CLOSED                             Done Submit
  A 1.7 Command C1 - CLOSED
  A 1.8 Command C2 - CLOSED
  A 1.9 Auto-Verify C1 - OFF 0.0
  A 1.10 Auto-Verify C2 - OFF 0.0
  A 1.11 Command A1 - SLOW (0.5)
  A 1.12 Command A2 - SLOW (0.5)
  A 1.13 Auto-Verify A1 - SLOW (0.5) 0.75
  A 1.14 Auto-Verify A2 - SLOW (0.5) 0.5
    1.15 If Supply Tank Level > 4500
    | M 1.15.1 allowable Flow Rate = 20 GPM
    1.16 Else
    | M 1.16.1 allowable Flow Rate = 26 GPM
```

NOTE
- If Supply Tank < 75%, allowable Flow Rate = 26 GPM
- If Supply Tank > 75%, allowable Flow Rate = 20 GPM

WARNING

**Figure 4. WebPD.**

diagnostic tools (described further in a later section). Any number of procedures can be running concurrently and monitored by the WebPD. However, only one procedure can be viewed a time; a single click switches the view to the desired procedure.

WebPD procedures are stored in Procedure Representation Language (PRL), a derivative of XML (Kortenkamp et al. 2008) and developed in a graphical environment called the Procedure Integrated Development Environment (PRIDE) (Izygon et al. 2008). PRL and a predecessor of WebPD have been used in previous simulations of mission operations environments. PRIDE is a graphical tool that allows easy drag-and-drop construction of procedures, in a fashion that only permits procedures with valid structure and content. In particular, the most system-specific procedure content – telemetry and commands – are provided in a system menu and do not need be looked up manually in documents, as was the prior approach. In addition, PRIDE provides a host of GUI features that make procedure authoring convenient. Procedures in PRL can be automatically translated to the Plan Execution Interchange Language (PLEXIL), which allows instruction-by-instruction automated execution of procedures according to the operator's wishes (Frank 2010).

## Advanced Caution and Warning

The Advanced Caution & Warning System (ACAWS) for both the fluid transfer simulation and EPS system consists of three main components. A diagnostics engine is responsible for diagnosis of any faults. This includes detection of off nominal behavior, isolating the cause for the off-nominal behavior, and determining the magnitude of the deviation from nominal behavior. A diagnosis to recommended procedure mapper is responsible for recommending disambiguation and/or mitigation procedures to be executed based on the current diagnosis provided. Finally, the ACAWS GUI is responsible for presenting the results from diagnosis and the procedure mapper to the user. Procedure recommendations are also displayed by WebPD. In the following sections we describe the diagnosis engine used to handle failures in that subsystem.

### TEAMS

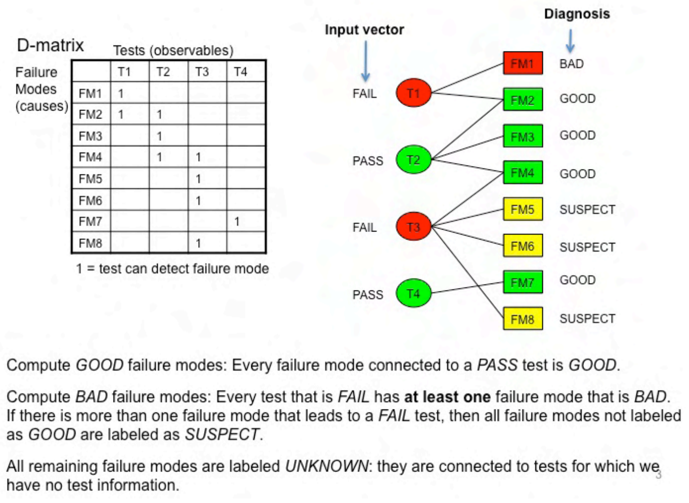
The diagnostics engine for EPS failure utilizes the Qualtech Inc. Testability Engineering and Maintenance System (TEAMS) tool (Mathur et al. 1998). TEAMS determines the root cause (failed components and their failure modes, the "bad" components in the TEAMS vernacular). When the sensor signature is ambiguous, TEAMS provides a list of possibly failed components (the "suspect" set). A companion tool, TEAMATE, provides the operator recommendations on additional observations to perform the most effectively reduce the ambiguity.

TEAMS is a model-based system. The model captures a system's structure, interconnections, tests, procedures, and failures. This dependency model captures the relationships between various system failure modes and system instrumentation.

For real-time diagnosis, a dependency matrix (D-matrix) is generated from the model. The D-matrix is a two-dimensional matrix of failure modes and effects ("tests"; things that can be observed). The values are binary with 1 meaning a test can detect a failure mode and 0 meaning that a test cannot detect that failure mode.

Input to TEAMS is a vector of binary health status tests as computed by the DSH software and supplemented by the ACAWS-EPS system. DSH software provides observations on whether certain telemetry parameters are valid and whether they are in bounds. The EPS system input used validity bits for a parameter rather than its actual value, since that provides the information necessary to determine whether an EPS component is being powered. ACAWS-EPS supplements these observations with heartbeat data providing observations on when the last time a component was heard from.

A simple example of how TEAMS uses the D-matrix and tests vector is shown in Figure 5. The D-matrix is shown on the left. The same matrix can be represented by the graph on the right. The input vector is the observed state of the tests. The right column shows the output from TEAMS – a diagnosis that explains the input vector given the D-matrix as generated by the model (not shown). In this case, given the two failed tests and two passed tests, TEAMS has determined that failure-mode-1 is definitely failed ("bad"), failure-modes 2, 3, 4, and 7 are all healthy ("good"), and failure-modes 5, 6, and 8 can each explain failed test T3, hence those three failure modes are placed into an ambiguity group of "suspects." In cases where the input vector leads to an ambiguity group, TEAMMATE recommends a procedure that can help disambiguate the


D-matrix

Failure Modes (causes) / Tests (observables)

| | T1 | T2 | T3 | T4 |
|---|---|---|---|---|
| FM1 | 1 | | | |
| FM2 | 1 | 1 | | |
| FM3 | | 1 | | |
| FM4 | | 1 | 1 | |
| FM5 | | | 1 | |
| FM6 | | | 1 | |
| FM7 | | | | 1 |
| FM8 | | | 1 | |

1 = test can detect failure mode

Compute *GOOD* failure modes: Every failure mode connected to a *PASS* test is *GOOD*.

Compute *BAD* failure modes: Every test that is *FAIL* has **at least one** failure mode that is *BAD*. If there is more than one failure mode that leads to a *FAIL* test, then all failure modes not labeled as *GOOD* are labeled as *SUSPECT*.

All remaining failure modes are labeled *UNKNOWN*: they are connected to tests for which we have no test information.

**Figure 5. TEAMS algorithm.**

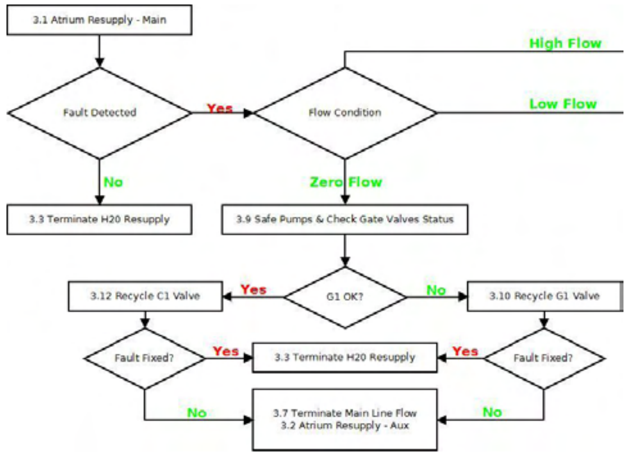

3.1 Atrium Resupply - Main
Fault Detected
Yes
Flow Condition
High Flow
Low Flow
No
Zero Flow
3.3 Terminate H20 Resupply
3.9 Safe Pumps & Check Gate Valves Status
G1 OK?
Yes
3.12 Recycle C1 Valve
No
3.10 Recycle G1 Valve
Fault Fixed?
Yes
3.3 Terminate H20 Resupply
Yes
Fault Fixed?
No
3.7 Terminate Main Line Flow
3.2 Atrium Resupply - Aux
No

**Figure 6. Subset of Fluid ACAWS model Diagnosis to Procedure Mapper.**

suspects. For the DSH system, this was exclusively a request for crew observation of data not available via telemetry, such as the status of an indicator light, the operation of an overhead light, etc. In the D-matrix above, these “manual tests” would be additional columns of the matrix, with mapping from those tests back to the failure modes they can observe or detect.

**HyDE**

HyDE (Narasimhan and Brownston, 2007) was used for fault detection and isolation of the fluids system. A model in HyDE is a hybrid, consisting of a finite set of states and transitions between those states (a discrete model), as well as sets equations over real-valued quantities that either hold within a state, or can trigger transitions between states (a continuous model). The models describe the behavior of the system under nominal and faulty conditions. HyDE uses commands sent to actual system to drive these models to predict the behavior of the system as it evolves over time. These predictions are checked for consistency with the observations available from the sensors. Any inconsistencies indicate presence of faults in the system. These inconsistencies, if any, are then used in a search to identify cause for the inconsistencies. This is achieved back propagating through the model to identify components in the model contributing to the inconsistency.

For the fluid transfer system HyDE was used to serve two purposes. First a hybrid quantitative model was used as an observer to track the behavior of the system. This observer used the same commands that were being sent to the simulation through the communication interface to predict the expected values for the flows and tank levels. These predictions were compared against sensor observations (available through the communications infrastructure) to generate qualitative symbols indicating low, high and no flow.

These qualitative symbols are then fed into the qualitative part of the HyDE model which then determines the state of the components and sensors. Once an initial diagnosis has been established HyDE uses a fault disambiguation and mitigation tree to recommend procedures to isolate the fault and mitigate the effects of fault so that the fluid transfer activity can be completed as planned. This tree is generated manually based on the set of faults and ambiguity groups that would be generated by HyDE; a portion of which is illustrated in Figure 6.

Figure 7 shows the architecture of the ACAWS-Fluids system as built for the Atrium fluid transfer system incorporating HyDE. The ACAWS-EPS architecture differs from this in only minor ways (test result input and invocation of TEAMMATE).

## Technology Integration

Figure 8 shows how all of these components were integrated for use by the crew and the FCT for the AMO Mitigation Configuration. A crewperson examining the timeline in Mobile Score can automatically invoke WebPD, which would display the procedure corresponding to the activity. The procedure (as written with PRIDE) has all necessary commands and telemetry elements embedded in it; using WebPD, the crew can send commands, check relevant telemetry values, step through the procedure and track the current instruction. Using shared situational awareness between crew and ground, the FCT could monitor procedure progress without the need to bother the crew. In the event of faults, ACAWS would send procedure recommendation messages to the WebPD, prompting the crew to perform a procedure. In cases where a further piece of information was needed (e.g. the crew had to examine a system and manually enter data) the

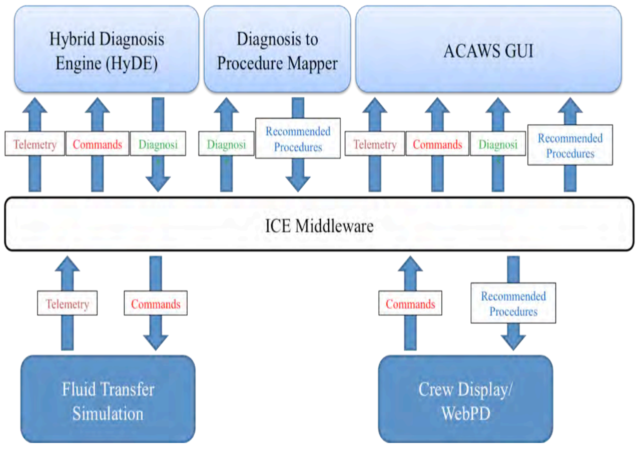


**Figure 7. Fluids System ACAWS architecture.**

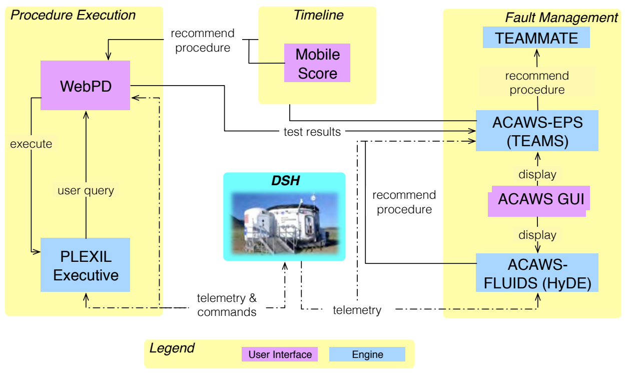


**Figure 8. Mitigation Configuration Software Integration Architecture.**

procedure recommendation function was performed by TEAMMATE; in cases where a unique fault diagnosis required a recovery action, this was accomplished by the more generic diagnosis to procedure mapper.

**Shared Situational Awareness**

WebPD status was shared over the air-ground link, so that the flight control team could see what procedures were executing, and what procedure step the crewperson running a procedure was presently executing. This information was rendered on the same WebPD UI the crew used, albeit after a delay. This is accomplished via a publish-subscribe paradigm, in which the WebPD software on one end of the time delay publishes any change of status (e.g. the execution of a procedure step), which is then received by the WebPD and causes the update of the receiver side. ACAWS was run both onboard and on ground; the same data was ingested and used to perform diagnosis, thereby also providing shared situational awareness.

## Failure Scenarios

In this section we describe the specific scenarios during which the plan execution and fault management technology were used during the AMO experiments.

### Fluid Transfer System Failures

The fluid transfer is the first activity on the timeline. The crew browses the plan with Mobile Score, and can either bring up the procedure as a PDF file (in Baseline) or navigate to the WebPD (in Mitigation) to initiate the activity.

The activity is initiated through a procedure which first verifies that all components are closed/off and then sets all the annin valves to desired values (based on level in the storage tank) and then opens the main transfer line by opening valves G1 and C1 and setting Pump1 speed to 1500 rpm. Initially while the flow stabilizes a low flow C&W is received, but ignored as per recommendation of the procedure.

After the flow has stabilized, an A1 Valve stuck at 25% fault is injected. This results in a low flow C&W message, which directs the crew to check consistency between the flow sensors. At this point all components in the main transfer line (G1, C1, Pump1, A1, and A3) are suspected to be faulty. The first step of the troubleshooting focuses on the G1 valve. The crew is asked to manually inspect the Gate Valve and report the status. When the G1 status indicates that it is open, the next step is to cycle the C1 valve (in case this gets the C Valve unstuck). When that does not resolve the problem, a test run using the auxiliary line is proposed. The main line components are closed or turned off and the auxiliary line components are opened or turned on. After the flow has stabilized the flow values are checked and indicate that the problem has been resolved. The crew records that one of C1, Pump1 or A1 is faulty and continues the fluid transfer activity using the auxiliary line. Recall that in Baseline, all of this activity is managed by the crew reading the PDF version of the procedure, and using other tools to monitor the status of the fluid system, or command, as needed.

By contrast, in the Mitigation configuration, HyDE is able to use the quantitative and dynamic information from the changes in the flow to determine that Pump1, A1 or A3 is causing the low flow. In addition HyDE is also able to provide estimates for the fault magnitude. The crew can see the fault candidates on the Fluids ACAWS system animated schematic of the Fluid system. Based on this diagnosis HyDE recommends a procedure to perform Test Run using Aux procedure. This recommendation is received by WebPD. All commands and data are fully integrated, so the crew can execute this procedure from WebPD, without referring to other tools. When this procedure is executed all the flows get back to normal and so HyDE does not recommend any more procedures. Steps involving troubleshooting G1 and C1 valves can be completely skipped because of the additional information available. This configuration was not completed in time

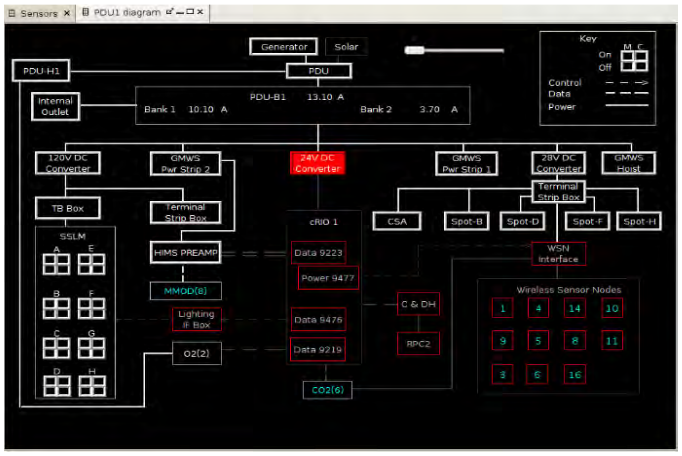


**Figure 9. ACAWS EPS UI.**

for the AMO experiments, was implemented later (along with fully automated procedure execution) and is described further in (Narasimhan et al. 2012).

### EPS System Failures

The EPS system and WSNs are organized in such a way that, initially, a 24Vdc converter failure and a cRIO card failure exhibit the same symptoms, namely, loss of data from all of the WSNs. This is distinguished from individual WSN failures. The 24Vdc converter has an

LED that the crew can inspect manually, which disambiguates a 24Vdc failure from a cRIO card failure.

In the Baseline configuration, the crew's indication of a problem is loss of data from some or all of the WSNs and the accompanying 'C&W' storm. The 28Vdc failure has a similar flavor to the 24Vdc failure. The crew had procedures available to handle this problem, but it was not actually introduced in simulation. The crew was trained on the EPS procedures, and would then have to call up the relevant EPS procedures and determine whether the problem was one or more WSN failures, a failure of the 24Vdc, the 28Vdc, or the cRIO card.

By contrast, in the Mitigation configuration, the crew had access via the ACAWS-EPS system to an animated schematic of the EPS system that was informed by the TEAMS model. This UI rendered the diagnoses produced by TEAMS. In addition, the procedure mapper would send recommended procedures to WebPD, which reduced the amount of time needed for the crew to bring up relevant procedures to run. As the crew executed steps and provided the answers to the diagnostics questions, ACAWS would automatically refine its diagnosis, render this on the UI, and then produce new procedure recommendations, which would appear in WebPD. Finally, with commands and data integrated into the WebPD, the crew could issue commands and read telemetry directly in WebPD instead of turning to other software tools to command the EPS system. The ACAWS-EPS UI is shown in Figure 9.

## Measuring the Impact

The AMO experiment incorporated both qualitative and quantitative measurements to assess the impact of time delay and the impact of the Baseline vs Mitigation configuration on operator performance. Since this paper is focused on a subset of all of the protocols and technologies used in the Mitigation configuration we focus attention on a subset of the performance impact story; a more complete description of the experiment and the results can be found in (Frank et al. 2013).

### Quantitative Impact

Quantitative performance measures for the experiment included task completion rates, crew and flight controller workload, and crew-flight controller coordination difficulty; discussions of how these metrics are collected can be found in (McCandless et al. 2005, McCann et al. 2006). Task completion rates, surprisingly, did not vary significantly between Baseline and Mitigation configuration. However, FCT and crew measurements of workload and coordination difficulty were positively impacted:

- Workload and coordination difficulty *decreased* at every time delay as a result of the Mitigation configuration.
- Communications acts *decreased* in the Mitigation configuration.

It is notable that ACAWS also rendered some procedure steps and some procedures performed in the Baseline tests unnecessary. For example, because of HyDE's capability to maintain the current state of the system and the ability to fuse multiple sensors, steps associated with checking for consistency among sensors can be eliminated. This also enabled revision of the set of procedures to a simpler and more concise set. Similarly, TEAMS replaced procedure steps to both determine the likely EPS failure causes, and the TEAMMATE component automatically recommended both fault isolation and recovery procedures.

### Qualitative Impact

Along with the quantitative measurements described above, experiment participants provided subjective comments on their experiences with the technology.

ACAWS technology provided two different forms of automated assistance with FDIR activities: Automated fault diagnosis, and automated recommendation of fault isolation or recovery procedures. Comments indicate both workload reduction and a reduction in the need for coordination followed from these capabilities:

*"ACAWS provided useful direction for the crew, so there was little need for us to do anything other than concur"*

*"ACAWS told me which procedure to work which the ground later confirmed but I had already completed the procedure."*

The last quote speaks to both the situation awareness and autonomy issues, and also notes the benefits of greater autonomy for mitigating the effects of time delay:

*"The time delay had little impact because ACAWS ran most of the procedure. Since the ground and crew can follow ACAWS, it was pretty seamless. MCC and DSH were able to come to common agreement with ACAWS. MCC and DSH statused each other via voice calls and texting."*

The following are two highly representative comments about the benefits of WebPD from FCT members:

*“WebPD made it very easy to follow along in the procedures even with the time delay”*

*“Very easy to see where the crew should go from the line they were on as well as where they were going”.*

*“The ability to track procedures and where the crew was in each step was awesome”.*

Not only did WebPD help the ground keep track of where the crew with within a procedure, but several mentions were made of the usefulness of the windows that showed what procedures were currently active, and which procedures had been completed:

*“[I liked] [Ability to] see when crew brings up and starts a procedure, can see when they are done with a procedure.”*

## Conclusions and Future Work

Human spaceflight missions to distant destinations impose significant added burdens on the FCT and the crew. The AMO experiment quantified these burdens, and showed that a tight integration of plan execution tracking (Timeline and procedures) and ACAWS provided both qualitative and quantitative benefits to both the FCT and crew during quiescent mission phases.

Extending these benefits to more systems, increasing automation, and conducting experiments in higher fidelity settings are the subject of future work.

## Acknowledgements

The AMO experiment could not have been conducted without the dedication of dozens of people at NASA ARC, JSC and KSC. The AMO project was funded by the NASA Advanced Exploration Systems (AES) Program.

## References

J.E. Bleacher, J.M. Hurtado, Jr., K.E. Young J. Rice, W.B. Garry, and D. Eppler. Desert Rats 2010 Operations Tests: Insights from the Geology Crew Members. Proceedings of the 42d Lunar and Planetary Sciences Conference, 2011.

Chappelle, S. Andrew F. Abercromby, Ph.D., Michael L. Gernhardt, Ph.D. NEEMO 15: Evaluation of Human Exploration Systems for Near Earth Asteroids. Proceedings of the Global Space Exploration Conference, 2012.

Steven P. Chappell, Andrew F. Abercromby, William L. Todd, Michael L. Gernhardt. Final Report of NEEMO 14: Evaluation of a Space Exploration Vehicle, Cargo Lander, and Crew Lander during Simulated Partial-gravity Exploration and Construction Task. NASA Technical Report 2011-216152, 2011.

Frank, J. When Plans are Executed by Mice and Men. Proceedings of the IEEE Aerospace conference, Big Sky, MT. 2010.

Izygon, M., Kortenkamp, D., Molin, A., A Procedure Integrated Development Environment for Future Spacecraft and Habitats. Space Technology and Applications International Forum, Albuquerque, NM, 2008.

V. W. Hurst IV, S. Peterson, M.D. K. Garcia, A. Sargsyan, D. Ebert, D. Ham, B.S. D. Amponsah. S. Dulchavsky, Smart Ultrasound Remote Guidance Experiment (SURGE) –Ultrasound Image Collection during Lunar and Near Earth Orbit Space Missions. 82nd Annual Scientific Meeting of the Aerospace Medical Association; 8-12 Mayu 2011; Anchorage, AK; United States

Kanas, N., Saylor, S., Harris, M., Neylan, T., Boyd, J., Weiss, D., Baskin, P., Cook, C., Marmar, C. High vs. Low Crewmember Autonomy in Space Simulation Environments. Acta Astronautica, v. 67, no. 7-8, 2010 p. 731 - 738

Kanas, N., Harris, M., Neylan, T., Boyd, J., Weiss, D., Cook, C., Saylor, S. High vs. Low Crewmember Autonomy during a 105-day Mars Simulation Mission. Acta Astronautica, v. 69, no. 7-8, 2011 p. 240 – 244

Kennedy, Kriss J. NASA Habitat Demonstration Unit Project – Deep Space Habitat Overview. 41st International Conference on Environmental Systems (ICES), Portland, Oregon, USA, 17-21 July 2011.

Kortenkamp, D., Verma, V., Dalal, K.M., Bonasso, R.P., Schreckenghost, D., Wang, L., A Procedure Representation Language for Human Space Flight Operations. 9th International Symposium on Artificial Intelligence, Robotics, and Automation for Space, Los Angeles, CA, 2008.

A. Mathur, S. Deb, and K. Pattipati. Modeling and Real-Time Diagnostics in TEAMS-RT. Proc. American Control Conf., IEEE Press, 1998, pp. 1610–1614.

McCandless, J. W., McCann, R. S., Berumen, K. W., Gauvain, S. S., Palmer, V. J., Stahl, W. D., & Hamilton, A. S (2005). Evaluation of the Space Shuttle Cockpit Avionics Upgrade (CAU) Displays. In Proceedings of the 49th Annual Meeting of the Human Factors and Ergonomics Society.

McCann, R. S., Beutter, B. R., Matessa, M., McCandless, J. W., Spirkovska, L., Liston, D., Hayashi, M., Huember, V., Lachter, J., Ravinder, U., Elkins, S., Renema, F., Lawrence, R., & Hamilton, A. (2006). Evaluation of an Onboard Real-Time Fault Management Support System for Next-Generation Space Vehicles. Report to the Human Research Program

Tri, T., Kennedy, K., Toups, L., Gill, T., Howe, A. S. Planning and Logistics for the In-Field Demonstration of NASA’s Habitat Demonstration Unit (HDU) Pressurized Excursion Module (PEM) at Desert Rats 2010. Proceedings of the International Conference on Environmental Systems, Portland OR. 2011.

Frank, J., Spirkovska, L., McCann, R., Pohlkamp, K., Wang, L., Morin, L. Autonomous Mission Operations. Proceedings of the IEEE Aerospace Conference. 2013.

Narasimhan, S., Gonzalez, R., Spirkovska, L., McCann, R., Frank, J., Lee, C. Advanced Caution & Warning System for Simulated Fluid Transfer Subsystem of a Deep Space Habitat. Proceedings of the 23d International Workshop on the Principles of Diagnosis. 2012.

Narasimhan S and Brownston L (2007), "*HyDE – A General Framework for Stochastic and Hybrid Model-based Diagnosis*", In 18th International Workshop on Principles of Diagnosis (DX 07). June, 2007, pp. 162-169.

# Scheduling a Multi-Cable Electric Vehicle Charging Facility

**Tony T. Tran[1], Mustafa K. Dogru[2], Ulas Ozen[2] and J. Christopher Beck[1]**

[1]Department of Mechanical and Industrial Engineering
University of Toronto
{tran,jcb}@mie.utoronto.ca

[2]Alcatel-Lucent Bell Labs
{Mustafa.Dogru,Ulas.Ozen}@alcatel-lucent.com

**Abstract**

We consider scheduling electric vehicles in a charging facility where customers arrive dynamically and tend to park longer than their charge time. In this setting, it is reasonable and technologically feasible to have charging docks with multiple cables, although such docks do not currently exist in practice. Assuming such a dock design, we study three information conditions: we know the number of electric vehicles at each dock, we know stochastic information about arrival and charging requirements, and we are able to observe exact charging requirements for vehicles in the system. We formulate a continuous-time Markov decision process (CTMDP) to optimize the system performance under the first two conditions and demonstrate that it does not scale to realistic-size problems with multiple docks. However, a single-dock version of the CTMDP is tractable. We propose and numerically evaluate a number of admission and scheduling schemes building on both the single-dock CTMDP and approaches from the scheduling literature under each of the three information conditions. Our results demonstrate (i) the value of a multi-cable dock, (ii) the importance of obtaining actual charging requirement information, and (iii) the integral role of admission and scheduling policies based on available information to improve performance.

## 1 Introduction

Advances in battery, electric engine and charging technologies have resulted in significant improvements in the performance of electric vehicles (EVs) in terms of range, charging time, etc. Although reduced emissions and lower fuel and maintenance costs over their lifetime favor EV adoption over internal combustion engine vehicles (CVs), range anxiety prevents more people from owning an EV. Range anxiety is the fear of being stranded because an EV has insufficient capacity to reach a destination (Tate, Harpster, and Savagian 2008). Unlike CVs which run on gasoline, an EV requires an electrical power source to recharge its battery and completely recharging a fully depleted battery can take from half an hour to almost a full day. For example, a Nissan Leaf with a 24 kWh capacity battery has a range of about 73 miles on a single charge and requires 16-18 hours to fully charge from a depleted battery under level I AC (120V) chargers. Level II AC (240V) chargers decrease the required charge time to 7 hours and level III DC (500+V) chargers further reduces the time to approximately half an hour.[1]

To address range anxiety, charging stations are being placed in convenient locations including highway rest stops and gas stations. It is also becoming popular to place charging stations in parking lots. Cars generally spend a large amount of time in parking lots, whether it is a shopping mall, an airport, or work place. These charging stations provide a convenient way to charge a battery by integrating the charging into time periods which drivers are naturally occupied.

Current charging docks have a single cable and can be connected to one EV. In a gas station or rest stop, one would expect customers to leave when charged and so a new car can be connected immediately. However, in a parking lot, a vehicle may be connected to a cable well after charging has been completed. A charging dock which incorporates multiple cables will allow as many connected cars as there are cables, even if only one car can be charged at a time. With a multi-cable dock, a car may complete its charge and stay connected, while another EV immediately begins charging. Such a dock is an economical way to improve effective charging capacity without purchasing more docks. For example, the annual cost (purchase + maintenance) of a level II AC charger ranges between \$900 and \$5000 USD over a 10-year life cycle, and only about 20% of the total cost is due to initial capital investment (Botsford 2012). Although the cost of adding a cable to a dock is not negligible, we expect much lower maintenance cost for the multi-cable dock design in comparison to an equivalent system with multiple single-cable docks. Hence, the multi-cable dock design decreases the total cost of a charging facility due to fewer docks needed to purchase and maintain, and lower initial installation cost.

In this paper, assuming multi-cable docks, we study the admission and scheduling problem associated with management of an EV charging facility. EVs arrive dynamically over time and can be plugged into an available cable to be charged. Admission and scheduling decisions must be made immediately upon arrival of an EV and the system manager aims to minimize the costs associated with rejecting and delaying customers. Given the relatively new application of

[1]http://www.nissan.ca/vehicles/ms/leaf/en.

EV charging facilities and EVs themselves, available functionality varies. In particular, the information available to a system manager from both his/her docks and the customers' EVs will vary. We therefore propose three information availability characteristics: (i) the number of EVs at each dock is known, (ii) stochastic information is available about EV arrival and charging rates, and (iii) exact charging requirements for all EVs in and arriving to the system are known. We create and study policies for admission and scheduling in each information environment and compare the performance of a multi-cable charging dock facility.

Our study demonstrates:

- the value of a multi-cable dock for parking lot facilities,
- the importance of obtaining actual charging requirement information from EVs, and
- the increase in system performance arising from intelligent admission and scheduling policies.

In the following section, the charging facility we study is described in detail. Section 3 presents a continuous-time Markov decision process (CTMDP) for the system. However, the model suffers from the curse of dimensionality and hence does not scale well to real life problems. Thus, heuristic methods for admission and scheduling decisions are introduced in Section 4. Experimental results are presented in Section 5, followed by a discussion in Section 6. Some related work on EV charging can be found in Section 7 and Section 8 concludes the paper.

## 2 System Model

We consider an EV charging facility with $N \in \mathbb{N}$ docks, each with $K \in \mathbb{N}$ cables. A cable connects a dock to a car and enables charging. However, being connected does not mean that the car is able to immediately start charging. Each dock is limited to charging a single car at a time.

The parking lot system studied assumes cars arrive dynamically following a Poisson process with rate $\lambda$. The amount of charging time each EV requires is exponentially distributed with mean $\mu^{-1}$. In order for the vehicle to leave the system, two conditions must be met: 1) the required charge is completed and 2) the deadline specified by the driver is reached. We assume the deadline is exactly $L$ time units after the arrival of the EV and represents the time at which the customer has agreed to return to remove the EV. This is a simplification of the real system which one can think of as having customers with different deadlines. However, our assumption represents a parking lot that sells an exact amount of parking time to all customers, but each customer will have different charging times. If a vehicle is charged before the deadline, it must wait until the deadline before it can exit because, typically, the driver will not return for the EV before the deadline. On the contrary, if charge completion occurs after the deadline, the EV is delayed and must wait until the charge completes before exiting the system.[2]

[2]An alternative system could have EVs leaving at the time of the deadline regardless of charge. The models presented in this paper can just as easily handle these systems with minor alterations.

We assume three information conditions for our system. The first is referred to as the *cardinality* condition: it is known how many EVs are at each dock and of those, whether or not a vehicle is delayed or charged. Under the second, *stochastic*, condition, the arrival ($\lambda$) and charging ($\mu$) rates and their distributions are available. It corresponds to assumptions common to stochastic modelling (Puterman 1994) and queueing theory (Gross and Harris 1998). Finally, we wish to consider information natural to the scheduling community (Pinedo 2008) which tends to include deterministic information about job durations and, often, arrival times. While deterministic arrival times are unrealistic in our application, it is reasonable to assume that charging time information is known upon an EV arrival. For example, the charging time can be found from either requesting the customer give the charge level they wish to purchase or by having a wireless transmitter from the vehicle broadcast this information.[3] Therefore, our third condition, which we term *observable*, assumes that the actual remaining charging times of every EV in the system at each time point can be observed. For an arriving EV $j$, the charging time $p_j$ is available upon arrival.

The system manager makes two decisions. The first is whether to accept or reject an incoming vehicle. If rejected, then there is a finite cost $c_r \geq 0$ for losing a customer. The second decision is how to schedule an accepted EV. Scheduling comprises of the decision of assigning a dock for an EV and determining the order that EVs are charged. If accepted, an EV is immediately assigned to an available cable and cannot be switched. When the owner returns to pick up his/her EV, if charging is not yet complete, the delay is penalized. If $T_j$ is the tardiness of a late EV $j$, that is, the time between the EVs deadline and when its charge is completed, then the delay cost is $c_d T_j$ where $c_d$ is finite and non-negative.

The system manager wants to find an admission and scheduling policy to minimize the overall system cost. However, the control a system manager has over a parking lot may vary. One can see in most common parking lots, customers arrive and choose a spot themselves. Here, a system manager would have no direct control over customers. Thus, an indirect method to control the system is by limiting the available spots (docks and cables). Although we do not explore the capacity planning problem, we will observe some of the effects of adding cables and docks to the system in our experiments. A system with moderate admission control could be seen as having a gate at the entrance of the parking facility to turn customers away. Once admitted, the customer is free to choose whichever spot they please. Finally, we can imagine a facility where customers must purchase a spot first and will then be assigned to a specific location. In this way, complete control over the admission and scheduling of a vehicle is possible upon arrival. Specific admission and scheduling policies will be discussed in Section 4.

[3]Such transmitters are already available (Botsford 2012), but not used widely.

## 3 Continuous-Time Markov Decision Process

We present a CTMDP model to handle the admission and scheduling of a charging facility when only cardinality and stochastic information is available. Our current definition of deadlines being a fixed $L$ time units after arrival does not adhere to the memoryless requirement of a CTMDP. Therefore, we assume that deadlines are not deterministic, but exponentially distributed with mean $L$.[4] We further simplify the CTMDP representation by enforcing first-come, first-served (FCFS) ordering of EVs once assigned to a dock.

The state of the system at time $t$ is represented by, $\mathbf{S}(t) = \{\mathbf{Q}(t), \mathbf{W}(t), \mathbf{D}(t)\}$. Here, $\mathbf{Q}(t)$, $\mathbf{W}(t)$, and $\mathbf{D}(t)$ are vectors of size $N$. $\mathbf{Q}(t)$ indicates the number of EVs in the system at time $t$ that are waiting for a charge and not yet due on each of the $N$ docks. $\mathbf{W}(t)$ represents the number of vehicles that have completed their charge, but are waiting for the deadline and $\mathbf{D}(t)$ is the number of vehicles that are not yet charged but have already reached their deadline, on each of the $N$ docks. We represent the element in each of the vectors using a lowercase letter with index $n$ to indicate the dock (e.g., the $n$th dock is fully described by $q_n(t)$, $w_n(t)$, and $d_n(t)$).

There are $N+1$ possible actions when an EV arrives. An action, $a \in A$, can either assign the EV to one of the $N$ docks or to reject the vehicle. Therefore, $A \in \{0, 1, \dots, N\}$ where 0 represents rejection. An EV cannot be assigned to a dock with no available cables (i.e., if $q_n(t) + w_n(t) + d_n(t) = K$). The cost function, $C(\mathbf{S}(t), a)$ defines the expected cost associated with action $a$ in state $\mathbf{S}(t)$. When a vehicle is rejected, independent of the current state, the cost is $c_r$. If a vehicle is admitted, then we must calculate the expected cost associated with the additional vehicle for each particular state. We denote the time that an EV $j$ completes its charge as $\phi_j$. Thus, the delay cost of a vehicle is $\max\{0, (\phi_j - L)c_d\}$.

We calculate the expected delay of an accepted vehicle by conditioning on the state of the system at time $t$ and the dock $n$ that will be assigned the vehicle. Since there are $q_n(t) + d_n(t)$ vehicles not yet charged on dock $n$, admission of a new vehicle requires a total of $B = q_n(t) + d_n(t) + 1$ exponentially distributed charges until the arriving vehicle has completed its charge. $B$ is the number of EVs present in the system that requires a charge plus the new arriving job. Thus, the expected delay given a state $i$ and assignment to dock $n$ is,

$$\mathrm{E}[\text{delay}|\mathbf{S}(t), a = n] = \int_L^\infty (x - L) f(x; B, \mu) \mathrm{d}x,$$

where $f(x; B, \mu)$ is the density function of the Erlang distribution. This yields

$$\mathrm{E}[\text{delay}|\mathbf{S}(t), a = n] = \frac{\left[\mu^{-1}\Gamma(B+1, \mu L) - L\Gamma(B, \mu L)\right]}{\Gamma(B)}.$$

[4]We found numerically through simulation that a system with deterministic deadlines does not behave differently from the calculated CTMDP with exponentially distributed deadlines under FCFS. Due to space restrictions, we do not present these details.

Here, $\Gamma(b)$ is the gamma function and $\Gamma(b, \mu L)$ is the upper incomplete gamma function. Therefore, for any action $a$, we know the expected delay, which we multiply by $c_d$ to obtain the expected delay cost.

The transition rates depend on the current state of the system, $\{\mathbf{Q}(t), \mathbf{W}(t), \mathbf{D}(t)\}$. Transitions occur because of three types of events: EV arrival, charge completion, and meeting a deadline. Actions only affect transition rates for the arrival events; the other events are independent of the actions taken.

We define an $N$-sized vector $\mathbf{e}_n$ that has 1 as the $n$th element and the rest 0. A deadline can occur on any dock which has $q_n(t) + w_n(t) > 0$. If $w_n(t) > 0$, then a transition occurs with rate $\frac{q_n(t)+w_n(t)}{L}$ and will change the state to $\{\mathbf{Q}(t), \mathbf{W}(t) - \mathbf{e}_n, \mathbf{D}(t)\}$. If $w_n(t) = 0$, a transition occurs with rate $\frac{q_n(t)}{L}$ to state $\{\mathbf{Q}(t) - \mathbf{e}_n, \mathbf{W}(t), \mathbf{D}(t) + \mathbf{e}_n\}$. Charge completions can occur on any dock which has $q_n(t) + d_n(t) > 0$. If $q_n(t) + d_n(t) > 0$, a transition occurs with rate $\mu$ to $\{\mathbf{Q}(t), \mathbf{W}(t), \mathbf{D}(t) - \mathbf{e}_n\}$ if $d_n(t) > 0$, and $\{\mathbf{Q}(t) - \mathbf{e}_n, \mathbf{W}(t) + \mathbf{e}_n, \mathbf{D}(t)\}$ otherwise. For an arrival event, we must consider the action taken. If an EV is rejected, then there is no transition. If we decide to assign an arriving vehicle to the $n$th dock, then there is a transition rate of $\lambda$ to $\{\mathbf{Q}(t) + \mathbf{e}_n, \mathbf{W}(t), \mathbf{D}(t)\}$. Since we consider exponentially distributed inter-arrival times, charging times, and deadlines, the system is memoryless and we can restrict the decision epochs to only the times when the state changes.

A policy, $\pi$, specifies the action, $a^\pi(\mathbf{S})$, for each state $\mathbf{S} = \{\mathbf{Q}, \mathbf{W}, \mathbf{D}\}$. We can use uniformization (Lippman 1975) to discretize the MDP and solve for an optimal policy using policy iteration (Howard 1960) since we have a finite state space with bounded costs (Puterman 1994).

The CTMDP suffers from the curse of dimensionality: solving the CTMDP for real life problems is intractable as the number of states grows exponentially. The number of states for any particular system is $(K+1)^{2N}$. With five cables and five docks, we see that there are more than 60 million states. Thus, such a model is intractable for parking facilities of even moderate capacity. Nevertheless, this model can guide us to heuristics that use stochastic information which we present in the following section.

## 4 Admission and Scheduling

We propose admission and scheduling policies to manage the charging facility for each of our information availabilities. Depending on the particular conditions of information availability, some policies may not be possible to perform.

### 4.1 Admission Policies

The admission policy decides whether to accept or reject an EV upon arrival. The policies consider each dock and decides which docks are able to be assigned the EV. We present three policies which represent systems that, respectively, use cardinality, stochastic, and observable information:

- Free Cable - A vehicle is admitted if there are available cables - i.e., if $\exists n \; : q_n(t) + w_n(t) + d_n(t) < K$. Any dock with an available cable may be assigned the EV.

- CTMDP1 - Consider a single-dock version of the model. Solve for the optimal single-dock policy using CTMDP of Section 3 with the same parameters of the original multi-dock model ($\mu$, $L$, $c_d$, $c_r$) except an arrival rate of $\frac{\lambda}{N}$. Given the state of the system $\mathbf{S}$, for every dock $n = 1, 2, \ldots, N$, check whether an arriving EV would be admitted in state ($q_n(t)$,$w_n(t)$,$d_n(t)$) under the optimal single-dock policy. The docks that accept an arriving EV are the only ones that can be assigned the EV.
- Myopic - Using the charging times, calculate the delay cost of scheduling an EV on each dock. If the cost of accepting the EV on the dock is less than the cost of rejecting the EV, then accept and assign to one of these docks.

Although the admission policy will limit how one can assign an EV, it does not assign a dock.

### 4.2 Scheduling Policies

Once an EV is admitted, a policy is used to assign a dock. Again, each policy represents systems that, respectively, use cardinality, stochastic, and observable information.

- Random - Randomly choose among one of the possible docks determined by the admission policy.
- CTMDP2 - Similar to CTMDP1, restrict the CTMDP model to a single dock and solve the Bellman equations to find the expected cost of being in each state. From the set of possible docks as defined by the admission policy, choose the dock in a state that yields the minimum expected cost.
- Earliest - From the set of possible docks defined by the admission policy, choose the dock that will result in the earliest completion time for the EV if all other already assigned EVs complete charge first.

These policies represent different levels of control from no involvement, where we expect customers to enter and choose a cable randomly, to complete control where a customer is sent to a particular dock in order to maximize performance. Once assigned to a dock, EVs are charged in FCFS order.

A system manager couples an admission policy with a scheduling policy to control the charging facility. It is obvious that the information availability would limit his/her choice of policies above. For example, CTMDP1-Earliest can only be used if cardinality, stochastic, and observable information conditions are met.

## 5 Experimental Results

We simulate the charging facility to observe the effects of using multiple cables and different policy combinations. For each experiment, 10 instances of 100,000 time units are simulated for every admission-scheduling policy pair. In all experiments, customers set a deadline of exactly 1 time unit after their arrival ($L = 1$) and docks have a charging rate of $\mu = 6$. For example, if our time unit is 3 hours, the parameters represents a parking lot which customers park for 3 hours and request on average 30 minutes charging time.

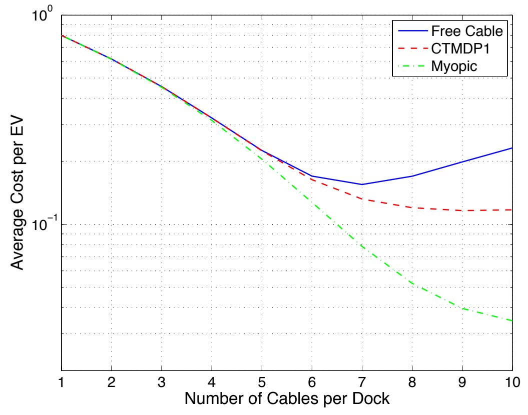


Figure 1: Experiment 1 - Single Dock Charging Facility.

As mentioned earlier, we wish to understand the effects of information availability. Since the underlying system does not change, only the available information, we can observe how having certain information affects performance.

The first experiment is a single dock charging facility. EVs arrive to this system with a rate $\lambda = 4$ and $K$ varies from 1 to 10. The cost of rejecting an EV is $c_r = 1$ and the delay cost is $c_d = 5$. Since there are no scheduling decisions to be made, this system only tests the admission policies, hence, comparing the three information availabilities.

Figure 1 presents simulation results for the single dock system. In general, we see a large decrease in cost with additional cables due to a significant increase in accepted EVs. The cost gap between CTMDP1 and Myopic, especially for $K > 6$, shows the extent of performance improvement that can be achieved by obtaining the actual charging time information.

Interestingly, the average cost per EV of the Free Cable policy increases with eight or more cables. To get a better idea of why the increase in cost occurs, we can think of accepting an EV when there are 7 other EVs waiting for a charge. In this scenario, it is likely that completing eight charges will require more time than the deadline allows because the docks are expected to only charge 6 EVs in 1 time period. Therefore, rejecting is generally a better choice. The Free Cable policy does not do so and continues to accept EVs when there are free cables.

The second experiment looks at a multi-cable, multiple dock system. In this system, there are ten identical docks with between one and ten cables each. Vehicles arrive at a rate of $\lambda = 50$. The rejection cost is $c_r = 1$ and the delay cost is $c_d = 5$. Results are shown in Figure 2 for each combination of admission and scheduling policy. Note that CTMDP1 does not always perform better than Free Cable. With observable information, using the Earliest policy favours Free Cable. Figure 2 illustrates the importance of information availability. The strong performance of Myopic-Earliest shows the clear advantages of having observable information. Further, obtaining some control of the system is

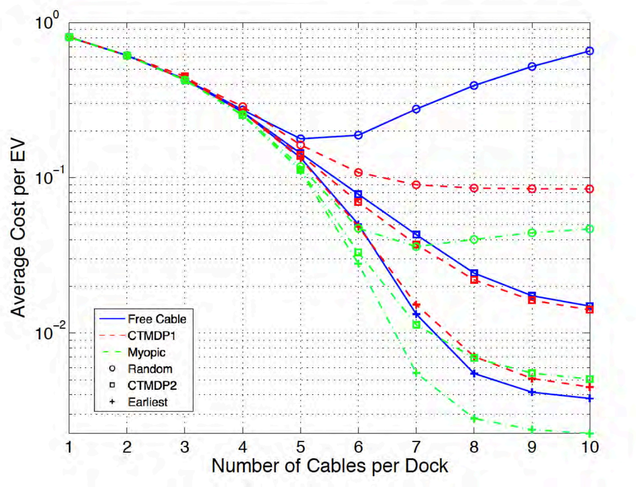


Figure 2: Experiment 2 - Multiple Dock Charging Facility.

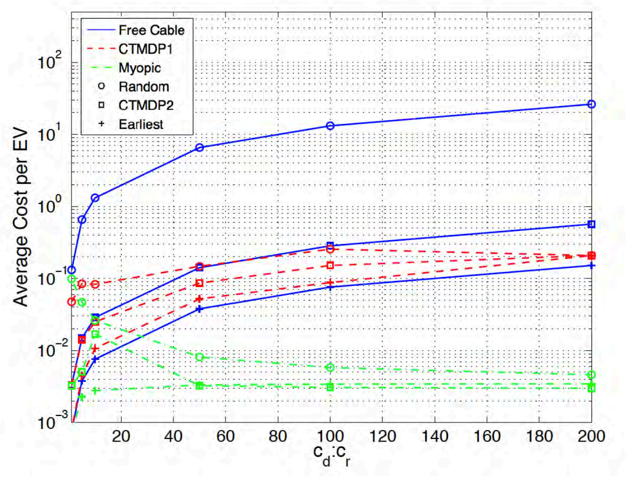


Figure 3: Experiment 3 - Multiple Dock Charging Facility.

quite important as Free Cable-Random is found to perform very poorly once there are seven or more cables. In fact, even Myopic-Random suffers when $K$ increases since there is less control over the scheduling of EVs.

To further study the effects of the system parameters, we experiment with varying the cost structure. Using the parameters from the multiple dock facility of the previous experiment, we fix the number of cables to ten and vary $c_d$ between 1 and 200. Figure 3 shows the results of this experiment.

We see for most policy pairs, increasing $c_d$ leads to increased overall costs per EV. However, this trend is not true when Myopic is paired with Random or CTMDP2. A possible explanation for this anomaly is that increasing $c_d$ will restrict potential candidate docks under the Myopic policy. We believe Myopic can give guidance when delay costs are high by removing the busier docks from consideration.

An interesting observation is that for $c_d = 200$, Myopic-CTMDP2 out-performs Myopic-Earliest. A scheduling policy using only stochastic information out-performs the policy that exploits exact information with Myopic admission. In the previous experiments, we have seen a large dominance when using observable information over stochastic but clearly this is not always the case. We return to this observation below.

The last experiment studies a charging facility with ten docks and ten cables each. Costs are as in the first two experiment: $c_r = 1$ and $c_d = 5$. We vary the arrival rate $\lambda = \{50, 55, 60\}$ to observe how the policies behave under varying system loads. Figure 4 graphs the results. We see a larger increase in costs for the Free Cable based policy pairs as load increases when contrasted with CTMDP1 and Myopic. Of interest in particular is the performance of Free Cable in comparison to CTMDP1 when using the Earliest scheduling policy. As before, we see at $\lambda = 50$ that Free Cable is better. However, as $\lambda$ increases, CTMDP1 becomes better.

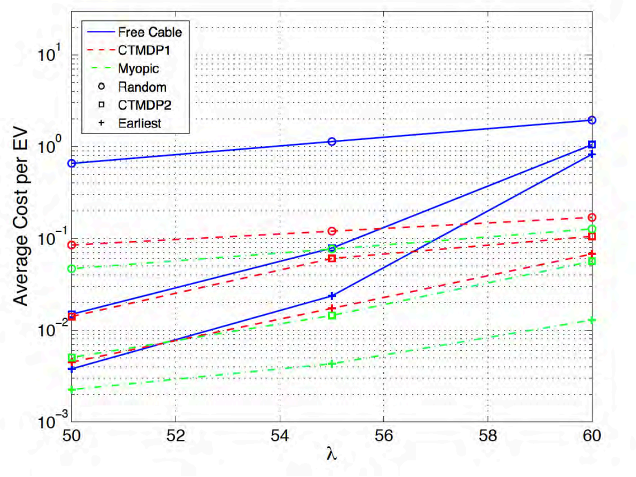


Figure 4: Experiment 4 - Multiple Dock Charging Facility.

## 6 Discussion and Future Work

The results from simulating the charging facility provide insights into the facility designs as well as the directions for building stronger system management models. For facility design, we see that multi-cable docks provide large performance improvements when there is a disparity between the charging requirements of an EV and the expected deadlines ($p_j < L$). Botsford (2012) discusses such systems and provides two solutions: valet parking and reduced power charging docks to increase charging time. Although these solutions increase dock utilization, they are not always practical as the cost for valet parking or requiring a dock for every customer can be high. If $L$ is much larger than $p_j$, adding cables to a dock will greatly improve utilization.

Comparisons of the different policy combinations gives us insight into how one would manage a charging facility.

The most important questions that must first be addressed is what information is available and how much control exists for the admission and scheduling of a customer. Parking facilities are used in a variety of settings and different parking lots will have different features. As mentioned in Section 2, management may not have substantial control over a shopping mall parking lot. Customers arrive and choose their spots freely as long as there is space. We can see the similarities to Free Cable-Random where there is no control over the customers. Here, the only decision making required is one of capacity planning; how many docks should be purchased and how many cables will these docks have. As we can see from Figures 1 and 2, more cables may lead to a *decrease* in performance, so choosing the right capacity is very important to the overall system costs. If the manager does have some control over the assignment of EVs and stochastic information is available without observable charging times, CTMDP1-CTMDP2 is the best performing policy. Since exact charging times are not known, using the Earliest scheduling rule is not possible and our experiments show, as expected, that Free Cable does not outperform CTMDP1 in these circumstances.

Although the policies using observable information were able to achieve the best overall performance, our results suggest the potential for using stochastic information. With increased delay costs, the best performing policy combination was Myopic-CTMDP2. This combination makes use of cardinality, stochastic, and observable information. We believe that to achieve the best performance, policies designed to use the stochastic and observable information is required. Myopic-CTMDP2 only achieves the lowest cost in one scenario, but a more sophisticated scheduler that actively uses all the available information has potential to perform favourably on most, if not all, cases.

Such hybrid reasoning in optimization has previously been proposed in the literature. Recent work by Terekhov et al. (2012) and Tran et al. (2013) looks at combining queueing theory and scheduling models to incorporate stochastic reasoning into combinatorial optimization. In Tran et al.'s (2013) work, a queueing model, using stochastic information, was shown to out-perform a number of scheduling models that made use of observable information. The seeming inconsistency with our result is interesting but is likely due to the very different underlying systems and solution approaches. However, the investigation of when combining stochastic and observable information benefits performance is a promising area of future work and our results from testing a system with high delay costs is an example of how such a combination can be advantageous.

Another direction for future work is online stochastic combinatorial optimization (OSCO) (Van Hentenryck and Bent 2006). OSCO creates schedules by generating and optimizing over samples of future arrivals derived from stochastic information. We see it as a promising direction, especially for more complicated scheduling problems.

We would like to expand this work by further examining the CTMDP model and building more sophisticated scheduling models that make use of stochastic reasoning. We believe that a deeper understanding of the characteristics of the optimal CTMDP policy will help provide necessary components one can utilize when creating a sophisticated scheduler that considers both the dynamics of the system and the combinatorial complexities. As well, we would like to explore possible methodologies of solving the CTMDP. Factored representations of an MDP, which uses dynamic Bayesian networks to represent the stochastic decisions of an MDP, are an interesting possibility for being able to solve the CTMDP (Boutilier, Dearden, and Goldszmidt 2000).

## 7 Related Work on EV Charging

While there are a few studies on EV charging, we are not aware of any work that investigates our parking lot charging scenario with multi-cable docks and dynamically arriving customers. Furthermore, to the best of our knowledge, our paper is the first to study the performance of a multi-cable dock design where the cables in a charging dock are modelled as a limited resource. In all other works, it is either assumed that docks always switch to other cars immediately or the system has unlimited single-cable docks.

Raghavan and Khaligh (2012) examine the effects of EV charging in a smart grid environment. They emphasize the differences between charging methods and time-of-day (evening or night). Li et al. (2011) use dynamic programming to minimize charging costs when electricity prices vary over time. Sioshansi (2012) develops two mixed integer programming models to minimize costs. These works consider the larger power grid problem where people are charging at home rather than in a shared charging facility.

Lee et al. (2011) focus on the problem of a charging station system where there are multiple charging docks and vehicles have different charge lengths, arrival times, and deadlines. In their work, they assume that complete information is known a priori, including the number of EVs. Each vehicle has a different power consumption profile and the objective is to reduce peak power usage over all time periods.

Work on waiting time performance of charging EVs is due to Qin and Zhang (2011). A network of roads is created where nodes represent rest stops to recharge EVs. Drivers are assumed to stop at nodes to charge when required and immediately leave once they are charged. A performance bound is derived and a distributed scheme is proposed which is shown empirically to perform closely to the bound results.

## 8 Conclusion

We studied scheduling electric vehicles in a charging facility where customers arrive dynamically and tend to park longer than their charge time. Our study considered three information conditions: cardinality, stochastic, and observable. We formulated a CTMDP to optimize the system performance under the first two conditions and demonstrate that it does not scale to realistic-size problems with multiple docks. However, a single-dock version of the CTMDP is tractable. We proposed and numerically evaluated a number of admission and scheduling schemes building on both the single-dock CTMDP and approaches from the scheduling literature under each of the three information conditions. We found that the information available significantly alters the overall performance of the system by limiting the admission and scheduling policies that can be implemented. Thus, it is crucial for any system manager to properly understand the information limitations of his/her system and choose the appropriate methodology to optimize performance.

## References

Botsford, C. 2012. The economics of non-residential level 2 EVSE charging infrastructure. In *Proceedings of the 2012 EVS26 International Battery, Hybrid and Fuel Cell Electric Vehicle Symposium*, 1–10.

Boutilier, C.; Dearden, R.; and Goldszmidt, M. 2000. Stochastic dynamic programming with factored representations. *Artificial Intelligence* 121(1):49–107.

Gross, D., and Harris, C. 1998. *Fundamentals of Queueing Theory*. John Wiley & Sons, Inc.

Howard, R. 1960. *Dynamic Programming and Markov Processes*. MIT Press.

Lee, J.; Park, G.; Kim, H.; and Jeon, H. 2011. Fast scheduling policy for electric vehicle charging stations in smart transportation. In *Proceedings of the 2011 ACM Symposium on Research in Applied Computation*, 110–112. ACM.

Li, Z.; Khaligh, A.; and Sabbaghi, N. 2011. Minimum charging-cost tracking based optimization algorithm with dynamic programming technique for plug-in hybrid electric vehicles. In *Proceedings of the Vehicle Power and Propulsion Conference (VPPC), 2011 IEEE*, 1 –6.

Lippman, S. 1975. Applying a new device in the optimization of exponential queueing systems. *Operations Research* 23:687–710.

Pinedo, M. 2008. *Scheduling: theory, algorithms, and systems*. Springer Verlag.

Puterman, M. L. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc.

Qin, H., and Zhang, W. 2011. Charging scheduling with minimal waiting in a network of electric vehicles and charging stations. In *Proceedings of the Eighth ACM international workshop on Vehicular inter-networking*, 51–60. ACM.

Raghavan, S. S., and Khaligh, A. 2012. Impact of plug-in hybrid electric vehicle charging on a distribution network in a smart grid environment. *Innovative Smart Grid Technologies, IEEE PES* 1–7.

Sioshansi, R. 2012. Modeling the impacts of electricity tariffs on plug-in hybrid electric vehicle charging, costs, and emissions. *Operations Research* 60(2):1–11.

Tate, E.; Harpster, M.; and Savagian, P. 2008. *The Electrification of the Automobile: From Conventional Hybrid, to Plug-in Hybrids, to Extended-range Electric Vehicles*. SAE International.

Terekhov, D.; Tran, T. T.; Down, D. G.; and Beck, J. C. 2012. Long-run stability in dynamic scheduling. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS 2012)*, 261–269.

Tran, T. T.; Terekhov, D.; Down, D. G.; and Beck, J. C. 2013. Hybrid queueing theory and scheduling models for dynamic environments with sequence-dependent setup times. In *Proceedings of the 23nd International Conference on Automated Planning and Scheduling (ICAPS 2013)*, To Appear.

Van Hentenryck, P., and Bent, R. 2006. *Online Stochastic Combinatorial Optimization*. MIT Press.

# A Steady-State Model for Automated Sequence Generation in a Robotic Assembly System

**Kimitomo Ochi**[1], **Alex Fukunaga**[2], **Chikako Kondo**[1], **Munehiko Maeda**[1],

**Fumio Hasegawa**[1] and **Yukihiro Kawano**[1]

[1] Corporate Research & Development, IHI Corporation, Japan
[2]Graduate School of Arts and Sciences, The University of Tokyo, Japan
{ kimitomo_ochi, chikako_kondo, munehiko_maeda, fumio_hasegawa, yukihiro_kawano}@ihi.co.jp, fukunaga@idea.c.u-tokyo.ac.jp

**Abstract**

Factory assembly robot systems are becoming increasingly more complex due to shortened product cycles and the need to rapidly manufacture many different types of products. It is necessary to generate efficient sequences which coordinate the actions of multiple, complex devices under a multitude of constraints. We investigate the application of domain-independent planning techniques to the sequence generation problem for a modern, cellular assembly robot system. We evaluate a PDDL model for this domain, and show that this domain poses some challenges for current planners. We propose and evaluate a new, steady-state model which enables high-quality plans to be generated using standard planners.

## Introduction

Industrial robots play a vital role in manufacturing, and are widely used for processing, assembly, carrying/manipulation, welding, paintwork, and inspection.

Due to the shortened life cycles of manufactured products in recent years, there is a need for flexible, robotic manufacturing systems that can be used for a wide range of products. The traditional, large-scale assembly lines designed for mass production (“1 product per belt conveyor”), are being replaced by flexible, cellular manufacturing systems which are staffed by relatively few workers. In cellular manufacturing systems, each “cell” consists of several pieces of manufacturing equipment, and is designed to allow efficient manufacturing of a wide array of low-demand products.

The robotic assembly system described in this paper is one example of a cellular manufacturing system. An assembly robot system automates assembly tasks that are traditionally performed by human workers. In an assembly robot system, a single robot can handle multiple tasks, such as attachment of parts, manipulation/transport, and inspection. By using multiple cooperating robots in a single cell, the system can be used to manufacture a variety of products.

Since a control *sequence* for a robotic assembly system (i.e., a sequence of parallel operations which are performed in an assembly system) needs to manufacture multiple products simultaneously, while minimizing *takt time* (cycle time), generating efficient sequences that make effective use of the increased capabilities of new robotic assembly systems is a difficult task. Sequence generation, which has traditionally been performed by a human worker, has become an increasingly challenging task which requires significantly more skill and time than for the simpler robotic systems that were used in the past.

In order to deal with the difficulties of sequence generation for complex, assembly systems, we are investigating methods for automating the sequence generation process using domain-independent planning technology. The use of domain-independent planning has two motivations: (1) automatic generation of efficient sequences for complex scenarios, with the goal of ultimately generating sequences that are more efficient than sequences developed by humans, and (2) fast, automated generation of sequences in order to enable rapid reconfiguration and deployment of the assembly system in response to demand.

Given a model of the robot’s actions, which encodes domain constraints such as resource and ordering constraints on the actions, a tool that generates sequences which are competitive with current, human-generated sequences is required.

In this paper, we first introduce the assembly robot domain. We describe the overall assembly system, as well as the structure and components of the products (called

"Work" in this domain) which are produced by the assembly system.

We then describe a PDDL model for the assembly robot system. We evaluate this model using a temporal planner which first generates a sequential plan and applies postprocessing to generate a concurrent plan. We show that it is difficult to generate high-quality plans using this straightforward model, and that the plan quality degrades as the amount of products being produced is increased. Then, we propose a technique for modeling our assembly problem as a steady-state process. We show that using this technique, efficient sequences that can be used to generate an arbitrary amount of the same product can be generated using a combination of Fast Downward [Helmert 2006, Fast Downward 2012] and a simple postprocessor for concurrency.

## Assembly Robot System

In this section, we describe the assembly robot system.

### Work and Parts

In this domain, the term "Work" refers to a product that is assembled by the system. The assembly robot system consists of multiple robots that cooperate in order to assemble products (Work). Examples of Work include automobile parts and precision instruments.

A piece of Work consists of a single base and multiple parts that are screwed to the Base. An example of a type of Work that can be assembled by the system is shown in Figure 1. The cylindrical object near the middle of the figure is the base. Part A is the bottom cover, and is screwed to the base. Part B is a piston which is inserted into the base. Part C is the top cover, which is screwed to the base. The upper and bottom cover prevent the piston from protruding from the object.

#### Assembly

Assembly of a piece of Work involves several types of operations, including screwing a part to the base, attaching a part to the base using an adhesive, and welding operations. Assembly is subject to the following kinds of constraints.

#### Ordering Constraints

There are ordering constraints when assembling a piece of Work. For example, in the example shown in Figure 1, if the top/bottom covers are screwed to the base before the piston is inserted into the base, then it will become impossible to insert the piston. These ordering constraints are determined during the design of the Work.

#### Assembly Operation Times

Each operation in the assembly requires a specific amount of time to execute, and the time required for an operation can be context-dependent. For example, the amount of time required to attach a part to a base varies, depending on the part. Assembly operation times are determined when the Work is designed.

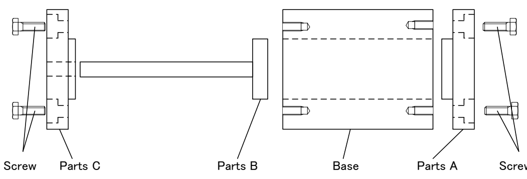
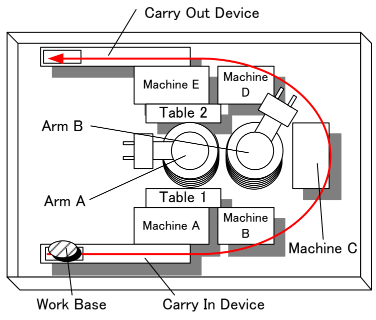
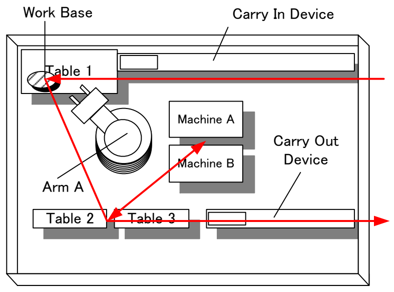


*Figure 1: Example of Work.*



*Model 2-(a)*



*Model 2-(b)*

*Figure 2: Assembly Robot System.*

#### Location Constraints

For each assembly operation, there are constraints on where they can be performed. For example, screw tightening operations must be performed at a dedicated screw tightening device, and insertion of a piston must be performed at a specific type of location called an "Assembly Table". These location constraints are also determined during the Work design process.

#### Constraints on the Work

Summarizing the constraints on the Work shown in Figure 1, the Work must be assembled as follows:

(a). Using an arm, the top cover (Part A) must be attached to the base at Table 1

(b). Part A must be screwed to the base using Machine A
(c). Using an arm, the piston (Part B) must be attached to the base at Table 2.
(d). Using an Arm, the bottom cover (Part C) must be attached to the base at Table 2.
(e). Using Machine B, Part C must be screwed to the base.

It should be clear from the description above that the order in which a single base (Work) flows through the system is fixed. However, the need to coordinate the actions of multiple arms and simultaneously process multiple Works leaves a significant amount of freedom in sequencing.

## Assembly Robot System

As explained in the previous section, assembling a Work requires various operations. In case of the Work shown in Figure 1, operations for attaching the top and bottom covers to the base, screwing the covers to the base, as well as moving the parts/base to appropriate locations so that operations can be performed,are necessary. In addition, because attachment and screwing operations must be performed at different locations, the Work must be moved from the location where the part is attached to the location of the machine which tightens the screw. An assembly robot system integrates multiple robots/machines so that these processes can be performed automatically and efficiently in order to assemble products.

The assembly robot system is comprised of the "assembly table", the base is placed, the "parts tray" where the parts are placed, multiple robot arms, specialized robotic devices/machines (e.g., screw tightening device, transport device).

In this section, we describe each of these components. Figure 2 shows a model of an example system as a whole.

(a). Robot Arm: Robot arms are the centerpieces of the assembly system. Each arm has sensors and a hand at the end, and can be used to perform multiple tasks. Some tasks that can be performed by the arm are described below. In addition, by exchanging the robot hand attachment, it is possible to perform additional tasks such as tightening a screw (with a screw tightening attachment), or inspection of the Work using force sensors.

i) Part attachment: The robot hand is used to grasp a part from the part tray, take it to the current location of the base, and attach it to the base, e.g., attachment of a cover or insertion of a piston.

ii) Movement of the base: Using the robot hand, the arm grasps the base. The arm moves to the destination, and then places the Base onto a table or processing device. (Example: movement of the base from the table to a screw tightening device after a cover has been attached to the base).

(b). Processing Device ("Machine"): A machine which performs a specific function (tightening a screw, welding, painting, inspection) on a base, e.g., the specialized machine which tightens the screw attaching a part and the base.

(c). Slide (carry-in/carry-out) device: A device which moves an object from/to a predetermined source/destination. For example, the carry-in device moves a new, unprocessed Base onto a table in the assembly system. The carry-out device moves a completed Work out of the assembly system to another location.

(d). Assembly Table: The location where the base is placed in order to attach parts to it using an arm, e.g., in the example Work, the covers and pistons must be attached to the base at a table.

(e). Parts-Tray: The place where parts used by the assembly system are initially placed. When an arm attaches a part to a base, the arm must first move to the parts tray, grasp a part, and then attach it to the base.

### Parallelism in cellular assembly sequences

At any given time, a cellular assembly system can be processing multiple Works in parallel. While one device is attaching a part to a base, another device can be tightening the screw on another base. A setup such as the one shown in Figure 2-(a) normally processes 3-5 Works in parallel.

## Applying Domain-Independent Planning to Assembly Planning

We are developing an automated sequence generation system for the cellular assembly robot system described above. As shown below, the system can be modeled straightforwardly using PDDL. Since the objective is to generate a sequence which effectively uses resources in parallel, the system needs to output a concurrent plan. Thus, one approach would be to apply planning algorithms that are designed to handle concurrency.

However, one major goal for this research project was to investigate the use of off-the-shelf, domain-independent planners as the core planning engine, to see whether it is feasible to develop a system for our domain without having to develop/modify/maintain the core planner – given the rapid rate of improvement in the state of the art of domain independent planning, it is preferable to use the core planner as an easily replaceable, commodity tool, much like a programming language compiler, and focus our efforts on model design and system integration, implementing pre/post processors as necessary. While numerous, temporal planners that output parallel plans have been developed, we found that the older, publicly available systems (e.g., Temporal Fast Downward) are not visibly actively maintained, and do not have a substantial open-source development community. Furthermore, preliminary experiments showed that for the models described above, the results did not differ significantly from the sequential planners we used below.

On the other hand, sequential planning is a more mature technology, and the current state of the art planner, Fast Downward has a significant user base and development community, which makes it attractive as an off-the-shelf component for our system. Therefore, we investigated an approach where we use a sequential planner (e.g., Fast Downward) to generate an initial, sequential plan, which is postprocessed using a scheduler we implemented in order to generate a plan with concurrent actions.

## A PDDL Model for Assembly System

We modeled the assembly system using PDDL version2.1 There are 7 types of objects, representing the objects described in the previous sections: (i) Robot Arm ("Arm"), (ii) Base, (iii) Parts, (iv) Parts Tray, (v) Machine, (vi) Table, and (vii) Slide Device.

There are 8 types of actions:

(a) Move Arm: Moves the arm (?arm) from the source (?from) to the destination (?to). The occupied and not-occupied propositions enforce a mutual exclusion constraint so that only one arm can occupy any given location at a time.

```
(action: move_arm
parameters (?arm - arm ?from – position ?to - position)
preconditions    (at ?arm ?from) (not-occupied ?to) (reachable ?arm ?to)
effects    (at ?arm ?to) (not-occupied ?from) (occupied ?to)
        (not (at ?arm ?from)) (not (occupied ?from)) (not (not-occupied ?to)))
```

(b) Eject Base: Uses an arm (?arm) to grasp and pick up a base (?base) from a machine or a table (?pos). Resets the mutual exclusion constraint enforcing the rule that at most 1 base can be at a location (notbaseplaced ?pos)

```
(action: eject_base
parameters   (?base - base ?arm - arm ?pos - position)
preconditions    (at ?base ?pos) (at ?arm ?pos) (free ?arm)
effects    (hold ?arm ?base) (notbaseplaced ?pos)
        (not (free ?arm)) (not (at ?base ?pos)))
```

(c) Set Base Base: Commands an arm (?arm) that is holding a particular base (?base) to set the base on a machine or table (?pos). Each machine/table has a mutual exclusion constraint ensuring at most 1 base is placed on it (notbaseplaced).

```
(action: set_base
parameters   (?base - base ?arm - arm ?pos - position)
preconditions    (hold ?arm ?base)    (at ?arm ?pos) (notbaseplaced ?pos)
                 (baselocation ?pos))
effects(at ?base ?pos) (free ?arm)
        (not (hold ?arm ?base)) (not (notbaseplaced ?pos)))
```

(d) Slide Base: Uses a slide (carry-in/carry-out) device to move a base.

```
(action slide_base
parameters (?base - base ?from - position ?to - position)
preconditions    (at ?base ?from) (connect ?from ?to)
                 (notbaseplaced ?to) (baselocation ?to))
effects    (at ?base ?to) (not (at ?base ?from))
           (not (notbaseplaced ?to)) (notbaseplaced ?from)))
```

(e) Pick Parts by Arm: Use an arm (?arm) to pick up a part (?part). The part will later be used by a BaseAssemblePickedPartsXByArm action (see below).

```
(action: pickup-part_X
parameters (?part_X - part_X ?arm - arm ?pos - position)
precondition (free ?arm) (at ?arm ?pos) (at ?part_X ?pos)
effects    (hold ?arm ?part_X) (not (at ?part_X ?pos))
           (not (free ?arm)))
```

(f). Base Assemble by Machine: Use a machine (?pos) to perform an assembly operation (e.g., tighten the screw) on a base (?base).

The ordering constraints which are determined when the object is designed are encoded as a set of ordering propositions, finished_Step_X and unfihished_Step_X, which represent whether an assembly step X has been performed already.

```
(action: Base_Assemble_JobA_by_Machine
parameters (?base - base ?pos - position)
preconditions    (Assemble_Parts_B ?base) (at-Job_A ?pos) (at ?base ?pos)
effects    (finished_Job_A ?base) (not (unfinish_Job_A ?base)))
```

(g). Base Assemble Picked Parts by Arm: Uses an arm (?arm) to attach a part (?part) to a base.

```
(action: Base_Assemble_Picked_PartsX_by_Arm
parameters (?parts_X – parts_X ?base - base ?arm – arm ?pos - position)
preconditions    (finished_Job_A ?base) (unused ?parts_X)
                 (at ?base ?pos) (at ?arm ?pos)
                 (at-Assemble_PartsX ?pos) (hold ?arm ?parts_X)
effect     (Assemble_Parts_X ?base) (used ?parts_X)  (free ?arm)
            (not (unused ?parts_X)) (not (hold ?arm ?parts_X))
           (not (unfinish_JobB ?base)))
```

## Evaluation of the Assembly System Model

We evaluated the assembly system PDDL model using several publicly available, domain-independent planners to generate sequential plans for the model.

### Benchmark Problem

As a benchmark problem, we use the Layout shown in Figure 2-(a) and described in detail in the previous sections.

The assembly robot system consists of 2 Arms, 5 Machines, 2 Tables, and 2 Slide Devices. Note that Arm A cannot reach Machine C because Arm B is in the way.

| N (Number of Work) | | FF/WA* | seq-opt-fd-autotune | seq-sat-fd-autotune-1 | seq-sat-fd-autotune-2 | seq-sat-lama-2011 | Human-generated sequence |
|---|---|---|---|---|---|---|---|
| 1 | cost | 453 | 441 | 441 | 441 | 441 | 465 |
| | cost/Work | 453 | 453 | 441 | 441 | 441 | 465 |
| 2 | cost | 921 | 657 | 633 | 699 | 687 | 738 |
| | cost/Work | 460.5 | 328.5 | 316.5 | 349.5 | 343.5 | 369 |
| 3 | cost | 1302 | - | 1401 | 1290 | 1212 | 984 |
| | cost/Work | 434 | | 467 | 430 | 404 | 328 |
| 4 | cost | 1836 | - | 2253 | 1767 | 1560 | 1230 |
| | cost/Work | 459 | | 563.25 | 441.75 | 390 | 307.5 |
| 5 | cost | 2271 | - | 2913 | 2214 | 1917 | 1476 |
| | cost/Work | 454.2 | | 582.6 | 442.8 | 383.4 | 295.2 |
| 6 | cost | 2937 | - | 3627 | 2754 | 3393 | 1722 |
| | cost/Work | 587.4 | | 725.4 | 550.8 | 678.6 | 287 |

*Table 1: Parallel plans generated from PDDL model using four configurations of Fast Downward (with postprocessing for plan parallelization).*

The Work requires 1 base and 4 parts. 9 assembly operations are required, including 5 assembly operations that use a Machine, and 4 assembly operations that are performed with an Arm. The amount of time required to assemble this Work is 465.

## Initial/Goal States

In the initial state, N bases are at the carry-in device, and the final state has N completed pieces of Work in the carryout device (1 <= N <= 6).

We evaluated model using 5 configurations of Fast Downward, a state-of-the-art sequential planner [Helmert 2006, Fast Downward 2012].

(a). seq-sat-lama-2011 – a configuration which mimics the Lama 2011 planner, which won the IPC 2011 configuration [Richter et al 2011]

(b). seq-sat-fd-autotune-1

(c). seq-sat-fd-autotune-2

(d). seq-opt-fd-autotune

(e). FF/WA* - weighted A* (heuristic weight=6) using Fast Forward heuristic [Hoffman and Nebel 2001]

The first 4 configurations are standard configurations included in the current Fast Downward distribution. While seq-sat-lama-2011, seq-sat-fd-autotune-1, seq-sat-fd-autotune-2, and FF/WA** are satisficing configurations which are not admissible, seq-opt-fd-autotune uses only admissible heuristics.

The plans that are generated by Fast Downward are sequential plans (1 action executed at each step). These are postprocessed with a simple, earliest-dispatch scheduler in order to parallelize the plans.

The experiments were performed on an Intel Core i7 975 3.33GHz with 4GB RAM. The tested code is single-threaded, and was run until either the search completed, or RAM was exhausted. The results are shown in Table 1. The costs of the concurrent plans for assembling 1-6 Works are shown, as well as the cost per work (cost for the N-work sequence divided by number of Works).

The rightmost column in Table 1 shows the makespan of a human-generated (parallel) sequence for N Works.

Table 1 shows that for 1-2 Works, the plans generated by the planners are competitive with the human-generated sequence. However for 3-6 Works, the human generated sequences are significantly more efficient than the sequences generated by the planners, and we see that the cost/work for the automatically generated sequences is approximately twice the cost/work for the human-generated sequences. Inspection of the suboptimal plans showed that the suboptimality is due many unnecessary actions, i.e., suboptimality is due to problems in the sequential plans generated by Fast Downward, and not due to a failure to parallelize the plans optimally. For example, there is a tendency to repeatedly try to move the arm into the position where they should be in the goal state. It appears that current heuristics are deceived into prematurely trying to move the arms into their final positions; this problem may be caused by the long lengths of the plans that are required in this assembly domain (hundreds of steps).

While we included the seq-opt-fd-autotune configuration to see whether optimal sequential plans might lead to high-quality concurrent plans, the results for 1-2 Works shows that this is not necessarily the case (for 2 Works, the concurrent plan cost for seq-opt-fd-autotune is not as good as seq-sat-fd-autotune-1. Furthermore, the seq-opt-fd-autotune configuration fails for more than 2 Works, showing that search with nonadmissible heuristics is necessary for this domain model.

In addition to the results shown in Table 1, we experimented with numerous configurations that used the various heuristics included in the Fast Downward code distribution (context-enhanced heuristics, merge-and-shrink abstractions, landmark-based heuristics), and could not find a configuration that could find significantly higher-quality plans.

In addition to Fast Downward, we also evaluated SatPlan (Kautz et al 2006). However, SatPlan exhausts memory during the conversion to SAT phase when the number of Works was more than 1.

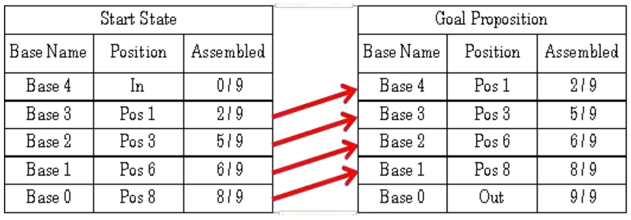
Based on these results, we concluded that it is difficult to obtain high-quality plans by applying current state-space search based planners to the standard PDDL model of our assembly robot domain described above.

## A Steady-State Model for Assembly

In the previous section, we evaluated the assembly of up to N=6 Works. However, in practice, an order to be fulfilled by an assembly robot system typically requires the assembly of 20-100 Works of each type, and a total of over 1000 Works for an entire order. The results in the previous section indicate that a standard approach where we simply increase the number of Works to be generated in the PDDL problem file does not seem like a viable approach, as we have already shown that there are scaling issues even with N>2 Works.

Therefore, we developed an alternative approach to modeling the system which naturally scales to large number of Works. Instead of starting with an “empty” start state and searching for a path to a goal state where all of the Work has exited the system through the carryout device, we formulate a steady-state problem, where the start and goal states are essentially the same, except that all objects have moved “forward” a step.

The manually generated sequences that are currently used in production are called steady-state sequences (Dawande et al 2005). In a steady-state sequence, start and goal states are basically identical, if we ignore the unique object IDs for objects that are the same type. When a single cycle of a steady-state sequence is executed, one Work is completed, and one new base enters the system. The sequence can be repeatedly executed an arbitrary number of times (as long as new bases are provided to the carry-in tray and parts are provided to the parts tray), and multiple Works are simultaneously processed.



| Start State | | | Goal Proposition | | |
|---|---|---|---|---|---|
| Base Name | Position | Assembled | Base Name | Position | Assembled |
| Base 4 | In | 0/9 | Base 4 | Pos 1 | 2/9 |
| Base 3 | Pos 1 | 2/9 | Base 3 | Pos 3 | 5/9 |
| Base 2 | Pos 3 | 5/9 | Base 2 | Pos 6 | 6/9 |
| Base 1 | Pos 6 | 6/9 | Base 1 | Pos 8 | 8/9 |
| Base 0 | Pos 8 | 8/9 | Base 0 | Out | 9/9 |

*Table 2: Start and end states in a steady-state model*

We developed a PDDL model in order to generate a steady-state sequence. A start state consists of an unprocessed base is at the carry-in device and 4 bases in various stages of processing. The goal state contains 1 completed Work, and each of the 4 Works that were incomplete in the start state have advanced 1 additional processing state (and are placed in new, appropriate locations). For example, Table 2 shows an example of a start state and goal state for this stead-state formulation. The “Assembled” column shows the number of assembly steps that have been completed already for a particular base. For example, in the start state, Base 0 is at Position 8, and has 8 out of 9 assembly steps completed.

When a plan for the model in Table 2 is executed, the system goes through 1 cycle of this steady-state model. Base0 is completed and exits the system. Base 1, goes through one more stage of processing, and moves to the same location as Base 0 in was (8/9 assembled, at Pos 8). Similarly, relative to the start state, Base 2 replaces Base 1, Base 3 replaces Base 2, and Base 4 replaces Base 3. Finally, by adding a new base(Base 5) in the carry-in device, we end up with a state where the locations and the processing states of all of the bases are the same as in the start state, except for the IDs.

Designing a steady-state model requires identifying an appropriate steady-state, and this does require some domain knowledge. However, this is relatively straightforward. As explained above, in the assembly robot domain, the order in which assembly steps are applied to a new base, resulting in a completed Work, is determined during the design of the Work, so the major degree of freedom remaining is the scheduling of the arm motions. A simple heuristic approach is to place a base at each location, in the correct state that the base needs to be in such that the preconditions for being in that position (these are already fully defined in the PDDL domain model described above) are satisfied. This is sufficient to specify a candidate start/goal state for a steady-state model. While this is currently done manually, automated generation of the steady-state model (e.g., by inferring feasible steady-states directly from the basic PDDL domain model) is a direction for future work.

The PDDL model for this steady-state formulation was evaluated using two configurations of Fast Downward: (a) Weighted A* with the Fast Forward Heuristic, weight=6, and (b) the seq-sat-lama-2011 configuration. The sequential plans generated by Fast Downward were then converted into parallel plans using the same postprocessor used above in the experiments for the simple PDDL model.

In order to actually execute a steady-state model plan, the execution time/cost will need to also include the setup time required to get from an initial state with no bases in the system (i.e., the start state used in the standard model) to the start state of the steady-state model. In addition, we must also generate a sequence that "flushes" the system by starting at the goal state of the steady-state model and ends up with all Work completed and out the carry-out tray. If the number of Work to be produced is small, these overheads can be significant, but in practice, 10's or 100's of Work need to be produced, so this one-time overhead is not significant.

We evaluated the system using the Layout model for Figure 2-(a) (Model 1-1), as well as 5 other product models. In Table 3, Model 1-1 is the same model used for the experiment in Table 1, except that a steady-state formulation is used (the basic models are the same, but the start/goal states are as described above for the steady-state model). In Models 1-2 and 1-3, the same assembly system layout as Model 1-1 is used (i.e., Figure 2-(a)), but different products are assembled. Models 2-1, 2-2, and 2-3 use the assembly system layout in Figure 2-(b), which has 1 Arm, 3 Tables, and 2 Machines.

Table 3 shows the results. First, note that for Model 1-1, which is the same domain used in the experiments for the straightforward PDDL model (Table 1), the cost of one cycle of the steady-state model is 258. Even accounting for the fact that the cycle makespans for the steady-state model do not include the setup/cleanup required to initialize the start state and flush the goal state, this is clearly significantly lower than the cost/work for the planner-generated sequences in Table 1. Furthermore, while the standard PDDL model becomes increasingly less efficient (increasing cost/Work) as the number of Works increases, the steady-state model incurs the same cost per cycle (i.e., per Work) for an arbitrary number of Works, and as previously mentioned, the setup/cleanup overhead for the steady-state model can be amortized over a large number of cycles.

The "Make 1 Work" column shows the optimal makespan for sequentially assembling 1 Work for each of the models. The cycle makespan for the steady-state plans are comparable with the time required to assemble 1 Work in that model, indicating that they are also fairly efficient in an absolute sense (not just relative to the straightforward PDDL model).

| Model | seq-sat-lama-2011 +postprocessing | WA+/FF + postprocessing | Make 1 Work (sequential) |
|---|---|---|---|
| 1.1 | 258 | 258 | 465 |
| 1.2 | 456 | 270 | 585 |
| 1.3 | 177 | 174 | 471 |
| 2.1 | 228 | 228 | 357 |
| 2.2 | 354 | 342 | 411 |
| 2.3 | 270 | 300 | 369 |

*Table 3. Steady-state model results.*

We also compared the steady-state plan generated by the system for Model 1-1 with a steady-state plan generated by a human expert. The manually crafted (parallel) plan has a cycle cost of 153, compared to 258 for the automatically generated steady-state plan. Thus, there is room for improvement.

However, using a prototype, proprietary implementation of weighted A* using the FF heuristic instead of the WA*/FF configuration for Fast Downward, we have obtained a steady-state plan with cost 144, which is better than the manually crafted sequence. There is no fundamental algorithmic difference between our prototype implementation of WA*/FF and Fast Downward WA*/FF, so this difference seems to be due to low-level details (e.g., tie-breaking). We are currently investigating the cause for this performance discrepancy..

## Related Work

Manufacturing applications of automated planning has been studied by previous researchers (c.f. Nau, Gupta, Regli 1995; Castillo, Fdez-Olivares, Gonzalez 2001; Fernandez, Aler, Borrajo 2005; Klein, Johnsson, Backstrom 1999). As far as we know, this is the first investigation of domain-independent planning for a multi-arm, assembly robot system. The problem considered in this paper is more constrained than previous research because the processing order for attaching parts to the base is predetermined, and the main remaining problem is how to coordinate the multiple arms and machines. While previous work has focused on models which, like our initial PDDL model, start with raw parts and result in one or more products, we introduced a new cyclic, steady-state formulation where the start and goal states are identical modulo part IDs, which is applicable when the processing ordering is highly constrained.

## Conclusions

The problem of generating sequences for cellular assembly systems is well suited for domain-independent planning techniques because there are a multitude of constraints, and a constantly changing demand for different types of products, which makes automation of sequence generation highly desirable.

We applied domain-independent planning to sequence generation for cellular assembly systems. We showed that standard cellular assembly systems could be modeled using PDDL. However, state-of-the-art planning algorithms such as those implemented on Fast Downward have some difficulties generating sequences for these domains. We developed a steady-state approach for modeling a continuous production cycle, which enabled us to generate much more efficient sequences which can be used to assemble an arbitrary amount of the same product. Directions for future work include automated generation of the steady-state model, as well as investigation of other approaches such as HTN planning.

## References

Castillo L, Fdez-Olivares J, González A. Mixing expressiveness and efficiency in a manufacturing planner. Journal of Experimental and Theoretical Artificial Intelligence, 13, pp. 141-162, 2001.

Dawande M, Geismar H.N, Sethi S, Sriskandarajah C. Sequencing and scheduling in robotic cells: recent developments. Journal of Scheduling, 8:387-426, 2005.

Fast Downward, downloaded from www.fast-downward.org (used latest version as of September 10, 2012).

Fernandez S, Aler R, Borrajo D. Machine learning in hybrid hierarchical and partial-order planners for manufacturing domains. Applied Artificial Intelligence, 19(8):783-809, 2005.

Fox, M., and Long, D. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. J. Artificial Intelligence Research 20:61-124, 2003.Helmert M. The Fast Downward Planning System. Journal of Artificial Intelligence Research 26:191-246, 2006.

Henry Kautz, Bart Selman, and Joerg Hoffmann. SatPlan: Planning as Satisfiability Abstracts of the 5th International Planning Competition, 2006.

Hoffmann J, Nebel B: The FF Planning System: Fast Plan Generation Through Heuristic Search. Journal of . Artificial Intelligence Research. 14: 253-302, 2001.

IHI 2012 , Precision Assembly Robot System. Journal of IHI Technologies Vol.52 No.1, 12-15

Klein I, Jonsson P, Backstrom C. Efficient planning for a miniature assembly line. Artificial Intelligence in Engineering 13:69-81, 1999.

Nau D, Gupta S, Regli W. AO planning versus manufacturing-operation planning: a case study. Proc. IJCAI, 1995.

Richter S, Westphal M, Helmert M. LAMA 2008 and 2011. Short paper for the International Planning Competition 2011 proceedings (ICAPS2011)

# Generating Alternative Plans for Scheduling Personal Activities

**Anastasios Alexiadis** and **Ioannis Refanidis**
Department of Applied Informatics, University of Macedonia,
Egnatia 156, 54006, Thessaloniki, Greece.
talex@java.uom.gr, yrefanid@uom.gr

**Abstract**

Alternative plan generation can be used to meet the user's preferences in a scheduling problem, when either the scheduling model does not take every available preference into account, or the user does not specify his preferences correctly in a formal manner. In this paper, an alternative plan generation method that takes into account in-domain specific attributes of the scheduling problem, is applied to an electronic calendar management scheduler in order to generate significantly different qualitative alternative plans. The scheduler is based on an adaption of the Squeaky Wheel Optimization Framework (SWO) and addresses the problem of optimizing individual activity plans (that concern activities one person has to accomplish independently of others), based on a rich model for their specification involving complex constraints and preferences and enhanced with extra attributes that measure the distance of each plan from the already found.

## Introduction

Generating alternative plans for planning and scheduling problems is another way of ensuring that at least one plan will meet the user's preferences. According to Kambhampati (Kambhampati 2007), in many real world planning scenarios the user's preferences are either unknown or at best partially specified. Other systems attempt to elicit user preferences in a non-intrusive manner, by presenting alternative plans to the user and building a preference model based on his choices (Berry et al. 2011), (Myers et al. 2007).

Intelligent assistance with time and task management has been targeted by many AI researchers (Myers et al. 2007), (Freed et al. 2008), (Refanidis 2007) (Refanidis and Alexiadis 2011), (Berry et al. 2011), (Bank et al. 2012). Electronic calendar applications are usually based on a series of fully specified and independent events. These events are specified by a fixed start-time, a duration for the event and usually a location. In addition, many systems support tasks. These represent individual commitments potentially having a deadline to be met (e.g., preparing for a lecture or planning for a trip). Tasks are kept in separate task lists and do not have a specific start time. Conversion of a task to event is, usually straightforward, accomplished by dropping the task into the electronic calendar.

(Refanidis and Yorke-Smith 2010) presents a model that treats events and tasks in a uniform way. They are specified as *activities* in the model and are characterized by a number of attributes, such as a temporal domain, a duration range, a set of alternative locations, interruptibility, utilization, preferences over the temporal domain and alternative durations, constraints and preferences over the way the parts of an interruptible activity are scheduled in time. The model also supports binary constraints (ordering, proximity and implication), as well as preferences between pairs of activities, thus specifying a Constraint Optimization Problem (COP). In the same work a scheduler, based on the Squeaky Wheel Optimization framework (SWO) and combined with domain-dependent heuristics to automatically schedule activities, is presented. (Alexiadis and Refanidis 2012) presents a post-processing module for SWO that increases solution quality via local-search post-optimization.

There are a number of ways for introducing alternative plan generation in a planner or scheduler. In the literature we found two main approaches for tackling this issue. One being the management of either the set of states or the order in which they are evaluated by the search algorithm (Roberts et al. 2012), (Dechter, Flerova, and Marinescu 2012). The other—which we based our alternative generation method on—being the modification of the heuristic of the planning algorithm (Coman and Muñoz-Avila 2011), (Nguyen et al. 2012).

In this paper we present our approach to generate qualitatively, significantly different alternative plans, based on the enhancement of the plan evaluation function used by the scheduler during the optimization process with additional attributes that measure the differences with the already generated solutions. We define the plan difference function that takes into account domain-specific traits of the problem formulation, as defined in the SWO article (Refanidis and Yorke-Smith 2010).

The rest of the paper is structured as follows. Firstly, we formulate the optimization problem and illustrate the SWO-based approach. Next, we present the new evaluation function that considers (among other criteria) differences with one generated plan. We proceed to define the plan difference function (*PDiff*) for comparing two plans. Afterward, we extend the above functions to work with an arbitrary number of pre-generated solutions. In the Evaluation Section we demonstrate their usage on a number of randomly generated problem instances. Finally, we conclude the paper and identify directions of future work.

## Background

In this Section we present the problem formulation, as well as the SWO approach to cope with the problem.

### Problem Formulation

In previous work (Refanidis and Yorke-Smith 2010), time is considered a non-negative integer, with zero denoting the current time. A set $T$ of $N$ activities, $T = \{T_1, T_2, \ldots, T_N\}$, is given. For each activity $T_i \in T$, its minimum duration is denoted with $d_i^{min}$ and its maximum duration with $d_i^{max}$. The decision variable $p_i$ denotes the number of parts in which the *i-th* activity has been split, with $p_i \geq 1$. $T_{ij}$ denotes the *j-th* part of the *i-th* activity, $1 \leq j \leq p_i$. The sum of the durations of all parts of an activity must be at least $d_i^{min}$ and no greater than $d_i^{max}$.[1] For each $T_{ij}$, the decision variables $t_{ij}$ and $d_{ij}$ denote its start time and duration. The sum of all $d_{ij}$, for a given $i$, must equal $d_i$.[2] Non-interruptible activities are scheduled as one part.

For each $T_i$, we define the minimum and maximum part duration $smin_i$ and $smax_i$,[3] as well as the minimum and maximum temporal distances between every pair of parts, $dmin_i$[4] and $dmax_i$.[5]

For each activity $T_i$, its temporal domain is defined as a set of temporal intervals defining $D_i = [a_{i1}, b_{i1}] \cup [a_{i2}, b_{i2}] \cup \ldots \cup [a_{i,F_i}, b_{i,F_i}]$, where $F_i$ is the number of intervals of $D_i$.[6]

A set of $M$ locations, $Loc = \{L_1, L_2, \ldots, L_M\}$, as well as a two dimensional, not necessarily symmetric, matrix $Dist$ that holds the temporal distances between locations are given. Each activity $T_i$ has a set of possible locations $Loc_i \subseteq Loc$, where its parts can be scheduled. The decision variable $l_{ij} \in Loc_i$[7]denotes the particular location where $T_{ij}$ is scheduled.[8]

Activities may overlap in time. Each activity $T_i$ is characterized by a utilization value, $utilization_i$.[9] At any exact time point, the set of scheduled activities should have compatible locations (i.e., locations with no temporal distance to each other as a person cannot be in two places at the same time) and the sum of their utilization values should not exceed the unit (the time point *maximum* utilization value).

The model supports four types of binary constraints: Ordering constraints, minimum and maximum proximity constraints and implication constraints. An ordering constraint between two activities $T_i$ and $T_j$, denoted with $T_i < T_j$, implies that no part of $T_j$ can start its execution before all parts of $T_i$ have finished.[10]A minimum (maximum) distance binary constraint between activities $T_i$ and $T_j$ implies every two parts, one of $T_i$ and another of $T_j$, must have a given minimum (maximum) temporal distance.[11] Finally, an implication constraint of the form $T_i \Rightarrow T_j$ implies that in order to include $T_i$ in the plan, $T_j$ should be included as well.[12]

Scheduling personal activities is considered a constraint optimization problem. That said, the empty schedule is a valid schedule but with low utility, thus we are interested in better schedules. There are several sources of utility. The main source concerns the activities themselves. Each activity $T_i$ included in the schedule contributes utility $U_i(d_i)$ that depends on its allocated duration. The way $T_i$ is scheduled by a schedule $\pi_i$ within its temporal domain constitutes another source of utility, $U_i^{time}(\pi_i)$. The user can define linear and stepwise utility functions of time over the temporal domain of each activity.

Any form of hard constraint can also be considered a soft constraint that might contribute utility. So, minimum and maximum distance constraints between the parts of an interruptible activity might contribute $U_{dmin_i}(\pi_i)$ and $U_{dmax_i}(\pi_i)$ respectively. Similarly, binary preferences can be defined as well over the way pairs of activities are scheduled. Especially for ordering and proximity preferences, partial satisfaction of the preference is allowed. The Degree of Satisfaction for a partial preference $p$, denoted with $DoS(p)$, is defined as the ratio of the number of pairs of parts, one from $T_i$ and another from $T_j$, for which the binary preference holds, to the total number of pairs of parts.

To summarize, the optimization problem is formulated as follows:

**Given:**

1. A set of $N$ activities, $T = \{T_1, T_2, \ldots, T_N\}$, each one of them characterized by its duration range, duration utility profile, temporal domain, temporal domain preference function, utilization, a set of alternative locations, interruptibility property, minimum and maximum part sizes as well as required minimum and maximum part distances for interruptible activities, preferred minimum and maximum part distances and the corresponding utilities.

2. A two-dimensional matrix with temporal distances between all locations.

---

[1]$$\forall T_i,\ d_i^{min} \leq d_i \leq d_i^{max} \text{ OR } d_i = 0 \quad (C1)$$

[2]$$\forall T_{ij}, \sum_{j=1}^{p_i} d_{ij} = d_i \quad (C2)$$

[3]$$\forall T_{ij}, smin_i \leq d_{ij} \leq smax_i \quad (C3)$$

[4]$$\forall T_{ij}, T_{ik}\ j \neq k \Rightarrow t_{ij} + d_{ij} + dmin_i \leq t_{ik} \vee t_{ik} + d_{ik} + dmin_i \leq t_{ij} \quad (C4)$$

[5]$$\forall T_{ij}, T_{ik}\ j \neq k \Rightarrow t_{ij} + dmax_i \geq t_{ik} + d_{ik} \wedge t_{ik} + dmax_i \geq t_{ij} + d_{ij} \quad (C5)$$

[6]$$\forall T_{ij}, \exists k, 1 \leq k \leq F_i : a_{ik} \leq t_{ij} \leq b_{ik} - d_{ij} \quad (C6)$$

[7]$$l_{ij} \in Loc_i \quad (C7)$$

[8]$$\forall T_{ij}, T_{mn}, T_{ij} \neq T_{mn} \wedge (Dist(l_{ij}, l_{mn}) > 0 \vee Dist(l_{mn}, l_{ij}) > 0) \Rightarrow t_{ij} + d_{ij} + Dist(l_{ij}, l_{mn}) \leq t_{mn} \vee t_{mn} + d_{mn} + Dist(l_{mn}, l_{ij}) \leq t_{ij} \quad (C8)$$

[9]$$\forall t, \sum_{\substack{T_{ij} \\ t_{ij} \leq t < t_{ij} + d_{ij}}} utilization \leq 1 \quad (C9)$$

[10]$$\forall T_i, T_j, T_i < T_j \Leftrightarrow d_i > 0 \wedge d_j > 0 \Rightarrow \forall T_{ik}, T_{jl}, t_{ik} + d_{ik} \leq t_{jl} \quad (C10)$$

[11]$$\forall T_{ik}, T_{jl}, t_{ik} + d_{ik} + dmin_{ij} \leq t_{jl} \vee t_{jl} + d_{jl} + dmin_{ij} \leq t_{ik} \quad (C11)$$

$$\forall T_{ik}, T_{jl}, t_{ik} + dmax_{ij} \geq t_{jl} + d_{jl} \wedge t_{jl} + dmax_{ij} \geq t_{ik} + d_{ik} \quad (C12)$$

[12]$$\forall T_i, T_j, T_i \Rightarrow T_j \Leftrightarrow d_i > 0 \Rightarrow d_j > 0 \quad (C13)$$

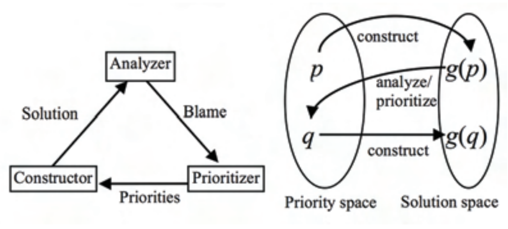


Figure 1: (a) The SWO cycle. (b) Coupled search spaces

3. A set $C$ of binary constraints (ordering, proximity and implication) over the activities.
4. A set $P$ of binary preferences (ordering, proximity and implication) over the activities.

**Schedule**

the activities in time and space, by deciding the values of their start times $t_{ij}$, their durations $d_{ij}$ and their locations $l_{ij}$, while trying to maximize the following objective function:

$$U = \sum_{\substack{i \\ d_i \geq d_i^{min}}} \begin{array}{c}(U_i(d_i) + U_i^{time}(\pi_i) + U_i^{dmin}(\pi_i) \\ +U_i^{dmax}(\pi_i))\end{array} \quad (1)$$

$$+ \sum_{p(T_i,T_j) \in P} u_p \times DoS(p(T_i, T_j))$$

subject to constraints (C1) to (C13).

## The SWO Approach

(Refanidis and Yorke-Smith 2010) solves the problem using the Squeaky Wheel Optimization (SWO) framework (Joslin and Clements 1999). At its core, SWO uses a Construct/Analyze/Prioritize cycle as shown in Figure 1(a). The solution is found by a greedy approach, where decisions are based on an order of the tasks determined by a priority queue. The solution is then analyzed to obtain the tasks that cannot be scheduled. Their priorities are increased, enabling the constructor to deal with them earlier on the next iteration. The cycle will be repeated till a termination condition occurs. The algorithm searches in two coupled spaces, as shown in Figure 1(b). These are the priority and solution spaces. Changes in the solution space are caused by changes in the priority space. Changes in the priority space occur as a result of analyzing the previous solution and using a different order of the tasks in the priority queue. A point in the solution space represents a possible solution to the problem.

SWO can easily be applied to new domains. The fact that it gives variation on the solution space makes it different than more traditional local search techniques such as WSAT (Selman, Kautz, and Cohen 1995). (Refanidis and Yorke-Smith 2010) presents how SWO was adapted to to the Constraint Optimization Problem presented in the previous Section.

## Generating Mutiple Plans

In this Section we present our work on generating and evaluating qualitatively, significantly different alternative plans. We begin by presenting a method to generate the first alternative plan (after the standard one is found) and we continue with extending the method so as it can generate an arbitrary number of alternative plans.

### Generating Qualitatively, Significantly Different Alternative Plan

Both the model and the scheduler have been designed with the assumption of generating *one* high-quality schedule for a set of activities. For generating an arbitrary number of qualitatively, significantly different alternative schedules, so as the user can choose one according to her preferences, we extended the model so as to integrate the notion of the *value* of an alternative plan.

We define the value of an alternative plan as a linear combination of its utility (1) with its deviation from the already generated plans. In the simplest case, when there is a single generated plan $\pi'$ to which we want to compare an alternative plan $\pi$, we adopt the following formula for evaluating $\pi$, as a replacement to formula (1):

$$V(\pi) = U(\pi) + C \times U(\pi') \times PDiff(\pi, \pi') \quad (2)$$

where $C$ is a constant that weights $\pi$'s difference over $\pi'$ and $PDiff(\pi, \pi')$ is a function assessing the two plans' differences. As it will be shown later in this Section, $PDiff(\pi, \pi')$ ranges between 0 and 1. The higher $PDiff(\pi, \pi')$ is, the greater the differences between the two plans; $PDiff(\pi, \pi') = 0$ stands for no differences.

In the initial stage, when the main plan hasn't been generated yet, $U(\pi') = 0$, so the above formula is simplified to $V(\pi) = U(\pi)$, thus resulting in no differences over the main plan generation procedure. When the scheduler has already found a plan for the user, it will try to maximize concurrently both the utility of the current plan, $U(\pi)$, as well as the deviation from the already found plan, $C \times U(\pi') \times PDiff(\pi, \pi')$. Note that the latter term of the sum depends both on the difference between the two plans and on the quality of the already found one. The factor of $U(\pi')$ is introduced in order to scale two terms of the sum similarly.

### Quantifying the Degree of Deviation Between Two Plans

We measure the deviation between two plans, $\pi$ and $\pi'$, by function $PDiff(\pi, \pi')$, which takes into account the following metrics:

1. The change in the total duration of each plan's activity.
2. The change in the location of each plan's activity or part of it.
3. The change in the time windows where the various parts of each activity have been scheduled.
4. The change in the order in which pairs of activities have been scheduled.

In order to define precisely the above metrics, we introduce two extra functions. First we define function $\tau(\pi, T_i, x) = t \in D_i$, which for a schedule $\pi$, an activity $T_i$ and time-slot index $x$, $1 \leq x \leq d_i$, maps it into the *x-th* time-slot of that activity, as scheduled in $\pi$. Time-slots are individual discrete points of time. So, this function maps the *order* of a time-slot of an activity in the solution, to the absolute time where this time slot has been scheduled. Similarly, we define function $\lambda(\pi, T_i, x) = l \in Loc_i$, which maps the triple $(\pi, T_i, x)$ to the location where the *x-th* time-slot of $T_i$ has been scheduled, according to $\pi$.

**Durations Deviation:** For the activities $T_i \in T$, which appear in at least one of the two plans, $\pi$ and $\pi'$, the total duration deviation between the two plans is computed according to formula:

$$\Delta D = \frac{\sum_i |d_i - d_i'|}{\sum_i max(d_i, d_i')} \tag{3}$$

where $d_i$ is the duration of $T_i$ in plan $\pi$ and $d_i'$ is the duration of $T_i$ in plan $\pi'$. $\Delta D$ ranges between 0 and 1. If $T_i$ appears in one of the two plans, its duration in the plan that does not appear in is considered zero.

**Locations Deviation:** For the activities $T_i \in T$, which appear in at least one of the two plans, $\pi$ and $\pi'$, the total location deviation between the two plans is computed according to formula:

$$\Delta L = \frac{1}{\sum_i min(d_i, d_i')} \times \sum_i \sum_{\substack{x=1 \\ \lambda_x^{T_i} \neq \lambda'^{T_i}_x}}^{min(d_i, d_i')} 1 \tag{4}$$

where $\lambda_x^{T_i} = \lambda(\pi, T_i, x)$ and $\lambda'^{T_i}_x = \lambda(\pi', T_i, x)$. $\Delta L$ ranges between 0 and 1. If $T_i$ appears in one of the two plans only, $\Delta L$ is set equal to 1.

**Absolute Time Deviation:** For each activity $T_i \in T$, which appears in both plans $\pi$ and $\pi'$, the total absolute time deviation for $T_i$ among the two plans is computed according to formula:

$$\Delta Time_i = \sum_{x=1}^{min(d_i, d_i')} \frac{|\tau_x^{T_i} - \tau'^{T_i}_x|}{max_{\tau_{D_i}} - min_x(\tau_x^{T_i}, \tau'^{T_i}_x)} \tag{5}$$

where $max_{\tau_{D_i}}$ is the is the maximum time-slot of the domain of activity $T_i$, $\tau_x^{T_i} = \tau(\pi, T_i, x)$ and $\tau'^{T_i}_x = \tau(\pi', T_i, x)$. $\Delta Time_i$ ranges between 0 and 1. For activities $T_i$ appearing in one only of $\pi$ and $\pi'$, we define $\Delta Time_i = 1$. On the other hand, for activities $T_i$ appearing in none of $\pi$ and $\pi'$, we define $\Delta Time_i = 0$.

The overall absolute time deviation is defined as:

$$\Delta Time = \sum_{i=1}^{N} \Delta Time_i \tag{6}$$

**Ordering Differences:** For each pair of activities, $T_i$ and $T_j \in T$, which both appear in a plan $\pi$, the precedence of $T_i$ over $T_j$ in $\pi$ is computed as:

$$\nu_{ij} = \frac{1}{d_i \times d_j} \sum_{x=1}^{d_i} \sum_{y=1}^{d_j} \begin{cases} 1 \text{ if } \tau_x^{T_i} \leq \tau_y^{T_j} \\ 0 \quad \text{otherwise} \end{cases} \tag{7}$$

That is, $\nu_{ij}$ represents the percentage of pairs of parts, one from $T_i$ and one from $T_j$, such that the part of $T_i$ is scheduled not later than the part from $T_j$. $\nu_{ij}$ ranges between 0 and 1. It $T_i$ appears in $\pi$ but $T_j$ does not appear, we define $\nu_{ij} = 1$. If $T_i$ does not appear in $\pi$ (irrelevant to whether $T_j$ appears in $\pi$ or not), we define $\nu_{ij} = 0$.

Subsequently, the ordering deviation for a pair of activities, $T_i$ and $T_j$ in two plans, $\pi$ and $\pi'$ is defined as:

$$\Delta O_{ij} = |\nu_{ij} - \nu_{ij}'| \tag{8}$$

where $\nu_{ij}'$ refers to plan $\pi'$. $\Delta O_{ij}$ ranges between 0 and 1.

Finally, we define the total ordering deviation as:

$$\Delta O = \frac{2}{N \times (N-1)} \times \sum_{i=1}^{N} \sum_{j=i+1}^{N} |\nu_{ij} - \nu_{ij}'| \tag{9}$$

$\Delta O$ also ranges between 0 and 1.

## Plan Difference Between Two Plans

Based on the above formulas, we define the plan difference between two plans, $PDiff(\pi, \pi')$ as:

$$PDiff(\pi, \pi') = \begin{array}{c} \Delta D \times W_D + \Delta L \times W_L \\ +\Delta Time \times W_{Time} + \Delta O \times W_O \end{array} \tag{10}$$

where $W_D$, $W_L$, $W_{Time}$ and $W_O$ are non-negative weights and $W_D + W_L + W_{Time} + W_O = 1$. The user can specify the values of the weights, thus specifying his preferences on multiple plan generation (thas is, emphasizing his priorities over the above attributes of an already found plan). If an activity is not scheduled in one of the two plans, it obtains the full difference penalty as it is considered to deviate in all the above methods.

## Generating More than Two Alternative Plans

In order to generate more than two plans, we extended formula (2) as follows:

$$V(\pi) = U(\pi) + C \times \frac{\sum_{\forall \pi' \in \Delta} U_{\pi'} \times PDiff(\pi, \pi')}{|\Delta|} \tag{11}$$

where $\Delta$ is the set of the already generated plans and $C$ is a parameter representing the weight standing for the user's preference on the deviation from the already found plans versus the utility of the alternative plan.

The new formula for calculating the utility $V(\pi)$ (formula 9) is simplified back to formula (2), when comparing between only two plans (the one being evaluated and one pregenerated). When generating the original plan it is simplified back to the original utility function (1).

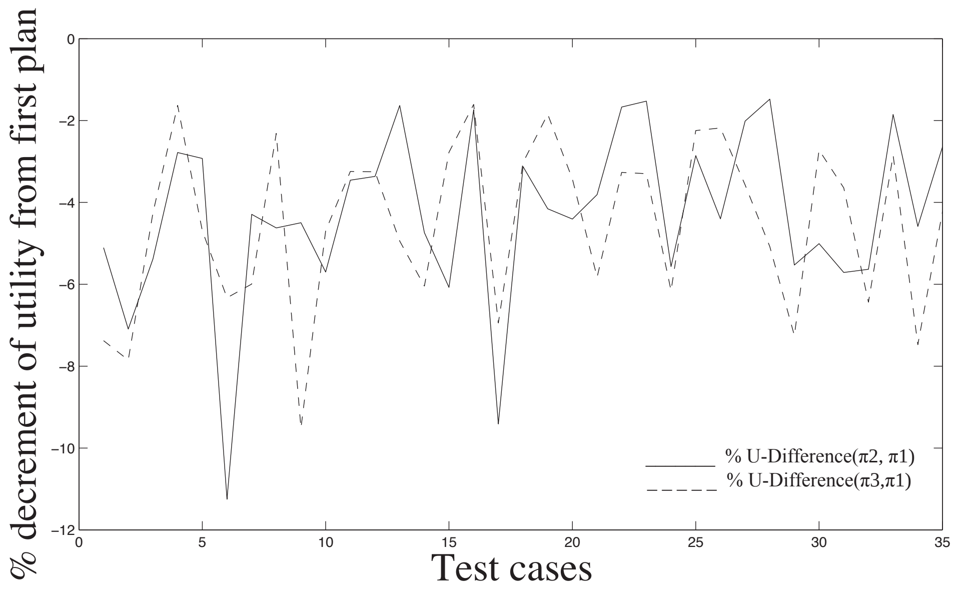


Figure 2: Percentage difference of utility between the alternative plans and the original

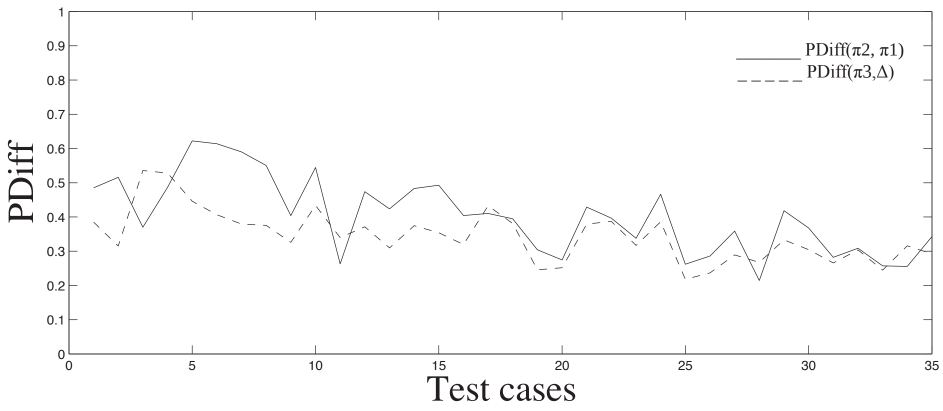


Figure 3: PDiff values for the two alternative plans for each test case

## Evaluation

The scheduler SWO (written in C++) was extended to include the alternative evaluation function $V$ (using $PDiff$), when generating plans and to allow the generation of more than one plan, by means of restarting itself. At each restart a dynamic list $\Delta$, holding the already generated plans, gets updated.

We generated three plans, on $35$ random test cases, ranging in size from problem instances of six activities to thirty six in steps of five. We set the parameters used for the alternative plan generation as follows: $C = 1.0, W_D = W_L = W_{Time} = W_O = 0.25$.

In Figure 2 we show the utility differences between the three plans, for each of the 35 problem instances of the test set. The solid line represents the utility percentage difference between the second plan (first alternative) and the original plan, whereas the dashed line represents the utility percentage difference between the third plan and the original. The original's (first plan's) quality is represented by the top line. We use the original utility function (formula 1) when comparing plan quality, as shown in the figure. $V$ is used for the generation phase only, when an intermediate solution gets evaluated while the scheduler optimizes the plan. The two alternative plans are of a slight lower utility value than the original plan though they are not far below in quality. The drop in utility for the second plan, in comparison to the first, ranges from $1.47\%$ to $11.25\%$. For the third plan the minimum and maximum drops are $1.6\%$ and $9.51\%$ respectively. The average drop for the second plan is $4.2\%$ and of the third $4.5\%$.

In Figure 3 we represent the *PDiff* values for the two alternative plans, the first alternative comparing to the original plan, and the second alternative comparing to the original plus the first alternative. The difference of each alternative plan to the original depends heavily on each problem instance's specific attributes, such as the available domain for each activity (more so than the number of activities in the problem instance), how strong (in utility values) are the preferences set on it and how many of them exist, as well how they intertwine. The minimum *PDiff* value was one of $0.2179$, in the second (after the original) alternative plan of a problem instance. The maximum value was one of $0.6224$, in the first (after the original) alternative plan of another problem instance.

One looking at the actual alternative solutions immediately sees apparent changes in many of the activities' parts temporal positions (the most common change), as well as their durations (another common change when applicable) and locations (rarest, as location selection don't provide a utility to standard SWO, except in the alternative plan generation phase). Changing the $C$ value parameters will shift the preference to either more different plans (of lower standard utility) or similar ones to the original.

## Conclusions and Future Work

The SWO scheduler with the alternative plan generation method, presented in this paper, manages to generate an arbitrary number of alternative plans for a user's scheduling problems, so the user can choose the best one suited to her needs. It is based on the premise that a user will not always be able to specify her constraints and preferences correctly in the formal model—which has been found to be the case in many real situations. In such cases, offering a number of alternative plans to the user provides her with more possibilities for picking a plan according to her actual preferences. A number of parameters for this method are also customizable by the user, enabling her to choose which in-domain characteristics she considers more important.

This paper presents our work on an alternative plan generation method, which can be potentially enriched with the exploration of more characteristics of domain-specific attributes of the scheduling problem being solved. Particularly, location differences should be further examined, as the model does not support location preferences but only alternative location options. Other domain dependent attributes to measure the deviation between two plans, either at the activity level or at a global level, could be considered as well. Last of all, alternative plans of user-given problem instances will be evaluated.

## Acknowledgements

The authors are supported for this work by the European Union and the Greek Ministry of Education, Lifelong Learning and Religions, under the program "Competitiveness and Enterprising" for the areas of Macedonia-Thrace, action "Cooperation 2009", project "A Personalized System to Plan Cultural Paths (myVisitPlanner$^{GR}$)", code: 09SYN-62-1129.

ΕΥΡΩΠΑΪΚΗ ΕΝΩΣΗ
ΕΥΡΩΠΑΪΚΟ ΤΑΜΕΙΟ
ΠΕΡΙΦΕΡΕΙΑΚΗΣ ΑΝΑΠΤΥΞΗΣ

Υπουργείο Παιδείας και Θρησκευμάτων, Πολιτισμού και Αθλητισμού
ΓΓΕΤ – ΕΥΔΕ-ΕΤΑΚ

Ε. Π. Ανταγωνιστικότητα και Επιχειρηματικότητα (ΕΠΑΝ ΙΙ), ΠΕΠ Μακεδονίας – Θράκης, ΠΕΠ Κρήτης και Νήσων Αιγαίου, ΠΕΠ Θεσσαλίας – Στερεάς Ελλάδας – Ηπείρου, ΠΕΠ Αττικής

## References

Alexiadis, A., and Refanidis, I. 2012. Meeting the objectives of personal activity scheduling through post-optimization. First International Workshop on Search Strategies and Non-standard Objectives (SSNOWorkshop'12), in conjunction with CPAIOR-2012, Nantes, France.

Bank, J.; Cain, Z.; Shoham, Y.; Suen, C.; and Ariely, D. 2012. Turning personal calendars into scheduling assistants. In *Proceedings of the 2012 ACM annual conference extended abstracts on Human Factors in Computing Systems Extended Abstracts*, CHI EA '12, 2667–2672. New York, NY, USA: ACM.

Berry, P. M.; Gervasio, M.; Peintner, B.; and Yorke-Smith, N. 2011. Ptime: Personalized assistance for calendaring. *ACM Trans. Intell. Syst. Technol.* 2(4):40:1–40:22.

Coman, A., and Muñoz-Avila, H. 2011. Generating diverse plans using quantitative and qualitative plan distance metrics. In Burgard, W., and Roth, D., eds., *AAAI*. AAAI Press.

Dechter, R.; Flerova, N.; and Marinescu, R. 2012. Search algorithms for m best solutions for graphical models.

Freed, M.; Carbonell, J.; Gordon, G.; Hayes, J.; Myers, B.; Siewiorek, D.; Smith, S.; Steinfeld, A.; and Tomasic, A. 2008. Radar: a personal assistant that learns to reduce email overload. In *Proceedings of the 23rd national conference on Artificial intelligence - Volume 3*, AAAI'08, 1287–1293. AAAI Press.

Joslin, D., and Clements, D. P. 1999. Squeaky wheel optimization. *J. Artif. Intell. Res. (JAIR)* 10:353–373.

Kambhampati, S. 2007. Model-lite planning for the web age masses: The challenges of planning with incomplete and evolving domain models.

Myers, K.; Berry, P.; Blythe, J.; Conley, K.; Gervasio, M.; McGuinness, D.; Morley, D.; Pfeffer, A.; Pollack, M.; and Tambe, M. 2007. An intelligent personal assistant for task and time management.

Nguyen, T. A.; Do, M. B.; Gerevini, A.; Serina, I.; Srivastava, B.; and Kambhampati, S. 2012. Generating diverse plans to handle unknown and partially known user preferences. *Artif. Intell.* 190:1–31.

Refanidis, I., and Alexiadis, A. 2011. Deployment and evaluation of selfplanner, an automated individual task management system. *Computational Intelligence* 27(1):41–59.

Refanidis, I., and Yorke-Smith, N. 2010. A constraint-based approach to scheduling an individual's activities. *ACM TIST* 1(2):12.

Refanidis, I. 2007. Managing personal tasks with time constraints and preferences. In Boddy, M. S.; Fox, M.; and Thiébaux, S., eds., *ICAPS*, 272–279. AAAI.

Roberts, M.; Howe, A.; Ray, I.; and Urbanska, M. 2012. Using planning for a personalized security agent.

Selman, B.; Kautz, H.; and Cohen, B. 1995. Local search strategies for satisfiability testing. In *DIMACS SERIES IN DISCRETE MATHEMATICS AND THEORETICAL COMPUTER SCIENCE*, 521–532.

# Adapting Planetary Rover Plans via Action Modality Reconfiguration

**Enrico Scala, Roberto Micalizio, Pietro Torasso**
Dipartimento di Informatica - Universita' di Torino
C.so Svizzera 185, Torino - Italy
{scala,micalizio,torasso}@di.unito.it

**Abstract**

Robust execution of exploration mission plans has to deal with limited computational power on-board a planetary rover, and with limited rover's autonomy. Typically, these limitations prevent the rover to synthesize a new mission plan when contingencies arise.
The paper shows that when contingencies are deviations on the consumption of resources, robust execution can be achieved efficiently through action reconfiguration rather than replanning from scratch. The paper therefore introduces a novel representation of actions with modalities, and proposes an action reconfiguration module - ReCon - that detects the violation of mission resource constraints, and finds (if any) a new configuration of action modalities to resolve these violations.

## Introduction

The execution of a space exploration mission is a critical activity that has to take into account several challenges. A planetary rover, in fact, operates in an environment which is just partially observable and loosely predictable. As a consequence, the rover must have some form of autonomy in order to guarantee robust plan execution (i.e., reacting to unexpected contingencies). The rover's autonomy, however, is typically bounded both because of limitations of on-board computational power, and because the rover is not in general allowed to change the high level plan synthesized on Earth. Space missions therefore exemplify situations where contingencies occurs, but plan repair must be achieved through novel techniques trading-off rover's autonomy and the stability of the mission plan.

Robust plan execution has been tackled in two ways: on-line and off-line. On-line approaches, such as (Gerevini and Serina 2010; van der Krogt and de Weerdt 2005; Garrido, Guzman, and Onaindia 2010; Brenner and Nebel 2009), interleave plan execution and replanning: whenever unexpected contingencies cause the failure of an action, the plan execution is stopped and a new plan is synthesized as a result of a new planning phase. Off-line approaches, such as (Block, Wehowsky, and Williams 2006; Conrad, Shah, and Williams 2009), avoid replanning by anticipating, at planning time, the possible contingencies. The result of such a planning phase is a contingent plan that encodes choices between functionally equivalent sub-plans. At execution time, the plan executor is able to select a contingent plan according to the current contextual conditions. However, as for instance in the work of (Policella et al. 2009), the focus is mainly on the temporal dimension and they do not consider consumable and even continuous resources.

In this paper we propose a novel on-line methodology to achieve robust plan execution, which is explicitly devised to deal with unexpected deviations in the consumption of rover's resources. First, in line with the action-based approach *a-la* STRIPS (Fox and Long 2003) and differently from the constrained based planning (Fratini, Pecora, and Cesta 2008; Muscettola 1993), we model consumable resources as numeric fluents (introduced in PDDL 2.1 (Fox and Long 2003)). Then, we enrich the model of the rover's actions by expliciting a set of *execution modalities*. The basic idea is that the propositional effects of an action can be achieved under different configurations of the rover's devices. These configurations, however, may have a different impact on the consumption of the resources. An *execution modality* explicitly models the resource consumption profile when an action is carried out in a given rover's configuration. The integration of *execution modality* at the PDDL level allows a seamless integration between planning and execution.

Relying on this extension, we propose to handle exceptions arising at execution time as a reconfiguration of action modalities, rather than as a replanning problem. In particular, the paper proposes an adaptive plan execution strategy, i.e. ReCon; once (significant) deviations from the nominal trajectory are detected, ReCon intervenes by reconfiguring the modalities of the actions still to be performed with the purpose of restoring the validity of resource constraints.

After introducing a motivating example, we introduce the employed action model, enriched with the notion of execution modality. Then we introduce the ReCon strategy and an example showing how the system actually works in a exploration rover mission. Finally, an experimental section, which evaluates the competence and the efficiency of the strategy w.r.t. a traditional replanning from scratch.

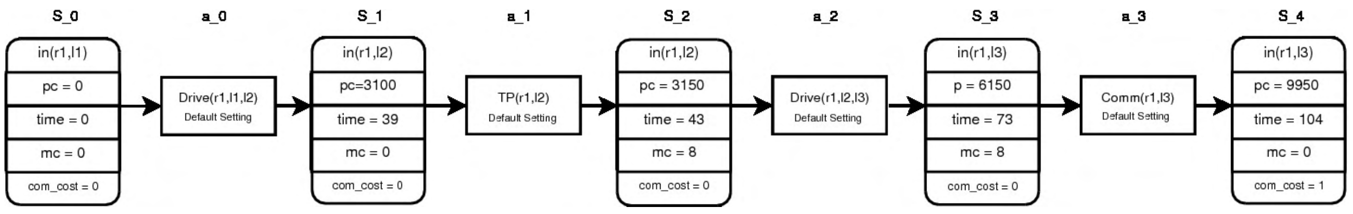


Figure 1: A simple mission plan.

## Motivating Example

Let us consider the simple exploration mission showed in Figure 1, involving *take picture, drive* and *communications* activities. This mission represents a feasible solution for a planning problem with goal: {`in(r1,l3), mem>=120, pwr>=0, time<=115`} ; that is, at the end of plan the rover must be located in `l3` (propositional fluent), the free memory must be (at least) 120 memory units, there must be a positive amount of power, and the mission must be completed within 115 secs.

The figure shows how the four actions (regular boxes) change the status of the rover over the time (rounded-corner boxes)[1]. Note that the status of a rover involves both propositional fluents, (e.g., `in(r1, l1)` meaning rover `r1` is in location `l1`); and numeric fluents: `memory` represents the amount of free memory, `power` is the amount of available power, `time` is the mission time given in seconds, and `com_cost` is an overall cost associated with communications.

The estimates about the rover's status are inferred by predicting, deterministically, the effects of the actions. In particular, the numeric fluents have been estimated by using a "default setting" (i.e., a standard modality) associated with each action.

Let us now assume that during the execution of the first drive action the rover has to travel across a rough terrain. Such an unexpected condition affects the drive as the rover is forced to slowdown[2], and as a consequence the drive action will take a longer time to be completed; the effects are propagated till the last snapshot, *s_4* where the goal constraint `time <= 115` will be no longer satisfied.

After detecting this inconsistency, approaches based on pure replanning step would compute a new plan achieving the goal by changing the original mission. For instance, some actions could be skipped in order to compensate the time lost during the first drive.

However, robotic systems as a planetary rover have typically different configurations of actions to be executed and each configuration can have a different impact on the mission progress. For instance the robotic systems described in (Calisi et al. 2008) and in (Micalizio, Scala, and Torasso 2011) can perform a drive action in fast or slow modes. Reliable transmission to the earth, for example, can be slow and cheap, or fast and expensive, depending on the devices actually used.

Our proposal is to explicitly represent such different configurations within the action models, and hence try to resolve an impasse via a reconfiguration of the actions still to be performed. Intuitively, our objective is to keep the high level plan structure unchanged, but to adjust the action modalities.

In the next section we will introduce the rover action model that explicitly expresses the set of *execution modality* at disposal.

## Modeling Rover's Actions

As we have seen in the previous section, a planetary rover can perform the same set of actions via different configurations of parameters or devices. To capture this aspect, this section introduces the rover action model adopted in this work. The model exploits (and extends) the numeric PDDL 2.1 action model ((Fox and Long 2003)), i.e. where the notion of numeric fluents has been proposed. In particular, we use the numeric fluents to model continuous and consumable resources.

The intuition of the extension is that, while actions differ each other in terms of qualitative effects (e.g. a drive action models how the position of the rover changes after the action application), the expected result of an action can actually be obtained in many different ways by appropriately configuring the rover's devices (e.g. the drive action can be performed with several engine configurations). Of course, different configurations have in general different resource profiles and it is therefore possible that the execution of an action in a given configuration would lead to a constraint violation, whereas the same action performed in another configuration would not. We call these alternative configurations *modalities* and we propose to capture the impact of a specific modality by modeling the use of specific configurations in terms of pre/post conditions on the numeric fluents involved, which becomes explicit in the action model definition.

The resulting model expresses the rover actions at two different levels of abstraction. The higher one is the qualitative level indicating "what" the action does. The lower one is the quantitative level expressing "how" the action achieves its effect.

By recalling our motivating example, Figure 2 shows the model of the drive action. The action template `drive (?r, ?l1, ?l2)` requires a rover `?r` to move from a location `?l1` to location `?l2. :modalities` introduces the set of modalities associated with a `drive`; in particular, we

[1]To simplify the picture, we show in the rover's status just a subset of the whole status variables

[2]In (Micalizio, Scala, and Torasso 2011) we have showed that the slowdown of the rover can be a consequence of a reactive supervisor, which operates as a continuous controller.

express for this action, three alternative modalities:
- `safe`: the rover moves slowly and far from obstacles; intuitively the action should spend more time but consuming less power
- `cruise`: the rover moves at its cruise speed and can go closer to obstacles;
- `agile`: the rover moves faster than `cruise`, consuming more power but requiring less time.

The `:precondition` and `:effect` fields list the applicability conditions and the effects, respectively, and are structured as follows: first a propositional formula encodes the condition under which the action is considered applicable; the second field (`:effect`) indicates the positive and the negative effects of the action. For each modality *m* in `:modalities` we have the amount of resources required (numeric precondition) or consumed/produced (numeric effect) by the action when performed under that specific modality *m*.

For instance, in the preconditions `(reachable l1, l2)` and `(in r1, l1)` are two propositional atoms required as preconditions for the application of the action. These two atoms must be satisfied independently of the modality actually used to perform the `drive` action. While the comparison `(safe: (>= (power ?r) (* (safe_cons ?r) (/ (distance ?l1 ?l2) (safe_speed ?r)))))` means that the modality `safe` can be selected when the rover's power is at least greater than a threshold given by evaluating the expression on the right side. Analogously, `(safe: (decrease (power ?r) (*(safe_cons ?r) (/ (distance ?l1 ?l2) (safe_speed ?r))))` describes in the effects how the rover's power is reduced after the execution of the drive action. More precisely, we have modeled the power consumption as a function depending on the duration of the drive action (computed considering distance and speed) and the average power consumption per time unit given a specific modality. For instance, in safe modality, the amount of power consumed depends on two parameters `(safe_cons ?r)` and `(safe_speed ?r)` which are the average consumption and the average speed for the safe modality, respectively, while `(distance ?l1 ?l2)` is the distance between the two locations `?l1` and `?l2`.

Finally, note that in the numeric effects of each modality, the model updates also the fluent `time` according to the selected modality. Also in this case, the duration of the action is estimated by a function associated with each possible action modality.

Analogously to the drive action we model modalities also for the Take Picture (TP) and the Communication (COMM). For TP we have the low (LR) and high (HR) resolution modalities which differ in the quality of the taken picture and the memory spent. Intuitively, the more the resolution is, the more the memory consumption will be. Whereas for the Communication we assume to have two different channels of transmissions: CH1 with low overall `comm_cost` and low bandwidth, and CH2 with high overall `comm_cost` but high bandwidth.

The selection of action modalities has to take into account

```
(:action drive
  :parameters ( ?r - robot ?l1 - site ?l2 - site)
  :modalities (safe,normal,agile)
  :precondition (and (in ?r ?l1) (road ?l1 ?l2)
  (safe: (>= (power ?r) (* (safe_cons ?r)
     (/ (distance ?l1 ?l2) (safe_speed ?r)))))
  (cruise: (>= (power ?r) (* (cruise_cons ?r)
           (/ (distance ?l1 ?l2) (cruise_speed ?r)))))
  (agile: (>= (power ?r) (* (agile_cons ?r)
           (/ (distance ?l1 ?l2) (agile_speed ?r)))))
  )
  :effect
  (and
    (in ?r ?l2) (not (in ?r ?l1))
    (safe: (decrease  (power ?r) (* (safe_cons ?r)
          (/ (distance ?l1 ?l2) (safe_speed ?r))))
         (increase  (time) (/ (distance ?l1 ?l2)) (safe_speed ?r)))
         (increase  (powerC ?r) (* (safe_cons ?r)
                         (/ (distance ?l1 ?l2) (safe_speed ?r))))
    (cruise: (decrease  (power ?r) (* (cruise_cons ?r)
                         (/ (distance ?l1 ?l2) (cruise_speed ?r))))
         (increase  (time) (/ (distance ?l1 ?l2)) (cruise_speed ?r))
         (increase  (powerC ?r) (* (cruise_cons ?r)
                         (/ (distance ?l1 ?l2) (cruise_speed ?r)))))
    (agile: (decrease  (power ?r) (* (agile_cons ?r)
                         (/ (distance ?l1 ?l2) (agile_speed ?r))))
         (increase  (time) (/ (distance ?l1 ?l2)) (agile_speed ?r))
         (increase  (powerC ?r) (* (agile_cons ?r)
                         (/ (distance ?l1 ?l2) (agile_speed ?r)))))
  )
```

Figure 2: The augmented model of a `drive` action.

that complex dependencies among resources could exist. For instance, even if a high resolution TP takes the same time as a low resolution TP, the selection has a big impact on the amount of time spent globally, too. As a matter of facts, as long as the amount of stored information increases, the time spent by a (possible) successive COMM grows up accordingly, which means that also the global mission horizon will be revised.

Given the rover's actions defined so far, a rover mission plan is a total ordered set of fully instantiated rover's action template[3]. Given a particular rover's state S and a given set of propositional goals G to be reached (including constraints on the amount of resources), the mission plan is valid iff it achieve G from S.

**Executing the mission plan.** As we have seen in the previous section, the rover's mission can be threatened many times by unexpected contingencies; so the validity of the mission can be easily compromised during its actual execution.

Nevertheless, when the detected unexpected contingency at execution time just invalidates the resource consumption expectations, even if the current modality allocation would not be consistent with the constraints involved in the plan and in the goal, there could be "other" allocations of modalities still feasible. By exploiting this intuition, the next section introduces an adaptive execution technique which, instead of abandoning the mission being executed, tries first to repair the flaws via a reconfiguration of the action modalities. The reconfiguration considers all those actions still to be executed.

Given a plan P, to indicate when a plan is just *resource inconsistent*, we will use the predicate *res_incon* over P, i.e.

[3]The plan can be also generated automatically by exploiting a numeric planner system, properly modified to handle actions with modalities. In our tests we used Metric-FF (Hoffmann 2003)

we will say *res_incon(P)*. Otherwise we will say that the plan is valid or structurally invalid. This latter case happens when, given the current plan formulation, at least an action in the plan is not propositional applicable, or there is at least a missing goal.

## ReCon: adaptive plan execution

In this section we describe how the plan adaptation process is actually carried on by exploiting a Constraint Satisfaction Problem representation. The main strategy implemented, namely ReCon, is a continual planning agent ((Brenner and Nebel 2009),(desJardins et al. 1999)), extended to deal with the rover actions model presented in the previous section. In order to handle with the CSP representation, ReCon exploits two further sub-modules: **Update** by means of which new observations are asserted within the CSP representation, and **Adapt** which has the task of making the mission execution adaptive to the incoming situation.

### The Continual Planning Loop

Algorithm 1 shows the main steps required to execute and (just in case) adapt the plan being executed. The algorithm takes in input the initial rover's state $S_0$, the mission goal $Goal$, and the plan $P$ expressed as discussed in the previous section. Note that each action has to have a particular modality of execution instantiated. The algorithm returns $Success$ when the execution of the whole mission plan achieves the goal; $Failure$ otherwise. In this case, a failure means that there is no way to adapt the current plan in order to reach the goal satisfying mission constraints. To recover from this failure, a replanning step altering the structure of the plan should be invoked, but this step requires the intervention of the ground control station on Earth.

The first step of the algorithm is to build a $CSPModel$ representing the mission plan (line 1). Due to lack of space, we cannot present this step in details; our approach, however, inherits the main steps by Lopex et al. in (Lopez and Bacchus 2003) in which the planning problem is addressed as a CSP. As a difference w.r.t. the classical planning, the encoding exploited by our approach needs to store variables for the modalities to be chosen, and variables for the numeric fluents involved in the plan. Numeric fluents variables are replicated as many steps in the plan. The purpose is to capture all the possible evolutions of resources profiles given the modalities that will be selected. The constraints oblige the selection of the modality to be consistent with the resource belonging to the previous and successive time step. Moreover, further constraints allow only reconfigurations consistent with the current observation acquired (which at start-up corresponds to the initial state), and the goals/requirement of the mission.

Once the $CSPModel$ has been built, the algorithm loops over the execution of the plan. Each iteration corresponds to the execution of the $i$-th action in the plan. At the end of the action execution the process verifies the current observation $obs_{i+1}$ with the rest of the mission to be executed. In case the plan is structurally invalid (some propositional conditions are not satisfied or the goal cannot be reached) ReCon stops the plan execution and returns a failure; i.e., a replanning procedure is required.

Otherwise we can have two other situations. First, there have been no consistent deviations from the nominal predictions therefore the execution can proceed with the remaining part of the plan. Second the plan is just resource inconsistent ($res\_incon(P)$, line 10). In this latter case, ReCon has to adapt the current plan by finding an alternative assignments to action modalities that satisfies the numeric constraints (line 11). If the adaptation has success, a new non-empty plan $newP$ is returned and substituted to the old one. This new plan is actually the old plan, but with a different allocations of action modalities. Otherwise, the plan cannot be adapted and a failure is returned; in this case, the plan execution is stopped and a new planning phase is needed.

**Algorithm 1: ReCon**

```
   Input: $S_0$, $Goal$, $P$
   Output: Success or Failure
1  $CSPModel = Init(S_0, Goal, P)$ ;
2  $i = 0$;
3  while ¬ P is completed do
4      execute($a_i$, $curMod(a_i)$);
5      $obs_{i+1}$ = observe();
6      if P is structurally invalid w.r.t. $obs_{i+1}$ and Goal then
7          return $Failure$
8      else
9          Update($CSPModel$,$a_i$,$num(obs_{i+1})$);
10         if $res\_incon(P)$ then
11             $newP$ = Adapt($CSPModel$,$i$,$Goal$,$P$);
12             if $newP \neq \emptyset$ then
13                 $P = newP$
14             else
15                 return $Failure$
16 return $Success$
```

### Update

The **Update** step is sketched in Algorithm 2. The algorithm takes in input the CSP model to update, the last performed action $a_i$, and the set $NObs$ of observations about numeric fluents. The algorithm starts by asserting within the model that the $i$-th action has been performed; see lines 1 and 2 in which variable $mod_i$ is constrained to assume the special value $exec$. In particular, a first role of the $exec$ value is to prevent the adaptation process to change the modality of an action that has already been performed, as we will see in the following section. Moreover, $exec$ allows also the acquisition of observations even when the observed values are completely unexpected. In fact, by assigning the modality of action $a_i$ to $exec$, we relax all the constraints over the numeric variables at step $i + 1$-th (which encode the action effects). This is done in lines 3-5 in which we iterate over the numeric fluents $N^j$ mentioned in the effects of action $a_i$, and assign to the corresponding variable at $i+1$-th step the value observed in $NObs$. On the other hand, all the numeric fluents that are not mentioned in the effects of action $a_i$ do not change, so the corresponding variables at step $i+1$ assume the same values as in the previous $i$-th step (lines 6-8). The idea of the Update is to make the CSP aware of the current new observations and the modalities already executed. In this way, a reconfiguration task does not need to rebuild the structure completely from scratch.

**Algorithm 2: Update**

```
   Input: $CSPModel$, $a_i$,$NObs$
   Output: modified $CSPModel$
1  delConstraint($CSPModel$,$mod_i$=curMod($a_i$)) ;
2  addConstraint($CSPModel$,$mod_i$=$exec$) ;
3  foreach $N^j \in affected(a_i)$ do
4      addConstraint($CSPModel$,
5      ($mod_i$=$exec$)→ $N^j_{i+1}$=get($NObs$,$N^j_{i+1}$))
6  foreach $N^j \in \neg affected(a_i)$ do
7      addConstraint($CSPModel$,
8      ($mod_i$=$exec$)→ $N^j_{i+1}$=$N^j_i$)
```

## Adapt

The **Adapt** module, shown in Algorithm 3, takes in input the CSP model, the index $i$ of the last action performed by the rover, the mission goal, and the plan $P$; the algorithm returns a new adapted plan, if it exists, or an empty plan when no solution exists.

The algorithm starts by removing from $CSPModel$ the constraints on the modalities of actions still to be performed; i.e., each variable $mod_k$ with $k$ greater than $i$ is no longer constrained ($a_i$ is the last performed action and its modality is set to $exec$) (lines 1-3). This step is essential since the current $CSPModel$ is inconsistent; that is, the current assignment of modalities does not satisfies the global constraints. By removing these constraints, we allow the CSP solver to search in space of possible assignments to modality variables (i.e., the actual decisional variables, since the numeric fluents are just side effects of the modality selection), and find an alternative assignment that satisfies the global constraints (line 4). If the solver returns an empty solution, then there is no way to adapt the current plan and **Adapt** returns no solution. Otherwise (lines 7-12), at least a solution has been found. In this last case, a new assignment of modalities to the variables $mod_k$ ($k : i+1..|P|$) is extracted from the solution, and this assignment is returned to the ReCon algorithm as a new plan $newP$ such that the actions are the same as in $P$, but the modality labels associated with the actions $a_{i+1}, .., a_{|P|}$ are different.

Note that, in order to keep updated the CSP model for future adaptations, the returned assignment of modalities is also asserted in $CSPModel$; see lines 9-9.

**Algorithm 3: Adapt**

```
    Input: $CSPModel$, $i$,$Goal$,$P$
    Output: a new plan, if any
1   for $k$=$i$+1 to $|P|$ do
2       delConstraint($CSPModel$,
3               $mod_k$=currentMod($a_k$));
4   $Solution$ = solve($CSPModel$);
5   if $Solution$ = null then
6       return $\emptyset$
7   else
8       $newP$=extractModalitiesVar($Solution$);
9       for $k$=$i$+1 to $|newP|$ do
10          addConstraint($CSPModel$,
11            $mod_i$=curMod($newP[i]$));
12      return $newP$
```

## Running the Mission Rover Example

Let us consider again the example in Figure 1, and let us see how RoCon manages its execution. First of all, the plan model must be enriched with the execution modalities as previously explained; Figure 3 (top) shows the initial configuration of action modalities: the drive actions have `cruise` modalities, the take picture (TP) has `LR` (low resolution) modality, and the communication (`Comm`) uses the low bandwidth channel (`CH1`). This is the enriched plan ReCon receives in input.

Now, let us assume that the actual execution of the first drive action takes a longer time than expected, 47s instead of 38s, and consumes more power, 3775 Joule instead of 3100 Joule. While the discrepancy on power is not a big issue as it will not cause a failure, the discrepancy on time will cause the violation of the constraint `time <=115`; in fact, performing the subsequent actions in their initial modalities would require 120 seconds. In other words, the assignment of modalities to the subsequent actions does not satisfies the mission constraints. This situation is detected by ReCon that intervenes and, by means of the **Adapt** algorithm discussed above, tries to find an alternative configuration of modalities.

Let us assume that communication cost is constrained; that is, the mission goal includes the constraint `com_cost = 1`; this prevents ReCon from using the fast communication channel. In order to gain some time, ReCon can just switch the modality of the second drive action from `cruise` to `agile`, which allows the rover to move faster. Figure 3 (bottom), shows the reconfigured plan with the estimation on the resource and time consumption. Note that the new configuration has an impact not only on `time`, but also on `power`; such an impact, however, does not represent a violation of the global constraints (the constraint on power is not so strict), and hence the proposed (re)configuration is a solution.

Of course, we assume that mission constraints leave ReCon some room to repair resource inconsistent situations. For instance, if the constraint on the power were very tight, the drive action performed in *agile* modality would not be

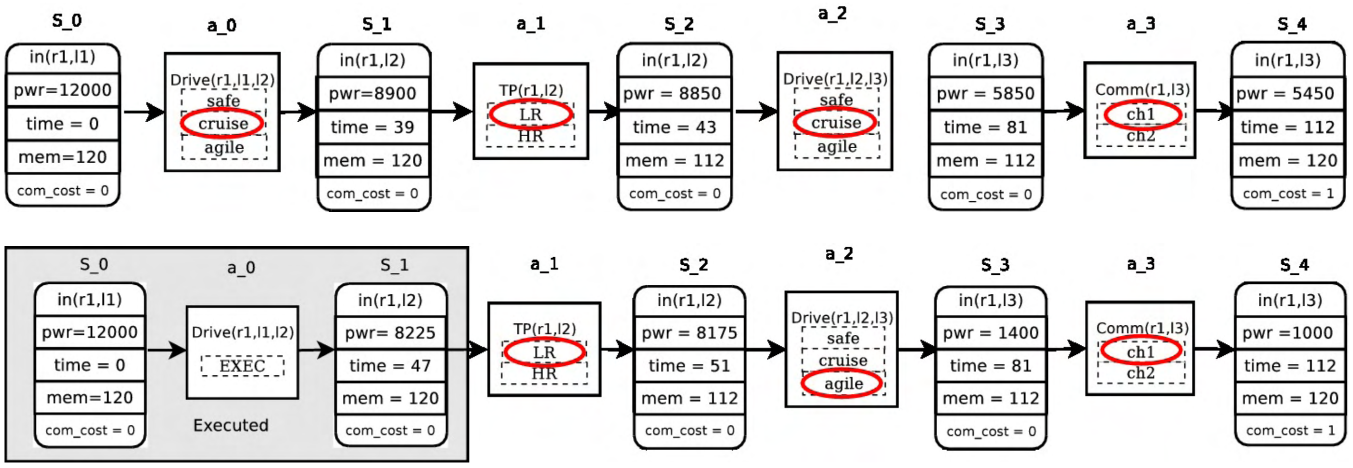


Figure 3: The initial configuration of modalities (above), and the reconfigured plan (below).

applicable, and hence no reconfiguration would be possible.

## Experimental Validation

To assess the effectiveness of our proposal, we evaluated two main parameters: (1) the computational cost of reconfiguration, and (2) the competence of ReCon, that is, the ability of completing a mission.

To this aim, we have compared ReCon with two alternative strategies: REPLAN and NoRep. Whenever the plan becomes resources inconsistent, REPLAN stops the execution of the plan and synthesizes a new plan from scratch. Conversely, NoRep just stops the plan execution as soon as it is no longer valid. Note that although REPLAN is not allowed in our rover scenario, we used it as a benchmark to better assess the contribution of ReCon.

We have implemented ReCon in Java 1.7; the Choco CSP solver (version 2.1.3) [4] has been used in the **Adapt** algorithm to find an alternative configuration. Whereas REPLAN invokes Metric-FF (Hoffmann 2003) by converting the rover actions with modalities in PDDL 2.1 actions, as explained in (Scala 2013). Each repair problem has a time computation threshold set to 60 secs.

Our test set consists of 105 plans; each plan involves from 9 up to 30 actions (i.e., drives, take pictures, and communications), it is fully instantiated (a modality has been assigned to each action), and feasible since all the goal constraints are satisfied when the plan is submitted for the execution.

To simulate unexpected deviations in the consumption of the resources, we have run[5] each test in thirteen different settings. In each of these settings we have noised the amount of resources consumed by the actions. In particular, in setting 1, an action consumes 10% more than expected at planning time. In setting 2, the noise was increased to 15%, and so on until in setting 13 where the noise was set to 70%, i.e. an action consumes 70% more resources than initially predicted.

Figure 4 reports the competence - measured as the percentage of performed actions in the plan - of the two strategies ReCon and REPLAN, in the thirteen settings we have considered. As expected, the competence of both solutions decreases as long as the amount of noise increases. Note that ReCon is more competent than REPLAN. In fact, even though REPLAN can modify the plan, and hence it can solve repair problems that ReCon cannot, REPLAN is less competent than ReCon due to the time limit of 60 secs. In particular we can observe an average gap of 20% between the percentage of plan completed by ReCon and REPLAN.

Figure 5 shows the computational cost, on average, of the two strategies. As for the competence evaluation, it is easy to see that ReCon outperforms REPLAN. In fact, even for the worst case (when the noise is set to be 70%), ReCon is extremely efficient, indeed it takes, on average, just 356 msec. Whereas, even for the cases with few noise, REPLAN takes about 5 secs of cpu-time till the 20 secs employed for the worst cases. For each case considered, the time for the repair corresponds to the sum of all the adaptation (reconfiguration or replanning) performed untill the end of the mission.

Finally, in Figure 6, we show the number of invocations to ReCon and REPLAN. It must be noticed that in the first ten noise settings (i.e., noise from 10% to 55%), REPLAN is activated, on average, more often than Recon. However, for the last three noise settings (i.e., noise from 60% to 70%) ReCon is invoked slightly more times than REPLAN. This happens because, as long as the plan execution process goes on, the constraints becomes more and more tight, causing the detection mechanism to be invoked more frequently. Differently, each invocation of REPLAN generates a completely new plan; therefore the plan execution till the end is not directly related to the previous plan execution problem. This is the reason why REPLAN almost preserves the same amount of invocations throughout the cases we have tested.

[4]The Choco Solver implements the state of the art algorithms for constraint programing and has already been used in space applications, see (Cesta and Fratini 2009). Choco can be downloaded at http://www.emn.fr/z-info/choco-solver/.

[5]Experiments have run on a 2.53GHz Intel(R) Core(TM)2 Duo processor with 4 GB.

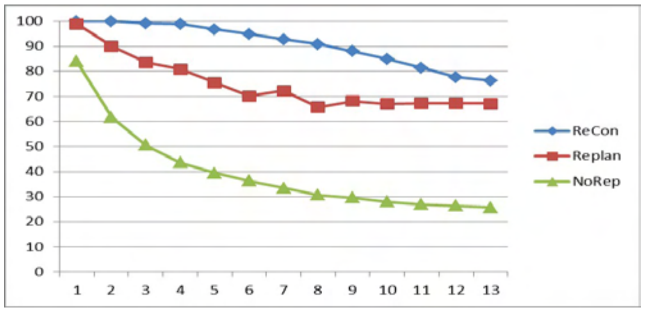


Figure 4: Competence: Percentage of performed actions

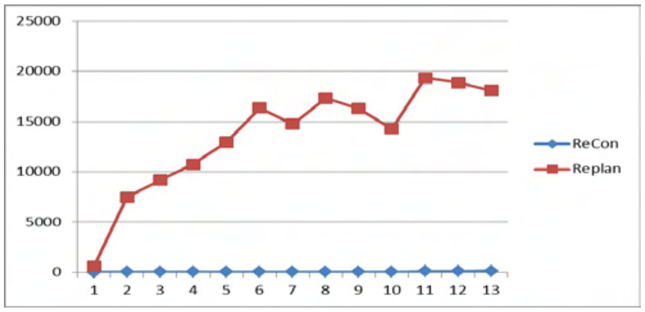


Figure 5: CPU time

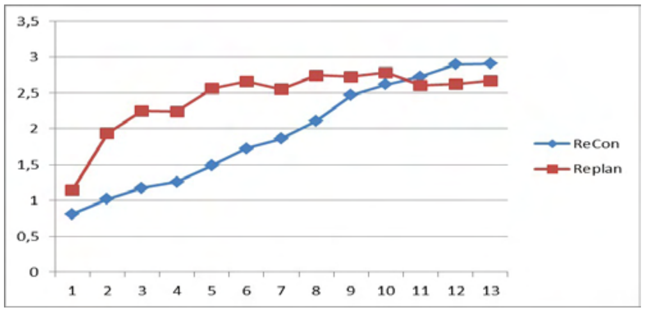


Figure 6: Average Number of Reconfiguration

## Conclusions

We have proposed in this paper a novel approach to the problem of robust plan execution. Rather than recovering from plan failures via a re-planning step (see e.g., (Gerevini and Serina 2010; van der Krogt and de Weerdt 2005; Garrido, Guzman, and Onaindia 2010)), we have proposed a methodology, called ReCon, based on the re-configuration of the plan actions. ReCon is justified in all those scenarios where a pure replanning approach is unfeasible. This is the case, for instance, of a planetary rover performing a space exploration mission. Albeit a rover must exhibit some form of autonomy, its autonomy is often bounded by two main factors: (1) the on-board computational power is not always sufficient to handle mission recovery problems, and (2) the rover cannot in general deviate from the given mission plan without the approval from the ground control station.

ReCon presents many advantages w.r.t. re-planning. First of all, reconfiguring plan actions is computationally cheaper than synthesizing a new plan from scratch, as the experiments have demonstrated. Moreover, ReCon leaves the high-level structure of the plan (i.e., the sequence of mission tasks) unchanged, but endows the rover with an appropriate level of autonomy for handling unexpected contingencies. ReCon can be considered as a complementary repair strategy to other works in the context of autonomy for space as those in (Chien et al. 2012); as matter of facts, ReCon explores a different dimension of the repair problem, which is based on an action-centered planning representation rather than on a timeline based perspective ((Fratini, Pecora, and Cesta 2008)).

The approach we have presented can be improved in a number of ways. A first important enhancement is the search for an optimal solution. In the current version, in fact, ReCon just finds one possible configuration that satisfies the global constraints. In general, one could be interested in finding the best configuration that optimizes a given objective function. Reasonably, the objective function could take into account the number of changes to action modalities; for instance, in some cases it is desirable to change the configuration as little as possible. Of course, the search for an optimal configuration is justified when the global constraints are not strict, and several alternative solutions are possible.

## References

Block, S. A.; Wehowsky, A. F.; and Williams, B. C. 2006. Robust execution of contingent, temporally flexible plans. In *AAAI 2006: 802-808*.

Brenner, M., and Nebel, B. 2009. Continual planning and acting in dynamic multiagent environments. *Journal of Autonomous Agents and Multiagent Systems* 19(3):297–331.

Calisi, D.; Iocchi, L.; Nardi, D.; Scalzo, C.; and Ziparo, V. A. 2008. Context-based design of robotic systems. *Robotics and Autonomous Systems (RAS)* 56(11):992–1003.

Cesta, A., and Fratini, S. 2009. The timeline representation framework as a planning and scheduling software development environment. In *The 28th Workshop of the UK PLANNING AND SCHEDULING*.

Chien, S.; Johnston, M.; Frank, J.; Giuliano, M.; Kavelaars, A.; Lenzen, C.; and Policella, N. 2012. A generalized timeline representation, services, and interface for automating space mission operations. Technical Report JPL TRS 1992+, Ames Research Center; Jet Propulsion Laboratory.

Conrad, P. R.; Shah, J. A.; and Williams, B. C. 2009. Flexible execution of plans with choice. In *ICAPS'09*, 74–81.

desJardins, M.; Durfee, E. H.; Jr., C. L. O.; and Wolverton, M. 1999. A survey of research in distributed, continual planning. *AI Magazine* 20(4):13–22.

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

Fratini, S.; Pecora, F.; and Cesta, A. 2008. Unifying planning and scheduling as timelines in a component-based perspective. *Archives of Control Sciences* 18(2):231–271.

Garrido, A.; Guzman, C.; and Onaindia, E. 2010. Anytime plan-adaptation for continuous planning. In *PlanSIG'10*.

Gerevini, A., and Serina, I. 2010. Efficient plan adaptation through replanning windows and heuristic goals. *Fundamenta Informaticae* 102(3-4):287–323.

Hoffmann, J. 2003. The metric-ff planning system: Translating ”ignoring delete lists” to numeric state variables. *JAIR* 20:291–341.

Lopez, A., and Bacchus, F. 2003. Generalizing graphplan by formulating planning as a csp. In *Proceedings of IJCAI'03*, 954–960.

Micalizio, R.; Scala, E.; and Torasso, P. 2011. Intelligent supervision for robust plan execution. In *LNCS 6954 of AI*IA*, 151–163.

Muscettola, N. 1993. Hsts: Integrating planning and scheduling. Technical Report CMU-RI-TR-93-05, Robotics Institute, Pittsburgh, PA.

Policella, N.; Cesta, A.; Oddi, A.; and Smith, S. 2009. Solve-and-robustify. *Journal of Scheduling* 12:299–314. 10.1007/s10951-008-0091-7.

Scala, E. 2013. *Reconfiguration and Replanning for robust Execution of Plans Involving Continous and Consumable Resources*. Ph.D. Dissertation, Department of Computer Science - Universita' di Torino.

van der Krogt, R., and de Weerdt, M. 2005. Plan repair as an extension of planning. In *ICAPS'05*, 161–170.

# Designing quiet rotorcraft landing trajectories with probabilistic road maps

**Robert A Morris,**
NASA Ames, USA
robert.a.morris@nasa.gov

**Michele Donini**
University of Padova, Italy
mdonini@math.unipd.it

**K. Brent Venable**
Tulane University/ IHMC, USA
kvenabl@tulane.edu

**Matthew Johnson**
IHMC, USA
mjohnson@ihmc.us

**Abstract**

Next generation commercial air transportation will likely see an increased use of rotorcraft, including helicopter and tilt-rotors. Two advantages of rotorcraft with respect to fixed wing aircraft are the optimized aerodynamic levitation and the possibility of takeoff and landing without a runway, which minimizes their interference with fixed wing traffic. A recurring obstacle to rotorcraft integration is the ground noise they generate, particularly near residential areas and hospitals. The problem of rotorcraft ground noise can be partially addressed by designing new flight profiles, in particular landing trajectories, that have a lower impact in the noise produced by a rotorcraft. In this paper we approach this problem computationally by modeling a high dimensional environment able to capture many parameters of landing trajectories that contribute to ground noise, and using probabilistic road maps (PRMs), coupled with a robust ground noise simulation system, for identifying noise-minimal approach trajectories.

## Introduction

Redesigning rotorcraft trajectories to optimize for ground noise is part of a set of improvements in rotorcraft operations that will facilitate their increased used for commercial transportation. Rotorcraft are of particular interest due to their runway independence which allows them to operate at an airport without directly interfering with major air carrier and commuter aircraft operations. The main concern which has prevented a more extensive use of rotorcraft is regarding the impact of noise on the communities surrounding the transportation facilities. However, steps can be taken to make flight trajectories less noisy through the careful redesign of approach and landing procedures.

This has lead to recent efforts (Morris et al. 2012; Atkins and Xue 2004) that aim at applying automated reasoning techniques to design flight profiles that may differ from standard pilot practice but reduce the accumulated ground noise while enforcing constraints related to safety and comfort of passengers. In most cases the problem of designing low noise flight profiles can be viewed as a trajectory optimization problem (LaValle 2006), informally consisting of a set of states, a vector of control decisions, a start and goal state, a cost function, and a set of constraints. Recent methods for solving trajectory optimization problems address the challenges of high-dimensional, non-linear systems by using randomized path-planning methods. Examples of recent trends include Rapidly expanding Random Trees (RRT) (P. Cheng and LaValle 2001) and Probabilistic Road Maps (PRMs) (Kavraki et al. August 1996).

This paper formulates a trajectory noise optimization problem and applies Probabilistic Road Maps to find low ground noise trajectories. Our system couples the power of path planning search with a state of the art rotorcraft noise simulator to produce more quiet landing trajectories. The strength of our results lies especially in the capability to produce very general trajectories and to allow for the embedding of obstacles, where obstacles could include restricted fixed-wing airspace or noise-sensitive areas such as hospitals. Moreover, the PRM approach appears ideal in terms of the deployment of our system on board of rotorcrafts. In this respect we envision making precomputed roadmaps for different landing sites available for upload and thus to limit the online effort to computing the noise-minimal trajectory given the starting point of the approach.

The remainder of this paper is structured as follows. In the background section we provide fundamental notions concerning rotorcraft noise, noise simulation and PRMs. We then describe our PRM implementation and we discuss experimental results which include a comparison with a local search approach. We conclude by suggesting some interesting future directions.

## Background

**Rotorcraft Noise and Noise Simulation** Helicopter noise sources include the main rotor, the tail rotor, the engine(s), and the drive systems. The most noticeable acoustical property of helicopters is referred to as BVI (Blade Vortex Interaction) noise. This impulsive noise occurs during high-speed forward flight as a result of blade thickness and compressible flow on the advancing blade. A common noise measure is the *Sound Exposure Level (SEL)*. *SEL* provides a comprehensive way to describe noise events for use in modeling and comparing noise environments. The average SEL value over the ground plane is called the SEL average ($SEL_{av}$). The equation for $SEL_{av}$ is

$$SEL_{av} = 10log_{10}\Sigma_n(\Sigma_i 10^{SPL_{dB,i,n}/10}\Delta t_{i,n}/T_0)\Delta A_n/A_0 \quad [1]$$

Here, $n$ ranges over the locations on the ground plane and $i$ refers to path elements. $SPL_{db,i,n}$ refers to the *Sound Pressure Level* in decibels for a location and a path element, and the $\Delta$s are elemental ground- or trajectory elements. $A_0$ refers to the area of the ground plane and $T_0$ is a reference interval of one second.

One challenge in performing a systematic study of approach trajectories for optimization is the cost of verifying results. The most accurate means of verification is through field tests, but these are too costly and time-consuming to perform on a casual basis. Fortunately, there are a number of robust noise models that allow for the evaluation of trajectories through simulation. One such modeling tool is the Rotorcraft Noise Model (RNM) (RNM 2007), a simulation program that predicts how the sound of a rotorcraft propagates through the atmosphere and accumulates on the ground. The core of the RNM method is a database of vehicle source noises defined as *sound spheres*. Spheres are obtained through measured test data or through models. The spheres allow for a representation of the 3D noise directivity patterns associated with the operating rotorcraft. A sphere is associated with one noise source and one flight condition (a value for flight path angle and airspeed). Each sphere represents constant airspeed conditions for a given flight path angle. The sound source properties are extracted from the sphere database using a linear interpolation of both required speed and flight path angle.

As inputs to RNM, a user specifies a flight path (in terms of acceleration and flight path angle). RNM outputs a time history and the effective SEL (or similar metrics) at any location along the path. The result is usually displayed as a contour plot (Figure 1) over a ground plane. Each color corresponds to a dB level (redder and lighter colors noisier).

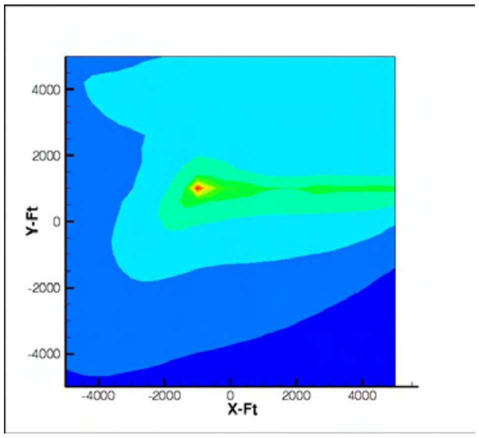


Figure 1: A Noise Contour Plot over a ground plane.

In (Morris et al. 2012) a method for aggregating the information in the contour plot into a scalar value using a *Binning Heuristic function* ($Bin$) was proposed. Given trajectory $t$, and a SEL value for each grid point $(x, y)$, denoted $SEL(t, x, y)$, a sequence of decreasing ranges, $\langle r_1, r_2, \ldots, r_n \rangle$ partitioning the SEL values of the grid points is defined. Moreover, given $S_i(t) = \{(x, y)|SEL(t, x, y) \in r_i\}$, that is, the set of grid values within the range $r_i$, a vector $b(t) = \langle b_1(t), b_2(t), \ldots, b_n(t) \rangle$, where $b_i(t) = |S_i(t)|$, represents the number of grid points with an associate SEL value within each range. The bin-score of solution $t$ can then be defined as $Bin(t) = \Sigma_{i=1\ldots n} w_i b_i(t)$ where $w_i$ is the weight associated to the $i$-th bin, $w_i > w_{i+1}$ and $\Sigma_{i=1,\ldots,n} w_i = 1$. Thus a solution that assigns lower levels of noise to larger regions of the grid is to be preferred. Weights can be tuned in various ways to penalize the presence of noisy regions in the grid.

**Probabilistic Road Maps** The trajectory planning problem is defined as the simultaneous planning of path and velocity for robotic systems (LaValle 2006). The search space is described in terms of a configuration space $\mathcal{C}$, capturing the significant parameters of the problem. $\mathcal{C}$ is divided into *free space* $\mathcal{C}_{free}$ and *obstacle space* $\mathcal{C}_{obj}$ where the latter represents the forbidden regions of the space. A planning instance is defined by the pair $(c_s, c_f)$ of an initial and a final configuration. A feasible solution for planning instance $(c_s, c_f)$ is a sequence of configurations in $\mathcal{C}$, $(c_s, ..., c_i, .., c_f)$ such that each configuration is in $\mathcal{C}_{free}$.

Sampling-based path planning techniques emerged out of the need to address the complexity of realistic path-planning problems. The main idea behind sampling-based approaches is to generate and organize a sequence of samples from $\mathcal{C}_{free}$ into a graph, where the edges are labeled with a cost, usually a metric on $\mathcal{C}$. The graph is then traversed to find a path that solves the planning instance.

Probabilistic Road Maps (PRMs) (Kavraki et al. August 1996) is a sampling-based method that consists of a road map construction phase and a user-defined query phase in which the roadmap is consulted for planning purposes. The basic PRM algorithm is easy to implement and performs well on a variety of problems. In the next section we describe how PRMs can be used to find noise-minimal landing trajectories for rotorcrafts.

## PRM for rotorcraft noise minimization

The *configuration space* for our rotorcraft trajectory optimization problem consists of state variables $X, Y, Z, V$ (location $(X, Y)$, altitude $(Z)$ and forward speed $(V)$). For now we ignore non-translational motion (i.e., roll, pitch and yaw) in the configuration model. They are not strong determinants of noise in the simulation, so their inclusion is not important. Let $(0, 0, 0, 0)$ represent the landing point and configuration space $\mathcal{C}$ be defined around the landing point by the following limit constraints:

$$\begin{cases} 0ft \leq z \leq 1500ft \\ -8000ft \leq x \leq 8000ft \\ -8000ft \leq y \leq 8000ft \\ 0ft/s \leq v \leq 140ft/s. \end{cases} \quad [2]$$

### Sampling the configuration space

The first step of the PRM approach is to densely sample the configuration space. The sampling strategy used here is via Halton sequences. A Halton sequence is a deterministic low discrepancy sequence used to generate points in a space which has been shown to be sufficiently random for many purposes (Choset et al. 2005). An example of Halton sequence sampling for a 2-dimensional space is shown in Figure 2. We remark that in our case we sample a 4-dimensional configuration space.

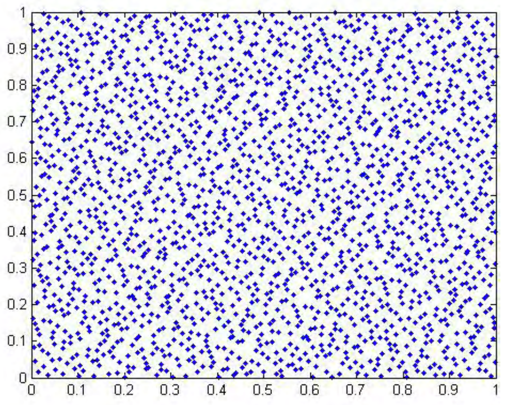


Figure 2: 2000 Halton sequence points in $[0, 1] \times [0, 1]$.

## Building the roadmap

Each point in the configuration space sampled in the previous step can be viewed as a node in a graph. In order to build the roadmap we must decide which points to connect with edges and how to label the connections with noise-dependent costs.

**Defining edges.** We describe a feasible connection between two sampled points, say $c_i$ and $c_j$, as an edge $c_i \to c_j$ and via constraints defined on their physical distance, their airspeed difference, their altitude difference and the obstacle avoidance constraints.

First, a *constraint on distance* is imposed via the Euclidean distance function $dist(n_i, n_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2}$ in the $3D$ space and by choosing a range $[dist_{min}, dist_{max}] \subset ]0, +\infty[$ of allowed distances. In our scenario we have set $[dist_{min}, dist_{max}] = [100ft, 3000ft]$. We note that the lower bound has been chosen sufficiently large in order to allow for sufficient time pilots to implement the changes in direction, speed or descent.

Second, we note that decelerations exceeding 0.1g (i.e. $201ft/sec^2$) are considered uncomfortable for the average passenger. This induces a *constraint on deceleration* $\Delta v_{ij} = |v_i - v_j|$ among two points: $\Delta v_{ij} \leq \sqrt{v_i^2 + 2 \cdot 0.1 \cdot dist_{ij}} - v_i$.

Third, passenger discomfort can be caused by too steep descents and, in particular, by those exceeding a corresponding descent angle of roughly $12^o$. This induces a *constraint on altitude difference* $\Delta z_{ij} = |z_i - z_j|$, between two points: $\Delta z_{ij} > \tan(12^o) \cdot dist_{ij}$.

The final constraint enables the enforcement of *obstacle avoidance*. In this domain, obstacles might be residential areas or hospitals, or restricted airspace, such as the space used for fixed-wing traffic as well as limitations related to the specific rotorcraft model. We employ standard methods such as those described in (LaValle 2006), in which a set of logical predicates $f_i : \mathcal{C} \to [0, 1] \subset \mathbb{R}$ define whether a configuration is or is not in the obstacle space. Given logical predicates $f_1, ..., f_m$ describing $m$ obstacles in $\mathcal{C}$, $Obs_i = \{p \in \mathcal{C} | f_i(p) = 0\} \quad \forall i = 1, ..., m$, we have that $\mathcal{C}_{free} = \mathcal{C} \setminus \bigcup_{i=1}^{m} Obs_i$. A real value $\epsilon > 0$, is derived empirically to define the sensitivity (constraint relaxation) of the obstacle, and the following approximation describes flyable points:

$$\forall p \in \mathcal{C} \quad p \in \mathcal{C}_{free} \iff f_i(p) > \epsilon \quad \forall i = 1, ..., m.$$

For example, the sensitivity could be used to specify separation distance between a point of interest (like a hospital) and any trajectory. With the obstacles $f_i$ defined, the collision detection algorithm subdivides each path edge $c_i \to c_j$ generated during roadmap construction to determine whether there is collision. The finer the subdivision the smaller the error in ensuring obstacle avoidance.

**Edge labeling** Computing a noise cost for an edge in the road map involves an invocation of the RNM noise simulator. In particular, for each feasible edge, we consider an area on which to compute the SEL contour plot that is large enough to allow for at least $2000ft$ around the projection on the ground of each edge point. The density of grid points in which this area is partitioned is defined by a grid distance of $500ft$. We then run RNM and obtain the corresponding SEL contour plot and aggregate it into a $Bin$ score, as defined earlier. After this step we obtain a weighted directed graph G (our road map).

## Query Phase

Given a user-defined query $P_{start}$ and $P_{goal}$, PRM solves the query by connecting these nodes to the road map $G$ via a distance function, and running Dijkstra's single-source shortest path algorithm to find the lowest-cost feasible path, where cost is the sum of the Bin scores of the edges in the path.

## Experimental results

In the tests reported here we created a roadmap with 12000 nodes using the Halton sampling method on the 4-dimensional space $[-8000, 8000] \times [-8000, 8000] \times [0, 1500] \times [0, 140]$ which is a realistic approximation of the area around a landing site. Without constraints we would have $12000^2 = 144 \cdot 10^6$ edges but, after all the constraints (defined before) are enforced we have approximately 376000 edges. For each of such edges we have computed the corresponding Bin value by running RNM. Roadmap construction took roughly 20 hours on a laptop[1] with medium capabilities. We note however that such a pre-processing must occur only once.

[1]Intel Core i7-2600K CPU @ 3.40 GHz 3.70GHz, 8 GB RAM.

In the query phase we tested the algorithm with different numbers of obstacles, all having the same volume. The obstacles we consider are parallelepipeds of type:

$$[x_s, x_f] \times [y_s, y_f] \times [0, 1500] \times [0, 140] \subset \mathcal{C}.$$

We first consider the impact of adding new obstacles on the average time to find an optimal path with its new road map.

The results are shown in Figure 3 and, as predictable, each time a new object is added a smaller number of edges must be checked in the query phase, thus yielding a sub-linear growth of time with respect to the number of objects.

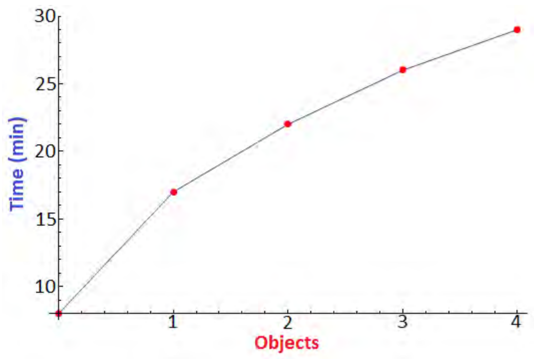


Figure 3: Impact of number of objects on computation time.

We have also conducted several experiments to confirm the soundness of our approach in terms of obstacle avoidance. In Figure 4 we show a 3D representation of a trajectory generated for a problem with 2 obstacles.

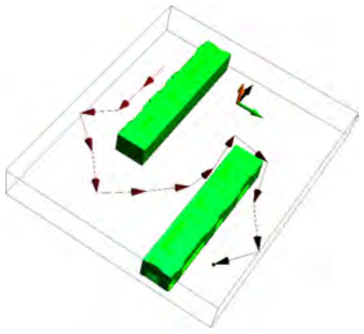


Figure 4: Correct trajectory avoiding two obstacles.

Another set of experiments we have conducted measures the performance of our PRM implementation in terms of solution quality when increasing the number of nodes. We selected two tasks:

1. A simple task, with only an object, defined as follows:
   $P_{start} = \{-5000, -5000, 800, 80\}$
   $P_{goal} = \{5000, 5000, 0, 60\}$
   Obstacle: $\{-3500, 3500, -3500, 3500, 0, 1500, 0, 140\}$, shown in Figure 5.
2. A more difficult task, with two objects and one speed limit:
   $P_{start} = \{-5000, -5000, 800, 80\}$
   $P_{goal} = \{5000, 5000, 0, 60\}$
   Obstacles:

$$\{\{-8100, 8100, -3500, -3000, 0, 850, 0, 140\}, \quad (1)$$
$$\{-8100, 8100, 3000, 3500, 100, 1400, 0, 140\}, \quad (2)$$
$$\{-3000, 3000, -3000, 3000, 0, 1500, 50, 150\}\}, \quad (3)$$

shown in Figure 6.

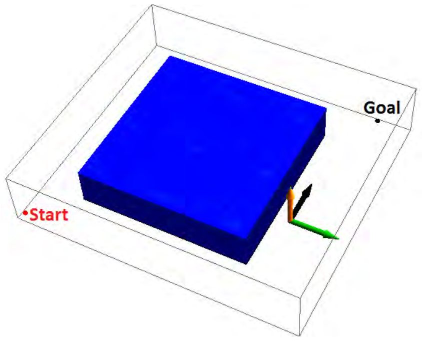


Figure 5: Easy task.

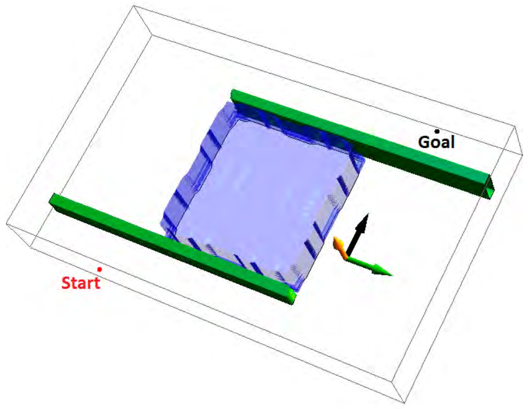


Figure 6: Hard task (with low opacity, in the middle, the speed limit obstacle (3)).

The results are:

| Nodes | Easy, Dijkstra-Noise | Hard, Dijkstra-Noise |
|---|---|---|
| 1000 | $+\infty$ | $+\infty$ |
| 1800 | 10.32 | 13.06 |
| 2000 | 9.55 | 11.77 |
| 3000 | 8.06 | 9.61 |
| 4000 | 6.85 | 7.96 |
| 8000 | 5.46 | 5.92 |
| 12000 | 5.26 | 5.36 |

The results are shown in Figure 7 where we plot the cost of the shortest path returned by the Dijkstra algorithm (proportional to the noise of the path). With only 1000 points the graph $G$ is not connected and the algorithm doesn't find a solution. Obviously the best path found in the easy task is better than the one found for more difficult one. Moreover,

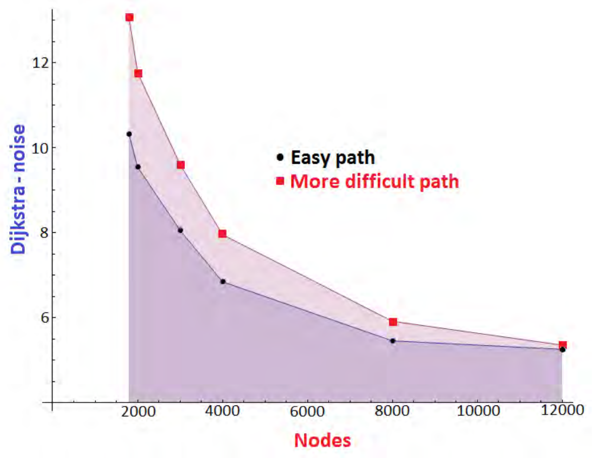


Figure 7: Solution cost vs number of nodes for the problems depicted in Figs. 5 and 6.

we observe that the number of nodes has to grow exponentially to improve the solution quality and that the more difficult path problems gain more advantage from an increased number of nodes than the easy one.

## Comparison with Local Search

Our final set of experimental results are aimed at providing a comparison with a different search technique, i.e. local search, which has been applied to solve a similar trajectory noise optimization problem in (Morris et al. 2012). The work presented here shares with the approach in (Morris et al. 2012) the use of RNM and of the Bin score function. However, there are differences worth noting before making the comparison. First, the search space in (Morris et al. 2012), is limited to "box"-shaped trajectories with two standard $45^o$ turns before the final approach leg. Second, with local search RNM is given as input a complete trajectory rather than single segments, as is the case with roadmap construction. Moreover, the result produced by RNM is not decomposable, in the sense that the SEL values computed for a complete trajectory are not linear combinations of those corresponding to segments of the trajectory.

Despite these differences, it is instructive to compare the performance of PRM with local search. To make the comparison more fair, we imposed a "box-shape" obstacle for PRM to use, as shown in Figure 8.

We performed a comparison between the Bin value of the solution obtained with PRM in the "box-shape" configuration space and the Bin value of the solution returned by local search on a fixed "box-shaped" trajectory lying in the free space of the configuration space.

Specifically, we ran PRM with the following initial configurations coinciding with the initial node of the seeds used for the local search runs:

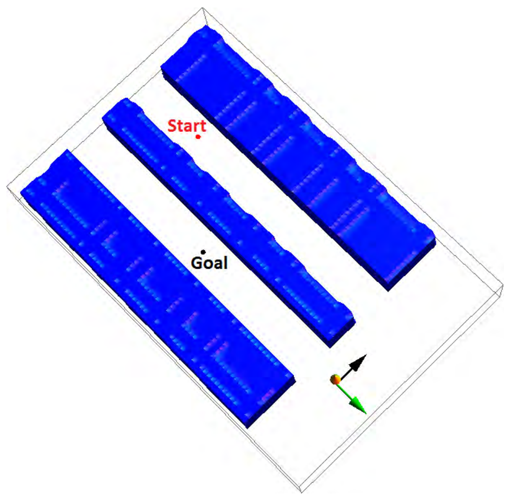


Figure 8: Box-shaped configuration space with 3 obstacles in order to better compare PRM and Local Search.

$$\begin{aligned} P_{start} &= \{-3900, 3600, 1000, v\} \in \mathcal{C}_{free}, \\ P_{goal} &= \{0,0,0,0\} \in \mathcal{C}_{free} \\ v &\in \{60, 80, 100, 120, 140\}. \end{aligned}$$

The average results of 100 Local Search tests with two levels of probability of random moves (low,high) and random initial velocity (from 60 ft/s to 140 ft/s) are reported in the following table:

| Test number | Local Search (high) | Local Search (low) |
|---|---|---|
| 1 | 147.9593 | 151.173 |
| 2 | 147.6934 | 151.5384 |
| 3 | 148.4025 | 151.1155 |

However PRM has obtained the following values:

| Start velocity (ft/s) | RNM-Noise value of PRM |
|---|---|
| 60 | 137.34 |
| 80 | 139.55 |
| 100 | 141.60 |
| 120 | 143.73 |
| 140 | 145.12 |

The results showed that PRM, even when constrained to a fixed trajectory shape, produced around a 6% improvement in noise minimization with respect to local search (an intensive search focussed on that class of trajectories). These results should, however, be taken with a grain of salt since, PRM, although limited, still benefits from a slightly larger configuration space, and from the advantage of a lengthy road map construction phase. In general, techniques like PRM are expected to perform better in higher dimensional configuration spaces than simpler approaches such as local search.

## Future Work

The work presented in this paper is a first step in the application of path planning techniques in the context of trajectory noise optimization for rotorcrafts. As such it suggests a number of extensions, which are summarized here.

We are currently finalizing the design and implementation of an A*-based approach to this problem in a similar setting as the one considered by local search (that is, with a fixed trajectory physical path). Preliminary results make it already obvious that PRM, as well as local search, outperforms complete search in terms of scalability and flexibility. We plan to study in more depth a comparison between these two approaches.

A natural extension of our current PRM is that of adding a $5^{th}$ dimension to the configuration space representing the rotorcraft heading (see Figure 9) and the associated feasibility constraints. This would allow the generation of more realistic trajectories, in particular, in terms of connections between segments of the road map, in fact rotorcrafts can travel in one direction while they are oriented along another heading. Another alternative we plan to pursue is post-processing the PRM solution with an interpolation or with a smart smoothing of the connections of the segments. We plan to implement and compare both these approaches in order to evaluate the tradeoff between configuration space augmentation and post-processing.

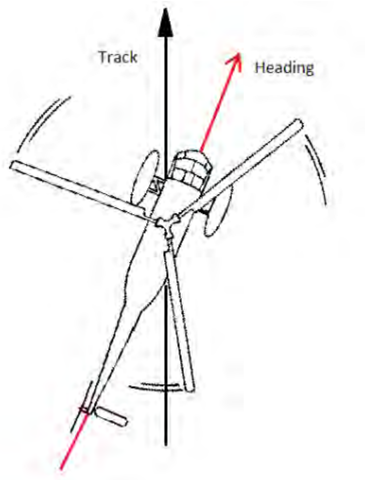


Figure 9: Heading in rotorcraft.

Finally, a possible development of PRM is its integration with local search in a two-phase process: for example by first finding a solution with PRM and then using this solution as an initial seed to local search. In particular, we think this may be effective if a smart way is found to set up the neighborhood for local search given the generality of the PRM solution. In that case the complementary strengths of the two approaches would allow to mitigate each others limitations.

## Conclusion

This paper has proposed the use of Probabilistic Road Maps to solve a 4D trajectory optimization path planning problem for rotorcraft. Sampling-based methods like PRMs were developed to address the problem of solving complex path planning problems in continuous, high-dimensional configuration spaces. In addition, sampling-based methods for path planning enable the representation of obstacles in the configuration space. These advantages of PRM suggest it as a logical approach to solving the problem defined here. One technical challenge in automated design of low noise trajectories is the need to deploy robust simulation tools for evaluating candidate trajectories during planning. Another challenge is to define a set of dynamic constraints on the solution space to allow solutions to be 'flyable' by pilots as well as quiet. The results of using PRMs documented in this paper, though preliminary, show their potential for devising effective rotorcraft noise mitigation strategies through approach trajectory design.

## References

Atkins, E., and Xue, M. 2004. Noise-sensitive final approach trajectory optimization for runway-independent aircraft. *Journal of Aerospace Computation, Information and Communication* 1:269–287.

Choset, H.; Lynch, K. M.; Hutchinson, S.; Kantor, G. A.; Burgard, W.; Kavraki, L. E.; and Thrun, S. 2005. *Principles of Robot Motion: Theory, Algorithms, and Implementations (Intelligent Robotics and Autonomous Agents series).* A Bradford Book.

Kavraki, L. E.; Svestka, P.; Latombe, J.-C.; and Overmars, M. H. August, 1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on robotics an automation, Vol.12, No.4.*

LaValle, S. 2006. *Planning Algorithms.* Cambridge University Press.

Morris, R.; Venable, K.; Pegoraro, M.; and Lindsay, J. 2012. Local search for designing noise-minimal rotorcraft approach trajectories. In *IAAI*. AAAI.

P. Cheng, S. Z., and LaValle, S. M. 2001. rrt-based trajectory design for autonomous automobiles and spacecraft. *Archives of Control Sciences* 11(3-4):167–194.

RNM. 2007. *Rotorcraft Noise Model Technical Reference and User Manual (Version 7).* NASA Langley Research Center.

# Synthesizing Fractionated Spacecraft Designs as a Planning Problem

**Minh Do[*], Martin Feather[‡], David Garcia[†], Kenneth Hicks[‡], Eric Huang[†], Dave Kluck[¶], Ryan Mackey[†], Tuan Nguyen[‖], Jami Shah[‖], Yorgos Stylianos[¶], Raffi Tikidjian[†], Serdar Uckun[§], Zihan Zhang[‖], Rong Zhou[†]**

[*]SGT Inc. & NASA ARC, Moffett Field, CA. Email: firstname.lastname@nasa.gov
[†]Palo Alto Research Center, Palo Alto, CA. Email: firstname.lastname@parc.com
[‡]Jet Propulsion Lab, Pasadena, CA. Email: firstname.lastname@jpl.nasa.gov
[§]CyDesign, Palo Alto, CA. Email: firstname.lastname@cydesign.com
[¶]Mission Critical Technology, El Segundo, CA. Email: firstname.lastname@mctinc.com
[‖]Arizona State University, Tempe, AZ. Email: firstname.lastname@asu.edu

## Abstract

There is an emerging interest in designing fractionated spacecraft that can share or take over duties from one another, leading to more flexibility and robustness in the face of requirement changes, failures, and delays. This new paradigm requires new software design tools to help with: (i) complex interdependency constraints and objective functions; (ii) uncertainties; and (iii) a very large number of feasible designs. *FRACSAT*, our design tool, automatically synthesizes feasible designs and graphically displays the results with all of their important quality measures. This allows designers to visually compare among various designs along different key metrics, and hone in on designs that are more likely to offer the best tradeoff. At the core of our design tool is the *PlanVisioner* planning system that takes on a PDDL-like model of the problem and generates all feasible designs. In this paper, we describe the fractionated spacecraft design synthesis problem, how it can be modeled in a PDDL-like language, the challenges that our particular application poses for the planning research community, and the empirical results of our planning software in this domain.

## Introduction and System Overview

Given a set of mission requirements and a library of available spacecraft components, spacecraft designers must design a completely functional spacecraft, along with a schedule to build and launch it, to satisfy the mission requirements while providing the best tradeoff between different quality measures. A fractionated spacecraft consists of multiple separate modules, or fractions[1], working together to provide improved flexibility, robustness, adaptability, and upgradability compared to a monolithic design with a single module/fraction. The challenge is how to choose and assign components to a set of fractions while respecting the functional and design constraints and optimizing various metrics.

Fractionated spacecraft design has not been previously addressed adequately by automated tools and typically requires a committee of domain experts over a span of several weeks. Furthermore, spacecraft design is complex enough that typical designs often forgo cost-benefit optimization in favor of safe, previously-proven, and simpler monolithic designs. Better value may be realized through fractionated design, but it is a more complex design problem that is often only manageable with automated design tools.

[1]Each fraction can be considered as a single working (small) satellite.

*FRACSAT* is our solution to conceptual design, cost-benefit analysis, and detailed trade studies for fractionated space systems[2]. It automatically synthesizes, compares, and explores thousands of feasible fractionated space architectures, leading to a short list of all solutions and their expected cost-benefit values within user-specified service requirements. Figure 1 summarizes *FRACSAT* while this paper concentrates on the software framework underlying the first and second steps: *feasible designs generation* and *design tradeoff exploration*. *FRACSAT* uses a familiar cost-benefit analysis framework to display the space of potential solutions, leading the user toward the preferred Pareto-optimal frontier of cost and utility. The core of our design tool is the *PlanVisioner* planning system, which generates and compares all feasible designs.

While this design problem can be solved by other optimization techniques, such as Constraint Satisfaction Problem or genetic algorithm, we use planning and scheduling technology due to the natural fit:

- Explicit mission objectives, requirements, and preferred system capabilities map well into planning goals (both hard and soft goals).
- Causally related tasks in different major steps (e.g., design, build, launch, operate) map well into planning actions.

Technically, *PlanVisioner* uses model-based forward state-space planning framework utilizing advanced search algorithms and techniques[3]. Its input are:

1. Library of spacecraft components, each with a different set of properties and constraints;
2. Multi-dimensional mission requirements.

Taking these inputs modeled in a PDDL-like standard planning modeling language, *PlanVisioner* exhaustively searches for all designs that (1) are consistent with all constraints such as mass, power, component compatibilities, and (2) satisfy all of the hard and some of the soft requirements. For each candidate, *PlanVisioner* reports various quality metrics, including estimated total cost and utility,

[2]The *FRACSAT* project was funded under DARPA System F6 program.

[3]Forward-state space search is a good choice to find simple valid solutions quickly while keeps on searching for more complex solutions within the full set of feasible solutions. That is important when there are human users looking at the screen waiting for solutions to analyze.

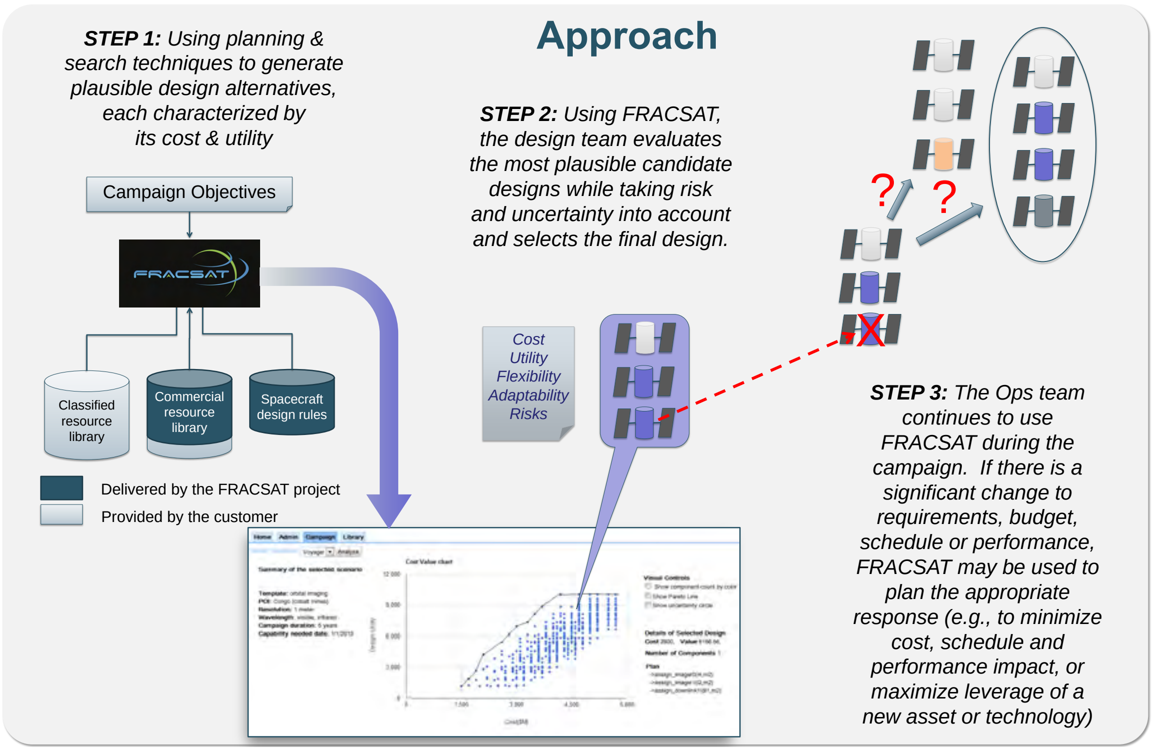

Approach
STEP 1: Using planning & search techniques to generate plausible design alternatives, each characterized by its cost & utility
Campaign Objectives
FRACSAT
Classified resource library
Commercial resource library
Spacecraft design rules
Delivered by the FRACSAT project
Provided by the customer
STEP 2: Using FRACSAT, the design team evaluates the most plausible candidate designs while taking risk and uncertainty into account and selects the final design.
Cost
Utility
Flexibility
Adaptability
Risks
STEP 3: The Ops team continues to use FRACSAT during the campaign. If there is a significant change to requirements, budget, schedule or performance, FRACSAT may be used to plan the appropriate response (e.g., to minimize cost, schedule and performance impact, or maximize leverage of a new asset or technology)
?
?
Cost Value chart

Figure 1: FRACSAT Overall Framework.

adaptability measure, and uncertainty level (due to mission risks). These metrics allow the user to narrow down the search space and focus on the most desirable designs. Once the space is narrowed down to a few choices, the user can initiate more detailed analyses including orbital geometry, cluster configuration, data throughput, campaign duration, or the use of certain components and a detailed adaptability analysis.

While the soft-goal constraints in this domain allow for the easier finding of a single plan than the traditional planning problems, this domain poses a set of challenges that are not common in current planning research benchmarks. Thus require significant extensions to existing planning systems. Specifically, our main contributions are:

- Introduce an application of automated planning technology in a new domain: fractionated spacecraft design.
- Extend the existing heuristic-search planning framework to find all feasible plans (not a single plan) quickly[4].
- Handling of complex goals/requirements that are: (1) both hard and soft constraints; (2) each involves multi-dimensional quality measures that are non-linear and not combinable; and (3) susceptible to changes resulted from various risks and uncertainties.

Our system is fully implemented and can run on multiple platforms. It was tested on several sets of benchmarks, finding up to more than 150,000 valid unique candidate designs for a given mission. Each design is returned with extensive information on: (1) quality measures: total cost, utility, and adaptability – all adjusted for various risks and uncertainties; (2) component composition; (3) build/launch timelines; and other useful information. The candidate designs are displayed through a web-based graphic user interface that has multiple filtering capabilities to assist human designer to navigate, compare, and select the final most suitable design.

The rest of this paper is structured as follow: we will start with a section on introducing the fractionated space-craft design problem and describe how we model it as a planning problem. We then follow with outlining the *PlanVisioner* software, its output, and the key techniques developed to solve this domain. We next provide the current empirical result and finish with the related and future work.

## Fractionated Spacecraft Design as a Planning Problem

The dominant paradigm in space operations is the *mission*, defined as a possibly repeating sequence of activities intended to achieve a given objective. Given a library of available spacecraft components, fixed objectives, timeframe, and budget, the design team finds the most suitable spacecraft design for that particular mission. From the planning research point of view, mission objectives can naturally be considered as planning *goals* and design activities as *actions*.

In *FRACSAT*, the process of building, assembling, and then launching a plausible spacecraft design is represented as a "plan" that consists of a sequence of primitive design actions. As the computation proceeds, this plan is refined from a set of design requirements, which include mission objective, to a concrete design. The design actions are

[4]In many critical applications, a person in charge would like to inspect *all* options and make the final decision himself instead of relying on the planner to select a single "best" solution for him.

```
(:functions
   ;; MASS-related of module/component/launch-vehicle
   (mass ?c - (either fraction physical_object)) - float
   (allowed_mass ?f - (either fraction spacecraft_bus)) - float
   (launch_capability ?v - launch_vehicle ?o - orbit) - float
   ;; COST (manufacturing/assemble/launch)
   (manufacturing_cost ?c - physical_object) - float
   ;; TIME & duration
   (manufacturing_duration ?c - physical_object) - float
   (lifespan ?o - (either component fraction)) - float
   ;; POWER: total power allowed for a spacecraft-bus
   (power ?f - (either fraction spacecraft_bus)) - float
   (power_consumption ?c - (either component fraction))  float
   ;; ORBITAL constraint
   (can_operate_on ?f - (either fraction spacecraft_bus))  orbit
 .....................................
;; Initial setting for a spacecraft-bus
(:init
   (= (manufacturing_cost LEOStar) 6000000) ;; USD
   (= (manufacturing_duration LEOStar) 29) ;; months
   (= (mass LEOStar) 263) ;; kg
   (= (allowed_mass LEOStar) 473) ;; kg
   (= (power LEOStar) 200) ;; watt
   (= (lifespan LEOStar) 120) ;; month
   (= (power_consumption LEOStar) 82) ;; watt
   (= (reliability LEOStar) 0.98) ;; success probability
   (= (can_operate_on LEOStar) LEO)
   (= (data_rate LEOStar) 2000) ;; kb/s
```

Figure 2: Spacecraft component properties modeled as function templates in the PTDL *domain* file and concrete specifications in *problem* file.

modeled in terms of action pre-conditions (e.g., an imager needs a bus and a power source) and effects (e.g., the imager can cover a certain spectrum and it increases the mass, power consumption, and cost of the fraction). The design action preconditions check for if a given set of constraints on component properties can be satisfied when adding/assigning a component to a design and the action effects reflect the overall design changes when such a component is added. Component properties and design actions are modeled using the PTDL modeling language, which is a variation of PDDL, a standard planning modeling language (Fox and Long 2003)[5].

**Examples:** Figure 2 shows examples of (i) component properties and constraints in PTDL in which they are modeled as functions, each with a list of parameters and the function value type; and (ii) how the actual values stored in the component library database are represented in PTDL as part of the initial state specification. Figure 3 shows an example of the design action that integrates an imager into a given spacecraft bus.

[5]While PDDL3.1 would be mostly sufficient, PTDL was used for two reasons: (1) the main reason is that it has been used as the modeling language for the planner at PARC for several earlier projects; that planner provides the basis for the *PlanVisioner* software in this project; (2) some actions such as "launch" use features beyond PDDL such as effects happening within the duration of an action.

```
(:action integrate_imager_into_fraction
  :parameters (?f - fraction ?i - imager ?o - orbit)
  :duration (manufacturing_duration ?i)
  :condition
   (start (has_bus ?f)
          (not (launched ?f ?o))
          (<= (mass ?f) (- (total_allowed_mass ?f) (mass ?i)))
          (<= (power_consumption ?f)
               (- (power ?f) (power_consumption ?i))))
  :effect
   (end (change (has_imager ?f ?i) F T))
   (start (increase (mass ?f) (mass ?i))
     (increase (power_consumption ?f) (power_consumption ?i))
     (increase (total_cost) (manufacturing_cost ?i)))
   (end (increase (total_mission_utility) 0.075))
  :conditional-effect)
```

Figure 3: Example of a design action modeled in PTDL: adding an imager to a fraction/module.

## *PlanVisioner*: Finding All Legal Designs

*PlanVisioner*[6], our design-space exploration engine, find all possible designs utilizing heuristic-search based planning framework. Specifically, *PlanVisioner* is a forward-state-space planner that uses either depth-first or best-first search with customized duplicate and symmetry detection routines.

Like other search-based planners, *PlanVisioner* first encodes/builds the planning-relevant structure and rules:

- *Start state* (root search node): an empty design.
- *Expansion function:* check for design actions' applicability and change the partial design when apply them.
- *Goal/terminal condition:* represents when a partial design satisfies all hard goal/requirement constraints.

It then calls either a depth-first or a best-first search algorithm to generate and navigate the search graph dynamically and exhaustively.

To ensure broad coverage of the design space and avoid neglecting some outliner design, our algorithm usually finds a much larger set of design alternatives than human designers could possibly create, resulting in a far more thorough exploration of the design trade space. Furthermore, upon termination the algorithm can prove that it has reached 100% of the design space implied by the design rules, offering a mathematically sound way to enumerate the space of all possible distinct designs. Again, this is an important feature in critical applications like spacecraft design.

To generate all unique designs effectively, it is important that we employ some advanced search/planning techniques.

**Pruning identical (duplicate) designs:** The set of all possible designs grows rapidly with the increasing total number and types of components. A naive approach can produce identical designs that not only slow down the enumeration process but also create difficulty for the comprehension of human user. This is due to the fact that applying the same set of design actions in different order can create identical designs, and the number increases exponentially with the number of design actions. To remove

[6]Envisioning is a technique, popular in diagnostic, to envision/find all possible solutions.

identical designs, our algorithm stores the set of partial and complete designs found so far in a lookup table: each time a new design is generated, it is checked against the table for potential duplicates, and only designs that have never been generated before are kept for further design refinement. To conserve memory, the set of design actions applied so far for each design state is summarized in a 64-bit integer as the “signature”of the design, and if two designs have the same signature, then they are considered as duplicates. With $2^{64}$ possible design signatures, the likelihood of having a false positive is very low[7].

**Pruning symmetrical designs:** Symmetrical designs are non-identical designs that create essentially the same fractionation of the spacecraft. For example, in a design with two fractions/modules, if one swaps all the components originally assigned to the first fraction with those assigned to the second fraction, nothing really changes, but the two designs would not be identical because the mapping from components to fractions are different. To detect symmetrical designs, we introduce the concept of canonical fractionation as follows: because each component in our system has a unique ID, the set of components assigned to a single fraction can be seen as a word with the set of possible component IDs as its alphabet. A canonical fractionation is the one such that all the fractions are ordered lexicographically. For example, if fraction #1 has components $\{3, 4\}$ and fraction #2 has $\{1, 2\}$, then the two fractions do not make a canonical fractionation, since $\{1, 2\}$ should come before $\{3, 4\}$ in a lexicographical ordering. Our software only returns canonical fractionation.

**Parallel design space exploration:** To further speed up design space exploration, we developed a parallel search algorithm to enumerate different parts of the design space simultaneously. It is based on depth-first search with two phases: In the first phase, the algorithm enumerates the design space up to a maximum depth $d$, and any partial designs generated at that depth are distributed to multiple concurrent search threads using a hash function that maps a state to an integer between 0 and $p - 1$, where $p$ is the number of parallel threads specified by the user (default is 7). In the second phase, the remaining search space is explored concurrently by multiple threads until they all finish enumerating the designs contained in the sub-space assigned to them. For maximal concurrency, the parallel algorithm chooses a shallow depth $d$ to limit the amount of search in the first phase, which is performed sequentially. Typically, the second phase is by far more time-consuming and thus the parallel speedup is close to linear.

## Measuring Design Quality

The main role of *FRACSAT* is to be able to extract and display intuitively all information that are deemed important to human spacecraft designers, help them quickly compare between all possible designs, and hone in on the ones

Design **#9823**  
Number of fraction: **4**  
On pareto-frontier: **No**  
Design cost:  
  Expected: **$311.6M**  
  Range: **$293.023 - $359.338M** with **95%** confidence level  
Mission requirement satisfaction (mission-utility):  
  Expected: **90.39%**  
  Range: **86.87% - 95%** with **95%** confidence level  
Adaptability Measure:  
  Architectural adaptability: **0.76**  
  Valuation adaptability: **0.73**  
  Overall adaptability measure: **0.75**  
Fraction composition and specification:  
  Fraction/module 1:  
    Launch time: **April 2013**  
    Launch vehicle: **Atlas V**  
    Spacecraft-bus: **Mid Star**  
    Spare power: **40.5 W**  
    Spare mass on the bus: **1303.54kg**  
    Spare mass on launch vehicle payload: **28783.5kg**  
    Cameras (Imagers): **ECAM** and **HiRiseJr IR**  
    Other equipment: **F6 TechPackage S**, **ANPED 1 laser detector**  
  Fraction/module 2:  
    ............  
  Fraction/module 3:  
    ............  
  Fraction/module 4:  
    ............

Figure 4: Example design synthesized by *PlanVisioner*.

that provide the best tradeoff between different objective functions. Therefore, when a legal design is found, it needs to be evaluated based on mission-relevant objective function. The evaluation is then used to visually guide the user in selecting the final design. Figure 4 shows one example output of *PlanVisioner* with spacecraft-design related important information. They are extracted by (1) first mapping the set of design actions in the design plan into the list of components and processes involved; and then (2) extracting the relevant information from the components and processes used. For each design, the tool also indicates if it’s on the Pareto-frontier based on the computed quality measures. For the rest of this section, we outline for each of them why it is important and how to compute it.

**Design Cost:** design cost and utility are the two most important criteria in measuring the usefulness of a given design, they answer the key question: *how much would a customer get for a given paid price?* Design cost is easier to compute due to the additive nature. A cost of a design is the summation of the costs of all “design actions”, be it manufacture or assembly a component into the final design or to launch and operate a spacecraft. However, due to cost-uncertainty (more on this in the next section), we return both the expected and range of the total cost within the 95% confidence level.

**Design Utility:** Total utility is measured by aggregating over the mission duration to determine how well the overall ca-

[7]The 64-bit signature, produced by a hash function, has been widely used in model checkers such as SPIN, and is also known as “hash compaction”. In case of duplication, since any two designs can have a non-zero probability of having the same signature, in general there is no way to bound or estimate their cost/utility distance.

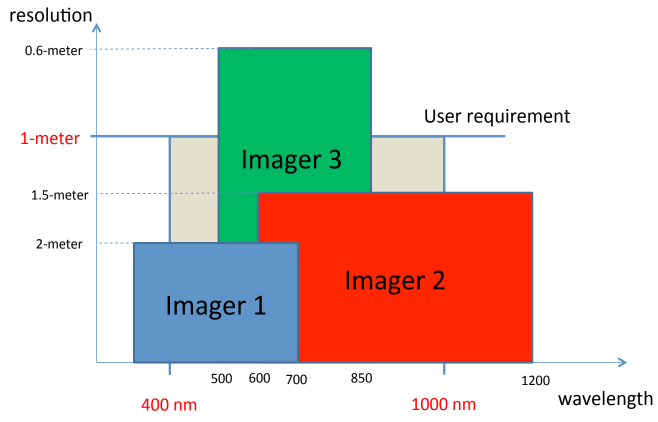


Figure 5: Imager-based Utility function

pabilities of a candidate design satisfies requirements. Thus, the maximum utility value is *1* when all requirements are satisfied and the minimum value is *0* when no requirement is met. Design utility computation is complex[8]; thus, we will only address time-sensitive image-quality related requirements in this paper as a concrete example. In the example depicted in Figure 5, assume that the user requirement is collecting images covering the entire *400 nm - 1000 nm* wavelength range, with an image resolution of *1m* during the four-year period. If for a duration $D$ there are 3 imagers available on all fractions of a fractionated design, then during $D$ the maximum design utility with regard to this requirement can be measured by how well the union of the three "rectangles" (see Figure 5), representing the three imagers, overlap with the rectangle representing the user requirement. Given that different fractions in this architecture may be launched at different times during the mission, and that each component may operate with a different lifespan, the set of equipments available at different time periods within the overall mission lifecycle will change. Therefore, we calculate design utility separately for each period where the set of operational equipment does not change and return the total aggregated design utility.

During our analysis, we compute a rough estimate of design utility as well as a more detailed analysis taking uncertainties into account. In our example in Figure 5, suppose that there is only a single risk, namely that *Imager 1* is associated with a fraction whose launch vehicle has a success rate of 90%, then we only expect *Imager 1* to contribute to our utility computation 90% of the time. Overall design expected utility is thus computed by aggregating over utilities computed with all possible failure combinations. Besides the expected utility value, like design cost, we also compute the 95% confidence intervals of design utility.

**Design Adaptability:** Given that various changes will happen during the designing/manufacturing/launching/operating phases of the spacecraft, adaptability is a critical criterion in measuring the quality of a given design. Whenever a point design is generated by the *PlanVisioner*, it will invoke adaptability computation to measure three different adaptability values: (1) valuation-based adaptability measure; (2) architecture-based adaptability measure; and (3) overall/combined adaptability measure (as shown in Figure 4):

- *Valuation-based adaptability* measure is related to the ability to gain additional value, possibly at additional cost, when requirements change. We calculate adaptability measure by aggregating over all possible requirement changes and compute the discrepancy in design utility measures with regard to the old and new requirements.
- *Architecture-based adaptability* measure is an alternative in which we identify characteristics of the product architecture that are conducive to change and *quantify* that characteristic for a given design. In the current implementation, this metric is computed using spare power ($P$), spare mass on bus ($M$), and spare mass on launch vehicle payload ($L$) (see Figure 4); the more they are, the more possibility for adaptation to changes. The architecture-based adaptability measures for a given design, among all $n$ candidate designs, are then computed as: $X_i = P_i/(\underset{1\leq j\leq n}{MAX} P_j)$, $Y_i = M_i/(\underset{1\leq j\leq n}{MAX} M_j)$, and $Z_i = L_i/(\underset{1\leq j\leq n}{MAX} L_j)$. Unlike valuation-based measure, we do not need to know the concrete set of possible changes to compute this adaptability measure.

The *overall-adaptability* measure of a given $i^{th}$ design is computed as a weighted sum of all adaptability measures: $A_i = w_1 \times R_i + w_2 \times X_i + w_3 \times Y_i + w_4 \times Z_i$ with $\sum_{1\leq j\leq 4} w_j = 1$.

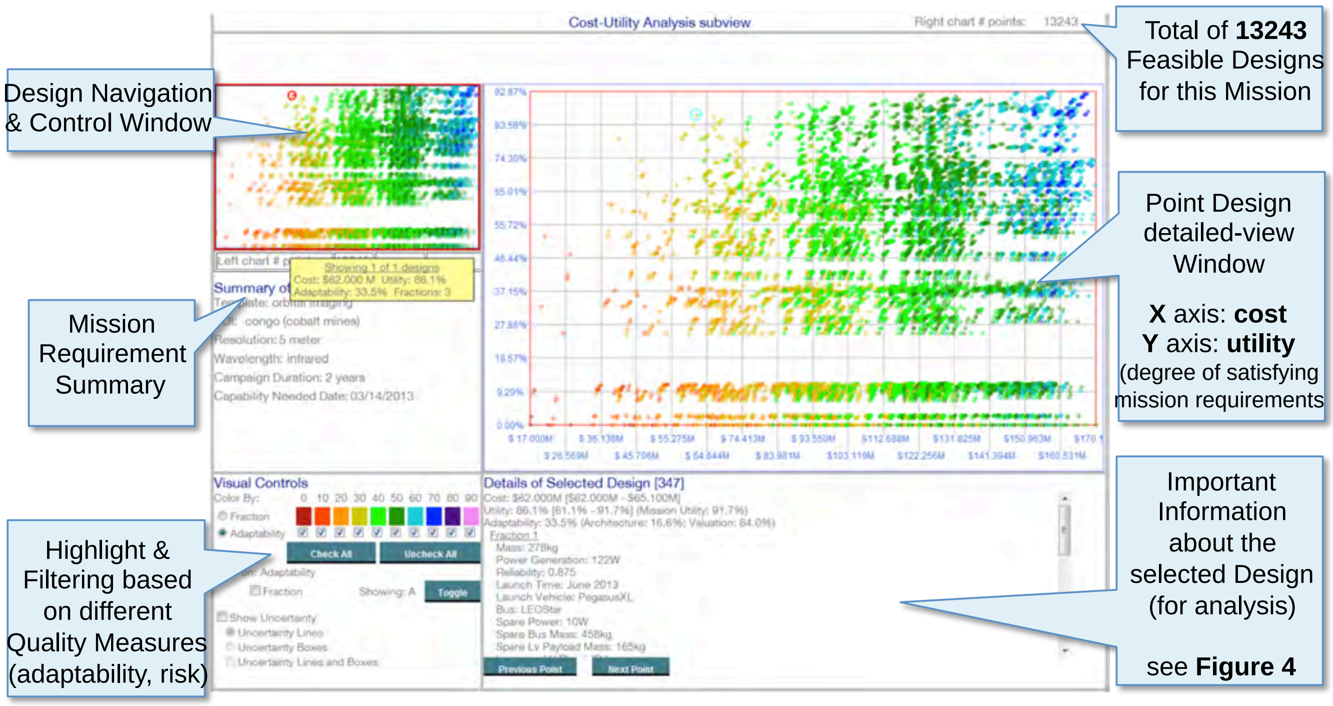
**Design components:** As shown in Figure 4, for each design, we output the detailed component composition such as: spacecraft-bus (frame) used, launch-vehicle (rocket) used for each module, and on-board cameras/imagers and other equipments (e.g., communication). The detailed composition will help the human designer, when using *FRACSAT*, to analyze each design at a detailed component level.

**Other design information:** Additionally, the planner also outputs various architectural information, as identified by experts, that are useful for the designer to access the quality of each design, such as spare mass and power on the bus (frame) and launch vehicles – indicating the flexibility in which additional equipments can be put on and launched along with that particular spacecraft design.

## Uncertainty & Risk in *FRACSAT*

Risks to the mission and outcome uncertainty are important to mission designers. For a given design, *PlanVisioner* computes a set of final results based on possible outcomes of a given uncertain event. In relatively simple cases, the entire set of outcomes is generated, with each outcome annotated with its probability of occurrence. In more complex situations, where it is computationally infeasible to generate and store the entire set of outcomes, a statistical profile of that set is generated using Monte Carlo sampling (Michael

[8]Overall, we have implemented 22 basic parameterized utility functions (e.g., step, linear, sigmoidal). They can be used individually or as a combination (e.g., weighted sum, nested functions) to compute design utility with regard to different mission requirements. Each numerical performance requirement is associated with a utility function, describing the relative penalty or reward for not quite meeting or exceeding the specification.


Cost-Utility Analysis subview
Right chart # points: 13243
Total of **13243** Feasible Designs for this Mission
Design Navigation & Control Window
Point Design detailed-view Window
**X** axis: **cost**
**Y** axis: **utility**
(degree of satisfying mission requirements
Mission Requirement Summary
Highlight & Filtering based on different Quality Measures (adaptability, risk)
Important Information about the selected Design (for analysis)
see **Figure 4**
Left chart # p
Visual Controls
Color By:
0 10 20 30 40 50 60 70 80 90
Fraction
Adaptability
Check All
Uncheck All
Showing: A
Toggle
Previous Point
Next Point

Showing 1 of 1 designs
Cost: $62.000 M Utility: 86.1%
Adaptability: 33.5% Fractions: 3

Summary of
Template: orbital imaging
AOI: congo (cobalt mines)
Resolution: 5 meter
Wavelength: infrared
Campaign Duration: 2 years
Capability Needed Date: 03/14/2013

Show Uncertainty
Uncertainty Lines
Uncertainty Boxes
Uncertainty Lines and Boxes

Details of Selected Design [347]
Cost: $62.000M [$62.000M - $65.100M]
Utility: 86.1% [61.1% - 91.7%] (Mission Utility: 91.7%)
Adaptability: 33.5% (Architecture: 16.6%; Valuation: 84.0%)
Fraction 1
Mass: 278kg
Power Generation: 122W
Reliability: 0.875
Launch Time: June 2013
Launch Vehicle: PegasusXL
Bus: LEOStar
Spare Power: 10W
Spare Bus Mass: 458kg
Spare Lv Payload Mass: 165kg

Figure 6: Analysis pane displaying *FRACSAT* designs

2002).

**Examples**: Instead of a launch rocket functioning correctly, it might explode or otherwise fail to deliver its payload to orbit. New components such as a new camera based on emerging technologies may have uncertain costs, since the cost of the technology may change as it is being developed. Supply chain disruptions and other delays in availability of a component may cause slips in the launch schedule, affecting both the mission utility and costs.

Currently, *PlanVisioner* represents risks as discrete events with two possible outcomes, each with an associated probability. It also represents the uncertainty associated with a measure (e.g., cost, schedule deadlines, component performance metrics) using a triangle distribution, comprising a triple of low, expected, and high values. Selection of the triangle distribution as *FRACSAT*'s default representation is due to its advantage in situations of very limited information such as the development of new technologies, or the use of novel combinations of existing technologies (of Defence 2007).

When computationally feasible, *PlanVisioner* explores all interactions of these risks and their ensuing effects on cost and utility, weighted by the probability of such occurrence. The results are used to compute intervals for the user-specified level of confidence. At present, we have focused on launch vehicle failure and launch delay risks, though the method can be extended to reason over many other forms of risk.

As mentioned in the previous section and shown in the example output (Figure 4), quality measures affected by risk-related computation include the expected utility and cost, the standard deviation, the minimum and maximum cost and utility values for a 95% confidence interval.

## Empirical Evaluation

Our *PlanVisioner* software is written in C++ and has been compiled for and tested on multiple platforms, including Windows, Linux, and Mac.

We have created several benchmark sets with different component libraries and classes of mission requirements, each set contains more than 10 problems varying by: (1) the type of components involved; (2) number of components for each type; (3) complexity of the mission requirements. In all cases, the component library contains actual publicly available spacecraft component specifications and reliability measure. For the mission requirements, we use the most common mission scenarios. Overall, there are up to: 21 components, 6 launch sites, up to 4 fractions, 2 objectives, 3 requirements, 10 required capabilities, approximately 5-year of mission duration, and 22 utility functions.

For the set of simpler surveillance objectives, our software found over 156,000 distinct designs, each contains no more than three fractions, completing the search in less than 30 seconds. In a more complex scenario sets, the *PlanVisioner* has found up to 90K solutions within up to an hour of search time. All experiments were run on either Windows or Linux servers with 4-8 GB of RAM. When the problems are too complex and take too much planning/search memory to find all designs, switching from best-first-search to depth-first-search helps reduce the memory consumption.

All designs when found and analyzed by *PlanVisioner* are displayed to user for further investigation through the *Analysis Pane* within the *FRACSAT*'s GUI. Figure 6 shows the analysis pane of an exemple scenario. In this particular example, *PlanVisioner* found 13243 designs in about 90 seconds. There are two navigation windows, the smaller

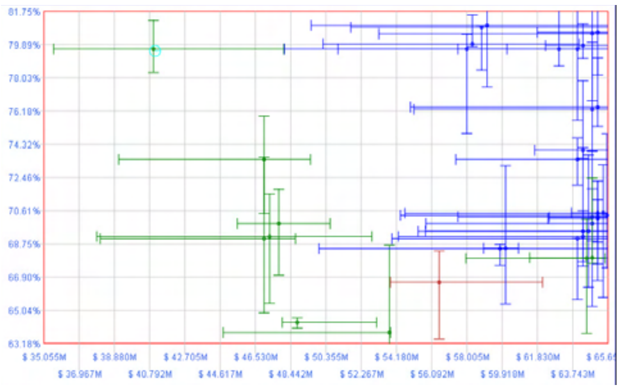


Figure 7: Detailed-view window with *Show Uncertanty* option: design cost and utility ranges are shown besides the expected values.

control/summary window at the top-left corner is used to select any smaller subset of designs. The detailed-view window at the top-right corner, which can be zoomed-in/out, displays in more details all designs selected in the control windows. The mission requirement summary pane is at left-middle. The detailed information (similar to the one in Figure 4) is shown in the lower-right corner. At the lower-left corner, the *Visual Controls* pane allows users to use different filters when viewing valid designs in the main window (including putting bounds on cost/utility). In this figure, the *adaptability filter* is shown where designs are color-coded according to their adaptability values. Figure 7 shows an example of the main analysis windows when *show uncertainty* option is selected: each individual design's cost and utility value is expanded from a single point into ranges of *[min,max]* value with 95% confidence level.

**Effectiveness of Duplicate & Symmetry Detection:** duplicate and symmetry detection helped greatly both in terms of speeding up the exhaustive search process and also in significantly reducing the number of designs being shown to the user. In one instance, the *PlanVisioner* algorithm found 912,672 designs after 8,031,080 expansions in 172 seconds. After activating symmetry detection, we found that only 156,848 designs are truly unique, which can be computed in 29 seconds. In this particular test case, given the upper bound of 3 fractions on each fractionated design, the improvement achieved by symmetry detection is a factor of 6 (= 3!). In general, the savings is O(n!), where n is the maximum number of fractions allowed.

**Uncertainty & Risk Computation Performance:** Because our algorithm is able to perform either an exploration of all combinations of events or a Monte-Carlo simulation, the user can choose to trade off between performance and solution quality. For risks of launch delay and launch vehicle failure, for example, with three fractions we would have to explore a tree of $2^{2\times3}$ possibilities and compute the cost and utility for each of these combinations. Similarly, with four fractions we would have to compute $2^{2\times4}$ possibilities. This problem grows exponentially with the number of risks and fractions, so we limit this exact computation to only risks that a user might believe to have low likelihoods of occurring but have high impact on cost and utility, otherwise known as "black swans". The effect of all other risks are computed by Monte-Carlo sampling such as with component cost uncertainties. We have found that assessing launch risks and component costs with 10,000 Monte-Carlo samples provides a reasonable response time of a few seconds to the user. The leaves from the combinatorial search tree as well as the samples from the Monte-Carlo method are used to determine the confidence intervals for cost and utility. As shown in Figure 7, each design is a dot represented on the cost-utility plot, with error bars representing the 95% confidence interval for the cost or utility value of the corresponding design.

## Related & Future Work

There are previous work on utilizing planning & scheduling and optimization techniques in spacecraft and mission design, especially work done at the Jet Propulsion Lab (JPL). They range from applying to a particular mission (Knight *et al.* 2012) to a metaheuristic optimization framework that can be applied and adapted to solve multiple problems related to spacecraft designs (Fukunaga *et al.* 1997). Perhaps the most closely related to our work is Smith et al.'s (2000) in which a planner is used to generate mission plan to best satisfy mission objectives while staying within constraints (e.g., cost, mass, and operability). This allow mission engineers to evaluate candidate designs against a given mission scenario. While previous work have gone to deeper length with test on actual missions, which we haven't, our work is different in the sense that we automatically generate *all* possible designs along with the most suitable mission plans that satisfy all hard constraints. This is in contrast to finding the best mission plan for a given design or evaluating cost surface for a particular component design.

The PTDL modeling language used in this project share a lot of similarities with the standard PDDL2.1 modeling language (Fox and Long 2003). The main extensions are: (1) the ability to model conditions and effects happening at any point during the action duration, and (2) it supports multi-valued variable natively, similar to PDDL3.1. The state representation of *PlanVisioner* is based on timeline representation developed within the Plantrol project (Do and Uckun 2011).

Computation of risk and uncertainty is commonplace. However, there is novelty in the *FRACSAT* application of these computations to the interrelated factors of cost, schedule and performance (utility) in the design of space system architectures. For risk, conventional design practice is to assess risks individually (e.g., via a 5x5 risk matrix) and plan the mitigation of the most serious of them. For uncertainty, conventional design practice is to hold program reserve (e.g., budget margin), with the fallback of descope options, as the approach. While these practices might have proven sufficient for monolithic architectures, the evolution to fractionated architectures exacerbates the complexity of the design challenge, and motivates the *FRACSAT* approach to explicitly compute the interrelated effects of risk and uncertainty.

Our fractionated spacecraft design problem introduces a challenging application domain, including many features that are of interest in planning and search community. The goal description can be defined by different teams of scientists with potential conflicting, and thus may not be fully satisfied (i.e. *soft* goals). This *over-subscription* planning problem has been motivated and formalized by Smith 2004, and several approaches to solving this problem based on heuristic search framework (Benton *et al.* 2009) and a method of compilation to planning with action costs (Keyder and Geffner 2009) have been devised. The goal conditions in our domain may also contain scientific “outcomes”, such as the number of soil images, during some specific interval of time. Such time-dependent goals have not been much considered by planning researchers.

While utility values of multiple soft goals are combined into a single objective function in over-subscription planning, several criteria in our design problem are not commensurable, and thus need to be optimized simultaneously. Though multiple objective search problem has long been considered in AI (c.f., (Stewart and White III 1991; Mandow *et al.* 2005; Mandow and De La Cruz 2010)), few planning work considers this realistic planning scenarios. The MO-GRT planner (Refanidis and Vlahavas 2002) derived a heuristic for state-space planners to find a most preferred plan, but can not handle prioritizing criteria of interest in a hierarchical fashion. On the other hand, Nguyen et al. (Nguyen *et al.* 2009) considers optimizing plan makespan and total execution cost at the same time, returning a *representative* set of plans to the user. It therefore shares similar motivation with ours where we are interested in presenting set of designs to the user on the space of design cost and mission utility.

As mentioned earlier, the users in our domain are interested in multiple requirements, which are modeled with different utility functions in our system. Such functions, in order to reflect realistic situations, can be complicated for the planner to optimize, and in fact planning with non-linear objective functions has not been paid enough attention in the planning community. Recently, Kawas et al. (Kawas *et al.* 2011) considers a planning problem with risk-averse utility function and formalizes it as both a mixed-integer nonlinear formulation and stochastic constraint satisfaction problem, tacking the function directly. Solving a risk-sensitive planning problem in MDP setting with one-switch utility function, Liu and Koenig (2005) takes different approach by approximating the function as multiple linear functions, and thus enabling optimization algorithm based on value iteration. We believe that these lines of work need to be closely considered in the community to bridge the gap between domain-independent planning techniques and real-world applications like ours.

**Future Work:** The main challenge that we want to address is related to the lack of the complete knowledge of user preference model. Even though the software needs to eventually return all designs so that it can guarantee the completeness and provide the user’s assurance that there is no legal design that is overlooked (this is important in mission-critical task), we would like to return designs that are more likely to be valuable and “interesting” to human designer (i.e., the ones that are more likely to be selected as the final design) earlier in the search process. This will allow the designers to inspect them closely while other (hopefully less interesting) designs are found. In other word, let $S$ be the set of designs that the planner has found and retuned to the human designer ($|S| = 0$ at the beginning and expands in real-time), we would like $S$ to be the most rep representative set of “good design” at any moment during the planning/search process[9]. We currently looking into extending the diverse-plan set finding framework (Nguyen *et al.* 2009) to adapt to the the metric measuring plan set quality in this domain.

## References

J. Benton, M. Do, and S. Kambhampati. Anytime heuristic search for partial satisfaction planning. *Artificial Intelligence*, 173(5-6):562–592, 2009.

Minh Do and Serdar Uckun. Timeline-based planning system for manufacturing applications. In *Proc. of Artificial Intelligence and Logistics Workshop (AILog) at IJCAI-11*, 2011.

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

A. Fukunaga, S. Chien, R. Sherwood D. Mutz, and A. Stechert. Automating the process of optimization in spacecraft design. In *Proc. of Aerospace Conference*, volume 4, pages 411–427, 1997.

B. Kawas, M. Laumanns, E. Pratsini, and S. Prestwich. Risk-averse production planning. *Algorithmic Decision Theory*, pages 108–120, 2011.

E. Keyder and H. Geffner. Soft goals can be compiled away. *Journal of Artificial Intelligence Research*, 36(1):547–556, 2009.

R. Knight, D. McLaren, and S. Hu. Planning coverage campaigns for mission design and analysis: clasp for the proposed desdyni mission. In *Proc. of Intl Symposium on Artificial Intelligence, Robotics, and Automation for Space*, 2012.

Y. Liu and S. Koenig. Risk-sensitive planning with one-switch utility functions: Value iteration. In *PROCEEDINGS OF THE NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE*, volume 20, page 993. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2005.

L. Mandow and J.L.P. De La Cruz. Multiobjective a* search with consistent heuristics. *Journal of the ACM (JACM)*, 57(5):27, 2010.

L. Mandow, J.L.P. de la Cruz, et al. A new approach to multiobjective a^* search. In *INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE*, volume 19, page 218, 2005.

Stamatelatos Michael. Probabilistic risk assessment procedures guide for nasa managers and practitioners. In *NASA Office of Safety and Mission Assurance*, 2002.

T.A. Nguyen, M.B. Do, S. Kambhampati, and B. Srivastava. Planning with partial preference models. In *Proceedings of the 21st International Joint Conference on Artificial intelligence*, pages 1772–1777, 2009.

Ministry of Defence. Three point estimates and quantitative risk analysis. a process guide for risk practitioners. 2007.

I. Refanidis and I. Vlahavas. The mo-grt system: Heuristic planning with multiple criteria. In *AIPS-02*, page 46, 2002.

B. Smith, B. Engelhardt, R. Knight, and D. Mutz. Automated planning for spacecraft and mission design. In *Proc. of 3rd International Symposium on Intelligent Automation and Control*, 2000.

D. Smith. Choosing objectives in over-subscription planning. In *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling*, pages 393–401, 2004.

B.S. Stewart and C.C. White III. Multiobjective a*. *Journal of the ACM (JACM)*, 38(4):775–814, 1991.

[9]While the plans/designs on the pareto-frontier are likely to be of good quality and interest to the users, collectively they are not the best representative set and thus adding designs on the pareto-frontier to $S$ one at a time is not a good strategy.

# Compilation of Maintenance Schedules based on their Key Performance Indicators for Fast Iterative Interaction

**Dara Curran, Roman van der Krogt, James Little, Nic Wilson**
Cork Constraint Computation Centre (4C), University College Cork, Ireland
n.wilson@4c.ucc.ie

**Abstract**

Finding the optimal solution for a scheduling problem is hard, both from a computational perspective and because it is frequently difficult to articulate what the user wants. Often there are a range of possible key performance indicators (such as makespan, resource utilisation, and priority of tasks), and thus there can be many objectives that we want to optimise. However, it will typically be hard for the user to state numerical tradeoffs between these objectives. Instead, it can be helpful if the user can explore the solution space themselves, to find which compromises between objectives they prefer. This paper demonstrates the use of Multi-valued Decision Diagrams in consideration of scheduling a real maintenance problem, namely the scheduling of Irish Navy dockyard maintenance. We show how candidate schedules can be compiled into MDDs, based on their associated Key Performance Indicators (KPIs). This representation allows the possible values of KPIs to be restricted by the user, and achievable values of other KPIs can be quickly determined, thus enabling fast iterative interaction with the user in order to achieve a satisfactory balance between the KPIs. We experimentally compare the performance of the MDD with that of a database, showing that the MDD can be considerably faster.

## 1 Introduction

Generating a good schedule is a complex task, for which highly optimised methods and software have been developed, see, e.g., (Brucker 2004; M.L.Pinedo 2008). This is especially true when there are multiple, possibly conflicting, objectives. There are several approaches for dealing with this problem, ranging from a simple linear combination of the objectives (for example, used by (Berrada, Ferland, and Michelon 1996)) to Pareto-based evaluation (such as employed by, e.g., (Johnston and Giuliano 2011)). However, when building a scheduling support tool, the clients' requirements and priorities are often unclear, even to themselves. As a consequence, it takes significant effort to elicit these from the end-user (van der Krogt, Little, and Simonis 2009). In the past, we have found that prompting the user with a schedule can generate feedback regarding what they want of a schedule. This can be used to produce another schedule, and over the course of several iterations, it is possible to learn the user's preferences.

However, the priorities are often subjective and can change between actual instances of a problem. It is therefore desirable to have ways of allowing the user to explore the possible solution space interactively—in particular, by putting bounds on the key performance indicators (KPIs)—before settling on a workable solution to move forward with. In order to facilitate this idea we need to generate sufficient compact compiled schedules in advance and to store them in such a way that makes navigation through them easy.

This paper proposes a system using Multi-valued Decision Diagrams (MDDs) to achieve this. In particular, it supports the Irish Navy in solving a maintenance scheduling problem. Initially, schedules, in the form of their key performance indicators, are compiled into an MDD. The KPIs can include, for example, such measures as makespan, utilisation of resources and duration of certain tasks. From the MDD, the system can efficiently retrieve the possible ranges of the KPIs, and show what is available in terms of possible schedules and their performance indicators. The user can inspect those, and focus on particular classes of solutions by constraining the KPIs, such as by enforcing that the utilisation of resource X should be at least 60%. The MDD can then be used to update the ranges of the indicators by enforcing the constraints proposed by the user. In parallel, each solution in the form of a set of KPIs has an associated actual schedule which the user can inspect at any time. This gives the user an end view to make a further judgement of choosing this plan or continuing searching.

The benefits of such a system are two-fold: *(i)* it takes away the burden of having to fully specify the desired behaviour; and *(ii)* it allows for a more flexible system where the user can easily vary preferences from one instance to the next. Also, since it puts the user in control, they may more easily accept the outcome of the tool, which does not always happen (see, e.g. (Fagerholt 2004), discussing vessel fleet scheduling).

This paper focuses not on the scheduling model (for that, we refer the reader to (Boyle et al. 2011)), but rather on the system around it. The remainder of this paper is therefore organised as follows. Section 2 briefly describes the maintenance scheduling problem; Section 3 discusses how we use

MDDs for representing achievable combinations of KPI values. The system architecture and experimental evaluation is described in Sections 4 and 5, and Section 6 concludes.

## 2 Context: Irish Navy Maintenance Scheduling

The context of our work is a maintenance scheduling problem on sea-going vessels for the Irish Naval Services. Their maintenance/refit policy across all their ships is based on a 28 day period (or 20 working days) every year. During that time, a team of specialist fitters, riggers, electricians and plumbers are employed at the dockyard to carry out the majority of tasks associated with the maintenance. Ideally, the schedule will have all the tasks completed within the time window without the need for unplanned outsourcing. Regarding the evaluation of a schedule, there are many criteria of interest, expressed as KPIs, representing utilisation of different resources, durations of activities, and so on.

The problem and our solution approach is described in detail in (Boyle et al. 2011); here we present only a short overview to give the user some context to the problem at hand. The constraints present in this problem can be divided into the following categories.

**Resource Constraints** There are two types of labour/equipment resources identified. The first type is the type one commonly sees in scheduling, which is dedicated to a single task for its entire duration (e.g. a welder replacing a piece of piping). The other type are those which are spread across a number of tasks at the same time in a supporting role such as cranes and foremen. A limit is imposed on how many tasks can be supervised simultaneously. The tasks are constrained generally in their durations, although several can be done in a variable amount of time depending on the number of resources assigned to it.

**Space Constraints** The restricted space on a ship can mean that it is sometimes difficult for two or more tasks to take place in the same area, or use the same access routes. The Naval Dockyard (NDY) (human) scheduler has already indicated which areas these are and hence which tasks are affected. For the same type of reason, tasks involving gas or welding, even in a large area, may require other tasks to be absent.

**Temporal Constraints** There are some cases where one task must follow another sequentially for logical reasons. Examples are *Deammunition* before *Magazine Service*, and *Remove Turbo* before *Rebalance Turbo*.

**Other Constraints** The granularity of time is half a day since this is the minimum the NDY scheduler currently allocates any task to a person. Using a scheduling model, it is easy to change this, and future work could look at the possible merits of adjusting the granularity. Of particular significance to scheduling is the task of engine service which takes the full 20 days to complete and sets a lower bound on completion time.

**Objectives** The objective is to maximise the number of tasks carried out internally, before any essential work is outsourced within the scheduling window. Beyond that, it is to finish the tasks as early as possible.

## 3 Background: MDDs

Multi-valued Decision Diagrams (MDDs), which generalise Binary Decision Diagrams (BDDs) (Bryant 1986; 1995) to non-Boolean values, have been studied for example in (Amilhastre, Fargier, and Marquis 2002; Wilson 2005; Andersen, Hadzic, and Pisinger 2010). An MDD is a directed graph with a unique *source* (i.e., initial) node and a unique *sink* (i.e., final) node. Each edge is associated with an assignment to a variable. Paths from Source to Sink correspond to assignments to a set of variables, and so the MDD represents a set of complete assignments via its set of paths. This can be a very compact representation, since the number of complete assignments represented can even be exponential in the number of nodes in the MDD.

We use MDDs as a compact representation of KPI values achievable by a consistent schedule for our problem. We have one variable for each KPI; the possible values of the variable represent small ranges of possible values for the KPI. An illustration of such an MDD is given in Figure 1. Here, the first variable might represent the KPI "Utilisation of Plumbers", with four possible value ranges. The second variable could represent the duration of some particular task. A path in the MDD now corresponds to a feasible assignent to all KPI variables, which corresponds to one or more schedules.

From such an MDD we can efficiently compute a number of things. Firstly, an MDD can efficiently return the number of possible designs remaining at any time, given a number of choices having already been made, simply by counting the remaining paths in the graph. For example in Figure 1, there are six paths left. This means that the user is informed of the size of the remaining search space at any time, helping the user in understanding the impact of their decisions. We can also use this information to guide the user to those variables whose choice makes the biggest impact.

Secondly, MDDs can invoke propagation between categories. When a particular choice/value is made for a schedule KPI, then all other associated edges of that choice in the MDD are removed. All paths going through those edges are therefore also no longer present in the MDD, thus removing some values in others choices. The remaining edges therefore represent the possible values for decisions within this new solution space. For example, suppose the user restricted the value of the first (topmost) variable to be one of the two edges on the left. This leaves only four paths in the MDD, none of which goes through the second value of the last (bottom) variable. Thus, this value can be removed from the set of possible values for this variable.

Thirdly, optimisation of any numerical decision simply becomes one of choosing the lowest (in the case of minimising) value and eliminating all the other edges for that particular KPI before propagating. This will leave at least one single path through the network representing the optimal set of decisions around the best value.

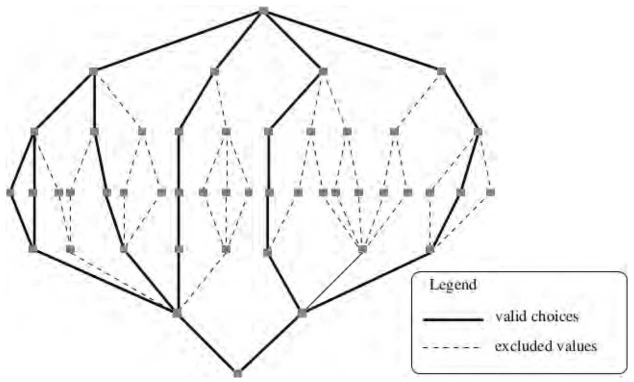


Figure 1: An example MDD

All of the above operations can be performed in time linearly dependent on the size of the MDD; indeed some of the results in Section 5 make this apparent.

## 4 System Architecture

The system described in this paper is comprised of five interconnected components:

1. a *schedule generator* to generate the initial set of schedules;
2. a *database* to store the schedules found;
3. an *MDD server* that stores the MDD generated for the set of solutions;
4. the *user interface* to allow the user to interact with the MDD server; and
5. a *local search module* to improve upon the schedules retrieved from the DB.

The following subsections describe each component in turn.

**Schedule Generator** The role of the schedule generator is to generate a large sample of schedules from a list of tasks, resources, resource assignments and schedule constraints provided by the user. In addition, the schedule generator creates an MDD from the values of KPIs derived from the generated schedules. Since our aim was to explore the usefulness of the MDD structure, the generator is very straightforward.

It creates schedules by iteratively altering the duration of each task and finding valid schedules for each task duration combination (if one exists). The schedule generator begins by calculating the minimum and maximum duration that is possible for each task, given the resources available. It then breaks up the minimum/maximum range into a series of segments, resulting in a list of possible durations for each task. For each of these durations, the generator first checks to ensure that a schedule is possible for a given task duration. If it is not, the task duration in question is blacklisted and not employed further. The generator then iterates through all possible combinations of task durations and attempts to generate schedules for each.

For each combination of task durations, the generator may find a large number of solutions. Many of these solutions will represent schedules that are very similar to one another which is not desirable from a user perspective. To limit this effect, each generated schedule is checked against previous schedules to ensure that it is significantly different. For simplicity, we use task start time as a symmetry breaking strategy. To be considered for inclusion, a schedule must have task start times that are different from previous schedules for the same set of task durations. To achieve this, we only include a candidate schedule if its task start times differences from other schedules are above a given threshold.

If the schedule is considered sufficiently different, an XML representation of the schedule along with the values for its KPIs are stored in the database (see the next section). A GANTT chart is also generated using GNUplot for each schedule and stored.

Once all the schedules have been stored, the generator builds an MDD from the KPI values in the following manner. For each KPI, its minimum and maximum values are obtained from the schedules. The minimum/maximum range is then broken into intervals and each interval for each KPI is added to the MDD and the MDD is stored on disk.

**Database** The database stores information on tasks, resources, the generated schedules and their corresponding KPI value assignments. The database is a MySQL database comprised of 6 tables. The schedules table holds the number of valid schedules (solutions) that have been found for a given task duration combination. These solutions are stored in the solutions table (each set of task durations may produce a number of valid schedule solutions). The KPI Assignment table holds the KPI values for each solution.

**MDD Server** The MDD server application loads the MDD from disk and listens for requests from the user interface to provide a number of functions:

1. Get the list of KPIs and their current value ranges;
2. Get the list of available schedules;
3. Select a value range for KPIs and update the MDD; and
4. Fetch schedule data from the database.

**User Interface** The user interface (see Figure 2 for a snapshot of the relevant part) allows the user to interact with the

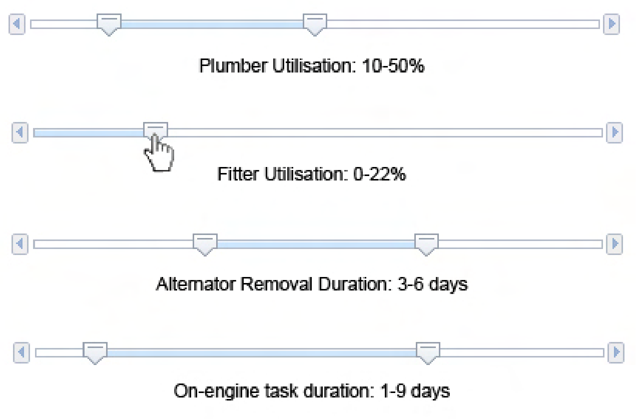

Plumber Utilisation: 10-50%
Fitter Utilisation: 0-22%
Alternator Removal Duration: 3-6 days
On-engine task duration: 1-9 days

Figure 2: Part of example User Interface

MDD Server by selecting ranges of values for each KPI, by moving the associated min and/or max sliders. The user can alter the desired range of a KPI by moving the sliders. For example, by dragging the left-most slider of the top range to the right, the user can specify that their desired value for the minimum "Plumber Utilisation" is more than the current value of 10%. Each time the user alters the range of a KPI, the MDD Server updates its MDD and returns a list of available KPI values remaining for this and other KPIs. These are then updated in the user interface, by automatically moving the min and max sliders to their new minima and maxima. For example, dragging the minimum plumber utilisation to 20% may result in the Alternator Removal duration changing to 4–6, which is indicated by the relevant slider moving. The user can then go on to restrict other KPIs. Maximising (minimising) a particular KPU can be achieved by moving the *min* slider (*max* slider) to the position of the other slider.

It is important to realise that although each of the values of a particular KPI can be achieved, this does not hold for arbitrary combinations of values for KPIs. When a choice in range alters the available options on another KPI, these are highlighted, so the user can clearly see the impact of their choices. An undo-mechanism allows the user to retract choices again (even out-of-order).

**"Local Search" for tweaking schedules** Once the user has set KPIs to their satisfaction and obtained a set of schedules from the system, they may find that while some schedules fit their constraints, they are prepared to relax certain KPI values in order to achieve better results elsewhere. For example, though they might like the KPIs of a particular schedule, they would prefer to increase the utilisation of fitters. This is similar to solution critiquing in recommender systems (McGinty and Reilly 2011; Ricci et al. 2011). However, adjusting KPI values manually at this point would become a trial-and-error process and so we devised a simple local search mechanism to perform this type of search automatically. Once the user has selected a schedule of interest, they have the option of selecting a KPI that they would like to either increase or decrease in value. To continue the example, the system might present the user with a list of choices such as including a different set of tasks or decreasing the duration of certain tasks by employing more fitters.

Again, the MDD data structure is helpful in this task. Given a particular new value or range that we want to achieve, the question essentially boils down to finding a new path through a particular edge (or one of a set of edges, as the same range or value may occur multiple times). This can be done as follows. For a particular edge, perform a best-first search both "upwards" and "downwards" to find nodes in the tree that are included in the current solution set. We define a cost function for each of the potential paths, with the cost being the number of edges whose values are not currently in the solution set. In this way, the cost reflects the number of changes we have to make to existing choices. Having found a new path, we can propose it to the user as a way of improving the desired KPI.

The procedure is illustrated in Figure 3. This shows only the upper part of an MDD for clarity. The user wants to include a value or range for the third variable. This occurs in the graph in three places, denoted by the numbers 1–3:

(1) This occurrence can be reached by expanding the restrictions on the second variable to allow the value or range represented by edge $a$, a cost of one change;

(2) This occurrence requires relaxing the restrictions on both the first and second variable, corresponding to edges $b$ and $c$, a cost of two changes; and

(3) This occurrence can be included by increasing the possible values of the first variable (edge $d$). This comes at a cost of one change, as it makes use of edge $e$, which represents a value for the second variable that is still included in some other valid path.

## 5 Experimental Evaluation

The justification for building an MDD to store possible user choices about schedules is motivated primarily by the efficiency at which such choices can be made and the model updated. In order to test whether our MDD approach has been successful in this regard, we conducted three experiments to compare the approach with the most obvious alternative of employing a database directly to search for solutions each time a user makes an adjustment to a KPI value.

The experiments are designed to simulate user interactions with the system in the form of repeated KPI min and max value adjustments. We compare our approach with a pure database system which uses SQL queries to retrieve the new minimum and maximum value ranges for each KPI as each choice is made. The experiments use data taken from Irish Navy ship maintenance schedules and consist of 64 tasks taking place over a fixed makespan utilising 47 resources. For our experiment we consider 2 types of KPI: task duration and resource utilisation (i.e., the amount of time that a resource is utilised over the entire makespan). This gives a total of 111 KPIs and the experiments employ 1329 schedules. (In practice, a human scheduler may choose to focus on a smaller set of KPIs.)

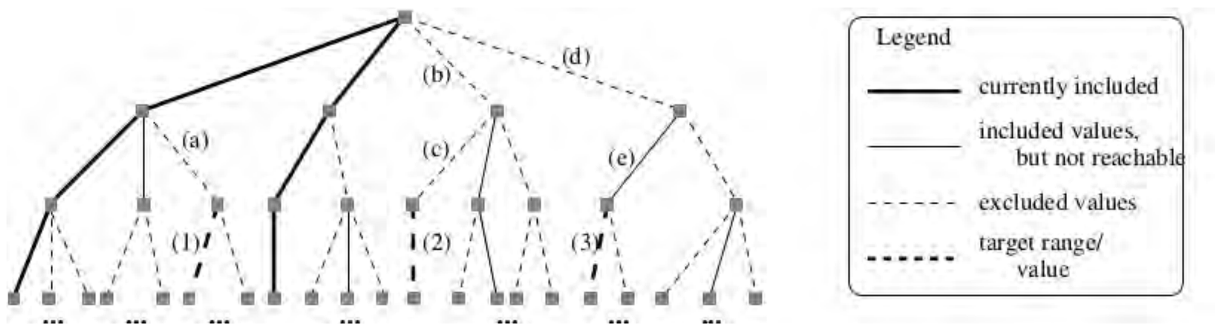


Figure 3: Illustration of the local search procedure. Shown is the upper part of an MDD

**Experiment 1** For the first experiment, a number of KPIs (between 1 and 111) are randomly selected for adjustment (with each KPI being chosen only once). Each selected KPI is set to a random range between its currently available minimum and maximum range, after which the MDD is updated. The times taken to retrieve the new minimum and maximum KPI ranges for each adjustment were measured for both approaches and the experiment was repeated for 19 runs. Figure 4a shows the average time taken for each system to respond to a KPI adjustment. It is clear from this figure that the time taken to make these adjustments is significantly lower for the MDD approach than for the database approach.

**Experiment 2** The second experiment initially sets all KPI value ranges to their minimum and maximum. Then, the range of a single KPI is altered, after which each of the remaining 111 KPIs are incrementally altered one by one. Again, the MDD is updated after each alteration. In each case, the value of the adjustment is randomly chosen between the current minimum and maximum range of the KPI being altered.

Figure 4b shows the average time taken for each KPI choice to be made for the MDD and database approaches (averaged from 20 independent runs). For low numbers of choices made, the MDD approach requires significantly more time than the database approach due to the initial overhead of loading the MDD. However, once the number of KPI choices reaches 5 the MDD approach begins to significantly outperform the database approach.

The performance difference between the MDD and database approaches is further illustrated by Figure 4c, which shows the time taken for each system to respond as more and more choices are made. As the number of choices increases, the time taken for the database approach to find new minimum and maximum ranges for each KPI increases linearly, while the MDD approach remains almost static.

**Experiment 3** The final experiment replicates the setup of Experiments 1 and 2 but for each run, the schedule constraints are randomly perturbed. The goal of the experiment is to examine whether the advantages of the MDD approach are not limited to a single set of schedule constraints. The perturbations proceed as follows. A number of resources to perturb is selected at random and the value for each resource is multiplied by a number randomly chosen between 0.05 and 2.0. The perturbation step occurs for each run of the experiments.

The results in Figures 4d, 4e and 4f show that the approach maintains its advantage even under different initial constraints.

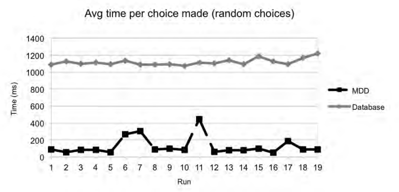
## 6 Discussion

An often occurring problem in real-world scheduling is the fact that there are multiple, conflicting objectives. Often, users find it hard to describe how the system should trade off one objective with another. The context of this work is a navy maintenance scheduling problem in which we ran into exactly this problem.

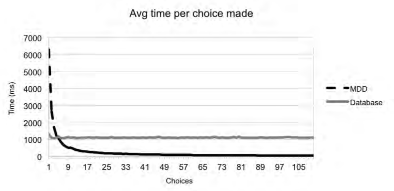
Configuration and recommender systems, such as described by (Hadzic et al. 2004; Nicholson, Bridge, and Wilson 2006; Andersen, Hadzic, and Pisinger 2010) have used compact compiled representations of sets of solutions. The main purpose of such representations is to allow fast interaction with the user, allowing extra (unary) conditions (including assigning a value to a variable) to be quickly added and retracted, and the consequences made visible to the user. In this paper we explore a similar system for presenting the possible schedules for the navy problem, based on MDDs as the representation, influenced in particular by the use of solution critiquing (McGinty and Reilly 2011; Ricci et al. 2011). The motivation for this kind of functionality seems even stronger for scheduling problems, because of the computational difficulties of solving scheduling problems, and the exponential number of schedules (even Pareto-optimal ones).

Our results show that the approach can work well on our problem. The MDD approach outperforms a database for the same task, which would be the obvious choice. This holds both for the original problem we faced, but also for random perturbations of that problem.

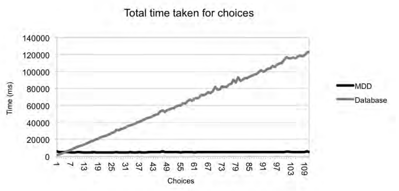
Future work includes a broader exploration of our methodology. Having shown promising results in one domain, we are keen to explore other types of scheduling problems to see if we can achieve the same results. We are also looking into cleverer ways of finding the initial set of solutions. This is a time-consuming step, so any reduction in the number of schedules we need to compute, while retaining the same or similar coverage would be a big benefit.



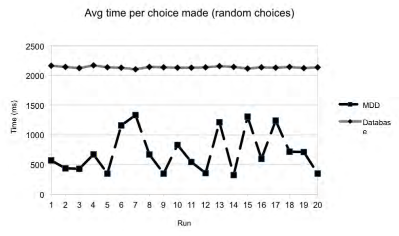
(a) Average time per choice made (random choices)



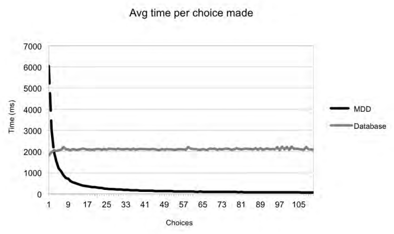
(b) Average time per choice made



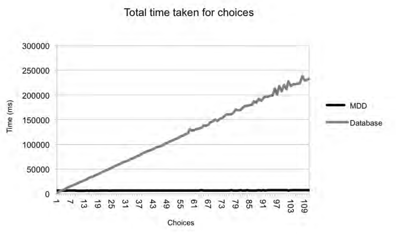
(c) Total time taken for choices



(d) Average time per choice made (random choices)



(e) Average time per choice made



(f) Total time take for choices

Figure 4: Experimental results. Figures (a), (b), and (c) are based on results from the actual problem; (d)–(f) are based on random variations of it

Finally, we are looking into a more advanced User Interface. We want to explore different ways of showing the relationships between KPIs (e.g. indicating which other KPIs are most influenced by making a choice for a particular KPI)

## Acknowledgements

This material is partly based on work supported by the Science Foundation Ireland under Grant TIDA I2000.

## References

Amilhastre, J.; Fargier, H.; and Marquis, P. 2002. Consistency restoration and explanations in dynamic CSPs—Application to configuration. *Artificial Intelligence (AI)* 135:199–234.

Andersen, H. R.; Hadzic, T.; and Pisinger, D. 2010. Interactive cost configuration over decision diagrams. *Journal of Artificial Intelligence Research (JAIR)* 37:99–139.

Berrada, I.; Ferland, J. A.; and Michelon, P. 1996. A multi-objective approach to nurse scheduling with both hard and soft constraints. *Socio-Economic Planning Sciences* 30:183–193.

Boyle, G.; Little, J.; Manning, J.; and van der Krogt, R. 2011. A constraint-based approach to ship maintenance for the irish navy. In *Proceedings of the Irish Transport Research Network Conference (ITRN-11)*.

Brucker, P. 2004. *Scheduling algorithms (4th edition)*. Springer.

Bryant, R. E. 1986. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers* 35(8):677–691.

Bryant, R. E. 1995. Binary Decision Diagrams and beyond: enabling technologies for formal verification. In *Proceedings of the 1995 IEEE/ACM international conference on Computer-aided design*, 236–243.

Fagerholt, K. 2004. A computer-based decision support system for vessel fleet scheduling–experience and future research. *Decision Support Systems* 37(1):35–47.

Hadzic, T.; Subbarayan, S.; Jensen, R. M.; Andersen, H. R.; Mueller, J.; and Hulgaard, H. 2004. Fast backtrack-free product configuration using a precompiled solution space representation. In *PETO Conference, DTU-tryk*, 131–138.

Johnston, M. D., and Giuliano, M. E. 2011. Multi-objective scheduling for space science missions. *Journal of Advanced Computational Intelligence and Intelligent Informatics (JACIII)* 15(8):1140–1148.

McGinty, L., and Reilly, J. 2011. On the evolution of critiquing recommenders. In Ricci, F.; Rokach, L.; Shapira, B.; and Kantor, P. B., eds., *Recommender Systems Handbook*. Springer. 419–453.

M.L.Pinedo. 2008. *Scheduling: Theory, Algorithms and Systems (3rd edition)*. Springer.

Nicholson, R.; Bridge, D. G.; and Wilson, N. 2006. Decision diagrams: Fast and flexible support for case retrieval and recommendation. In *ECCBR*, 136–150.

Ricci, F.; Rokach, L.; Shapira, B.; and Kantor, P. B., eds. 2011. *Recommender Systems Handbook*. Springer.

van der Krogt, R.; Little, J.; and Simonis, H. 2009. Scheduling in the real world: Lessons learnt. In *Proceedings of the ICAPS'09 Scheduling and Planning Applications workshop*.

Wilson, N. 2005. Decision diagrams for the computation of semiring valuations. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-05)*, 331– 336.

# Adaptive Route Planning for Metropolitan-Scale Traffic

**David Wilkie** and **Jur van den Berg**[*] and **Ming Lin** and **Dinesh Manocha**
University of North Carolina at Chapel Hill
E-mail: {wilkie, berg, lin, dm}@cs.unc.edu
http://gamma.cs.unc.edu/TROUTE

**Abstract**

Traffic congestion management is a global challenge that engineers, city planners, policy makers, and the public at large are likely to contend with for decades. It is unlikely that traditional physically-centered mitigation strategies by themselves will be successful or sustainable in the current economical and environmental climate. Numerous strategies have been proposed to construct Intelligent Transportation Systems (ITS), by incorporating sensing, information, and communication technologies in transportation infrastructure and vehicles. Through networks of sensors, recent cutting-edge efforts can provide real-time traffic monitoring, visualization, and some rather limited vehicle-based rerouting, but do not offer immediate, coordinated system-level relief to the traffic congestion problem.

In this position paper, we propose a concept on adaptive route planning to alleviate traffic congestion in a metropolitan area using stochastic route planning coupled with real-time traffic reconstruction. We demonstrate some of our early results on our traffic route planner that can more accurately predict future traffic conditions, which results in a reduction of the travel time for those vehicles that use our algorithm (Wilkie et al. 2011).

## Introduction

Traffic congestion management is a global challenge that engineers, city planners, policy makers, and the public at large are likely to contend with for decades. Besides the obvious energy and environmental impacts, traffic congestion imposes tangible costs on society. It is unlikely that traditional physically-centered mitigation strategies by themselves will be successful or sustainable in the current economical and environmental climate. Numerous strategies have been proposed to construct Intelligent Transportation Systems (ITS), by incorporating sensing, information, and communication technologies in transportation infrastructure and vehicles. Many of these efforts tend to perform off-line simulation and decoupled analysis. Most existing traffic simulations focus on either microscopic (e.g. agent-based simulation) or macroscopic (e.g. flow-like) behaviors; few have examined the intriguing interplay across different physical scales in a complex transportation system. Through networks of sensors, recent cutting-edge efforts can provide real-time traffic monitoring, visualization, and some limited vehicle-based rerouting, but do not offer immediate, coordinated system-level relief to the traffic congestion problem.

State-of-the-art route planners consider possible delays due to traffic congestion based on current traffic conditions and/or historical traffic data. Live traffic data can be collected by loop-detectors, cameras, toll port data, and cell phone localization. These systems provide the traffic velocity at certain locations at a fixed frequency (Brakatsoulas et al. 2005), which can then be used for vehicles to plan around congested areas. Live data alone does not enable predicting future traffic conditions. For example, if a route is planned to let a car arrive at a certain road in half an hour, the current conditions may no longer be an accurate estimate for that road 30 minutes later. This problem can be addressed by using a prediction scheme of the future traffic conditions based on historical probabilistic data of traffic conditions at similar times of the day under the similar weather (Horvitz et al. 2005), (Nikolova, Brand, and Karger 2006), (Min, Wynter, and Amemiya 2007).

However, given a large-scale system view of the entire road network and the traffic in the system, such an approach still has a problem: a route planner can affect future traffic conditions by planning for a large portion of the vehicles, thus making prediction based on current and historical data insufficient. Instead, the route planner must also take into account its own previous actions. For example, if a route planner controlled *every* car in the system, historically congested areas would be unduly avoided, causing congestion to appear at the routes that the planning system has provided. This is clearly the worst-case scenario, but the underlying problem is that the historical prediction assumes that cars tend to act the same way as they have historically, which may no longer be the case if a route planner is controlling the trajectories of all vehicles. We propose a novel *adaptive* traffic route planner that uses the routes of vehicles that it has planned based on current traffic condition to more accurately predict future traffic conditions for vehicles whose routes are subsequently planned (see Fig. 1). As a result, our approach overcomes the oscillation issue in case of large-scale adoption of traffic route planners.

[*]Now at University of Utah; email: berg@cs.utah.edu.

Our adaptive approach accounts for the fact that a route planned for a car will cause a little extra traffic density at the roads it will traverse. We use the predicted paths of the route planner itself in addition to historical data to estimate future traffic conditions. Assuming that a large percentage of the cars use the route planning system, the collection of all their planned routes can be used to accurately estimate the future traffic conditions. Every car that queries the route planner can then use this information to plan a route for itself. Its planned route is then used to update the estimate of future traffic conditions for vehicles come in later in the road network. Our experimental results suggest that our adaptive route planner can more accurately predict future traffic conditions, resulting in a reduction of the travel time for those vehicles that use our algorithm.

The rest of the paper is organized as follows. In Section 2, we discuss background work related to our approach. In Section 3, we detail the method used to update the historical probabilistic prediction and the overall planning system. In Section 4, we discuss the implementation and validation of our method.

## Prior Work and Background

Our work is perhaps most similar to that of (Nikolova et al. 2006) and (Lim et al. 2009). In fact, our work directly extends these methods to perform 'adaptive' routing. In (Nikolova et al. 2006), the authors propose a method to optimally route cars given uncertain information about travel times within a network. (Lim et al. 2009) provides an optimization to the procedure that allows fewer paths to be explored, while optimizing for a specific arrival deadline, and additional extensions were carried out in (Nikolova 2010) and (Hua and Pei 2010).

Another area of similar work is the study of Dynamic Traffic Assignment (DTA) done primarily in Civil Engineering. This problem involves flows of traffic from known origins to destinations (OD flows). The solution approaches attempt to optimally route all the flows in order to maximize aggregate or individual statistics. A summary of approaches can be found in (Peeta and Ziliaskopoulos 2001). The most relevant of these approaches are the simulation methods, such as (Florian, Mahut, and Tremblay 2008). In these approaches, cars are iteratively routed and simulated. The simulation provides the estimate of the network state that is used for the next iteration of routing. Over a number of iterations, the routes settle into an equilibrium.

Our work is inspired by (Lim et al. 2009), which presents an planning algorithm using graphs with stochastic (time-invariant) edge costs. Their planner assumes a cost function that invalidates the "optimal substructure" property of paths, which prevents using a straightforward A* algorithm, and present an efficient approach to compute optimal paths for such cost functions. In contrast, our algorithm uses a simpler cost function that still makes it possible to use an A* search algorithm, but assumes *time-varying* stochastic edge costs. We use insights from (Chabini and Lan 2010) regarding the first-in-first-out property of traffic that allows us to use A* even if the edge-costs are time-dependent (in general, time-dependent edge costs prohibit the use of A*). Typical traffic related planning approaches assume the edge cost to be given as travel times (potentially time-varying and stochastic) (Lim et al. 2009; Chabini and Lan 2010).

Our approach assumes the input data to be traffic *densities* of the road segments in the network, and we use the *fundamental diagram* of traffic to translate these densities to velocities and travel times. Maintaining densities allows us to update the data with the routes that our system generate to create an adaptive routing system. The observation that the flow (and velocity) of traffic was dependent on the traffic density was made in early traffic studies (Greenshields and others 1935). Since then, the concept has been used as a basis for continuum traffic simulation formulations (Siebel and Mauser 2005) as well as in schemes to estimate the state of traffic given sparse data, such as cell phone localization signals(Work et al. 2010).

## Approach

We assume the road network to be given as a directed graph $\mathcal{G} = (V, E)$ consisting of a set of vertices $V$ that model road intersections and edges $E \subset V \times V$ that model road segments between intersections. Associated with each edge $e$ is the capacity $C_e$, maximum speed $v_e^{\max}$, and length $\ell_e$ of the corresponding road segment. In addition, a stochastic function $\rho_e(t) \sim \mathcal{N}(\bar{\rho}_e(t), \tilde{\rho}_e(t))$ is maintained for all road segments $e$ that gives a normal distribution with mean $\bar{\rho}_e(t)$ and variance $\tilde{\rho}_e(t)$ of the traffic density of $e$ at time $t$. We assume this distribution is independent from the density distributions at other road segments or at other times (similar assumptions were made in (Lim et al. 2009)). Further, we assume that the time-axis is cyclical (e.g. with a daily or weekly period) and discretized into small steps, such that only a finite amount of data is stored with each edge $e$. The stochasticity of the density function models the uncertainty about future traffic conditions as well as the variability of conditions from day to day.

Our approach can be summarized as follows. We assume that over time, different queries for optimal routes come in from cars that use our adaptive planning system. If a query comes in from a car $i$ at a given time $t_0$, we plan a route for car $i$ between its start node $s$ and goal node $g$ that optimizes a cost function based on its expected travel time given the current density functions $\rho_e(t)$ for each edge $e$. Subsequently, assuming this car will actually follow the route it was given, we update the density functions $\rho_e(t)$ along its route such that its presence is accounted for when a route is planned for a subsequent car $i + 1$. This cycle continues indefinitely with each query coming in for an optimal route computation. As such, the planner is aware of the routes it has suggested earlier, in order to optimally estimate future traffic conditions.

We will first describe how an optimal route is planned for a car given the stochastic density functions $\rho_e(t)$. Next, we will discuss how this plan is used to update the stochastic density functions such that the presence of the car is accounted for in future plans for other cars. Finally, we will discuss how the problem of ”double-counting” cars can be

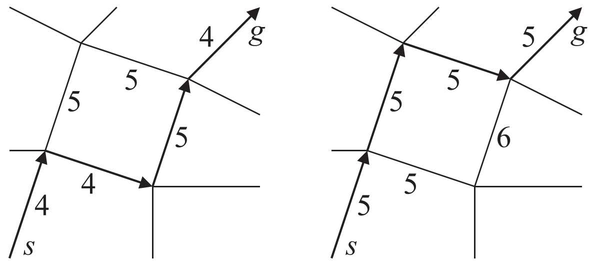


Figure 1: A schematic picture illustrating the idea of our approach. A road network is shown with edge costs. (a) If a route from $s$ to $g$ is requested, the optimal path is computed (shown with thick arrows). If the car follows this route, the densities and hence the edge costs along its path increases (in this case with 1). In (b), the network with the updated edge costs are shown. If a same query $(s, g)$ comes in from a subsequent car, it takes a different route (shown with thick arrows) to avoid the increased traffic densities. Note that this schematic picture does not illustrate the stochastic and time-varying aspects of our approach.

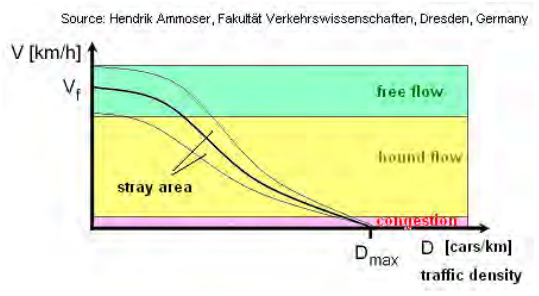


Figure 2: The fundamental diagram relating traffic density to travel speed.

avoided, which would occur when a car being routed also appears in the historical data.

## Route Planning

**Density and Travel Time** Given a query $(s, g, t_0)$ for a car between a start node $s \in V$ and a goal node $g \in V$ leaving $s$ at time $t_0$, we want to compute a route that minimizes the travel time to $g$ given the stochastic density functions $\rho_e(t)$. To relate density to travel time, we use the *fundamental diagram*, which is a well-known empirical concept in traffic simulation research (Greenshields and others 1935; Siebel and Mauser 2005; Work et al. 2010) that gives a mapping from the traffic density $\rho$ to the average speed $v$ on a road segment $e$, given the maximum speed $v_e^{\max}$ and capacity $C_e$ of the road segment $e$. Let the function described by the fundamental diagram be given by $v = f_e(\rho)$ (see Fig. 2).

Now, if a car arrives at the beginning of a road segment $e$ at time $t$, we assume the speed with which it can traverse the road segment is given by $f_e(\rho_e(t))$. The travel time $\tau_e(t)$ to traverse $e$ starting at time $t$ is then given by:

$$\tau_e(t) = \frac{\ell_e}{f_e(\rho_e(t))}. \tag{1}$$

Since, the density function $\rho_e(t)$ is stochastic, the travel time is stochastic too. We approximate it with a normal distribution as $\mathcal{N}(\bar{\tau}_e(t), \tilde{\tau}_e(t))$, with mean $\bar{\tau}_e(t)$ and variance $\tilde{\tau}_e(t)$ given by the first-order Taylor expansion of $\tau_e(t)$:

$$\bar{\tau}_e(t) = \frac{\ell_e}{f_e(\bar{\rho}_e(t))} \tag{2}$$

$$\tilde{\tau}_e(t) = \big(\frac{d\tau_e(t)}{d\rho}[\bar{\rho}_e(t)]\big)^2 \tilde{\rho}_e(t). \tag{3}$$

For a path $\pi = \{e_1, \ldots, e_n\}$ consisting of a series of road segments when travel is commenced at time $t_0$, the total travel time $\tau_\pi(t_0)$ is given recursively by:

$$\tau_{\{e_1\}}(t_0) = \tau_{e_1}(t_0) \tag{4}$$

$$\tau_{\{e_1,\ldots,e_k\}}(t_0) = \tau_{\{e_1,\ldots,e_{k-1}\}}(t_0) + \tau_{e_k}(t_0 + \tau_{\{e_1,\ldots,e_{k-1}\}}(t_0)) \tag{5}$$

Its mean $\bar{\tau}_\pi(t_0)$ and variance $\tilde{\tau}_\pi(t_0)$ are hence given by:

$$\bar{\tau}_{\{e_1\}}(t_0) = \bar{\tau}_{e_1}(t_0) \tag{6}$$

$$\tilde{\tau}_{\{e_1\}}(t_0) = \tilde{\tau}_{e_1}(t_0) \tag{7}$$

$$\bar{\tau}_{\{e_1,\ldots,e_k\}}(t_0) = \bar{\tau}_{\{e_1,\ldots,e_{k-1}\}}(t_0) + \bar{\tau}_{e_k}(t_0 + \bar{\tau}_{\{e_1,\ldots,e_{k-1}\}}(t_0)) \tag{8}$$

$$\tilde{\tau}_{\{e_1,\ldots,e_k\}}(t_0) = \tilde{\tau}_{\{e_1,\ldots,e_{k-1}\}}(t_0) + \tilde{\tau}_{e_k}(t_0 + \bar{\tau}_{\{e_1,\ldots,e_{k-1}\}}(t_0)). \tag{9}$$

**Cost Function** Our objective is to find a route $\pi$ that minimizes the expectation $\mathbb{E}[c(\tau_\pi(t_0))]$, given a cost function $c(\tau)$ on the travel time $\tau$. We consider two cases here:

- *Linear cost:* The cost increases linearly with the travel time: $c(\tau) = \tau$. Let $pdf_A(t)$ denote the probability density function of normal distribution $A$. Then, the expected cost is given by

  $$\mathbb{E}[c(\tau_\pi(t_0))] = \int_{-\infty}^{\infty} pdf_{\tau_\pi(t_0)}(t) \cdot t\, dt = \bar{\tau}_\pi(t_0),$$

  which is the mean of the travel time of route $\pi$ when travel is commenced at time $t_0$.
- *Exponential cost:* The cost increases exponentially with the travel time to more heavily penalize late arrivals: $c(\tau) = \exp(2w\tau)$ for some weight parameter $w$. The expected cost in this case is given by

  $$\mathbb{E}[c(\tau_\pi(t_0))] = \int_{-\infty}^{\infty} pdf_{\tau_\pi(t_0)}(t) \exp(2wt)\, dt = \exp(\bar{\tau}_\pi(t_0) + w\tilde{\tau}_\pi(t_0)).$$

  The result of minimizing for $\mathbb{E}[c(\tau)]$ in equivalent to minimizing for $\log \mathbb{E}[c(\tau)]$. Following this, the cost then becomes $\bar{\tau}_\pi(t_0) + w\tilde{\tau}_\pi(t_0)$, and is hence a linear combination of the mean and the variance of the travel time (Lim et al. 2009).

Our approach works for either of these cost functions. In our implementation we used the exponential cost function, for it attempts to minimize both the mean and the variance of the travel time.

**Planning Algorithm** To find a path in the graph $\mathcal{G}$ between start node $s$ and goal node $g$, we are confronted with a shortest path problem in a graph with time-varying and stochastic edge costs. In general, such problems are hard (Lim et al. 2009; Chabini and Lan 2010), but in our case we can exploit properties of the cost function that allow us to a standard A* algorithm, which we will slightly adapt to handle the stochastic travel times.

Firstly, both of the cost functions as defined above are *additive* given the way the mean and variance of the travel time are computed (see Equations (8) and (9)). That is,

$$\mathbb{E}[c(\tau_{\{e_1,\dots,e_k\}}(t_0))] = \mathbb{E}[c(\tau_{\{e_1,\dots,e_{k-1}\}}(t_0))] + x, \quad (10)$$

where $x$ is a linear combination of the second terms of Equations (8) and (9). Second, traffic observes the so-called *first-in-first-out* property (Chabini and Lan 2010). This means that arriving earlier at a node $u$ in the graph will never produce a costlier route than a route that arrives later at $u$. Note that this is not the case for graphs with general time-dependent edge costs.

These two properties allow us to use the standard A* algorithm, which is adapted to handle the stochastic travel times along a route. The algorithm is given in Fig. 3. Instead of maintaining a single cost value of each node $u$ in the graph as in standard A*, we maintain both the mean $\bar{\tau}_u$ and variance $\tilde{\tau}_u$ of the travel time of the current-best route from $s$ to $u$. Initially, these are infinity for all nodes $u$, except the start node $s$, for which they are zero. The heuristic value $\bar{h}(u)$ provides a lower-bound estimate of the mean travel time to the goal $g$ from a given node $u$, for which we use the Euclidean distance between $u$ and $g$ divided by the largest maximum speed in the road network. The heursitic value $\tilde{h}(u)$ provides a lower-bound estimate of the variance of the travel time between $u$ and $g$, for which we use $\tilde{h}(u) = 0$. The functions $\bar{\tau}_e(t)$ and $\tilde{\tau}_e(t)$ which we refer to in lines 9 and 10 are given by Equations (2) and (3).

```
TRAFFICA*$(s, g, t_0)$
 1: $\forall u \in V : \bar{\tau}_u \leftarrow \infty, \tilde{\tau}_u \leftarrow \infty; \bar{\tau}_s \leftarrow 0; \tilde{\tau}_s \leftarrow 0$
 2: $OPEN \leftarrow \{s\}$
 3: while $OPEN \neq \emptyset$ do
 4:     $u \leftarrow \arg\max_{u \in OPEN}\{\bar{\tau}_u + \bar{h}(u) + w(\tilde{\tau}_u + \tilde{h}(u))\}$
 5:     $OPEN \leftarrow OPEN \setminus \{u\}$
 6:     if $u = g$ then
 7:         return
 8:     for each edge $e = (u, v)$ in $\mathcal{G}$ do
 9:         $\mu \leftarrow \bar{\tau}_e(t_0 + \bar{\tau}_u)$
10:         $\sigma \leftarrow \tilde{\tau}_e(t_0 + \bar{\tau}_u)$
11:         if $\bar{\tau}_u + \mu + w(\tilde{\tau}_u + \sigma) < \bar{\tau}_v + w\tilde{\tau}_v$ then
12:             $\bar{\tau}_v = \bar{\tau}_u + \mu$
13:             $\tilde{\tau}_v = \tilde{\tau}_u + \sigma$
14:             pred$(v) \leftarrow u$
15:             $OPEN \leftarrow OPEN \cup \{v\}$
```

Figure 3: The modified A* algorithm to compute an optimal route with respect to the exponential cost metric between start node $s$ and goal node $g$ when traffic is commenced at time $t_0$. When planning has finished, the optimal route is inferred by following the backpointers from the goal $g$.

## Maintaining Density Functions

Once a route has been planned for a car, we wish to take its presence into account when subsequent routes are planned for other cars. Based on the route that has been suggested, we can assume the car will follow it and add to the traffic densities at the road segments along its route at the times it is expected to traverse these road segments. At the same time, not all cars on the road use our adaptive planning system, and the system is not aware of the future plans of the cars that do use our system but have not entered the road network (yet). So, existing planned routes alone do not provide an accurate estimate of future traffic data.

**Blending Historical and System Data** In order to predict future traffic conditions, we let the density functions $\rho_e(t)$ used in the above algorithm be a combination of historical traffic density data $\rho_e^{\text{hist}}(t)$, and density data $\rho_e^{\text{syst}}(t)$ generated by route plans provided by our planning system. However, care needs to be taken that the historical data is partly *phased out* when actual data of planned routes is included in the densities, as to avoid cars being double counted. We proceed as follows. Let $\alpha \in [0, 1]$ be the proportion of cars that use our system to compute their routes. Further, let there be a function $\beta(\Delta t) \in [0, 1]$ that provides the proportion of cars that will be on the road at $\Delta t$ time into the future which are already on the road currently. We assume $\beta(\Delta t)$ is time-independent, and can be inferred from historical traffic data on average travel times.

The traffic density $\rho_e(t)$ as used in our algorithm for a car starting travel at time $t_0$ is then computed as follows:

$$\rho_e(t) = (1 - \alpha\beta(t - t_0))\rho_e^{\text{hist}}(t) + \rho_e^{\text{syst}}(t). \quad (11)$$

This can be explained by the fact that a fraction $\alpha\beta(t - t_0)$ of all cars that will be on the road at time $t$ have already been accounted for at time $t_0$ in the densities generated by our system.

**Updating Traffic Densities** When a route has been planned for a car by our algorithm, we wish to take its presence into account when subsequent routes are planned for other cars. To this end, we update the density data $\rho_e^{\text{syst}}(t)$ that only counts cars for which routes have been planned using our adaptive route planning system. We update these traffic densities as follows.

The algorithm above will give us a route $\pi = (e_1, \dots, e_n)$ and distributions $\tau_u \sim \mathcal{N}(\bar{\tau}_u, \tilde{\tau}_u)$ of the travel times from the start node $s$ to each node $u$ along path $\pi$. Let edge $e = (u, v)$ be part of $\pi$. The car for which a path is planned will be on $e = (u, v)$ at time $t$ with probability:

$$q_{(u,v)}(t) = \int_{-\infty}^{t-t_0} pdf_{\tau_u}(t')\,dt' \cdot \int_{t-t_0}^{\infty} pdf_{\tau_v}(t')\,dt', \quad (12)$$

where $t_0$ is the time at which the car commences its route $\pi$.

The above equation computes the probability that the car both arrives at $e$ before time $t$ and leaves $e$ after time $t$. The density on $e$ at time $t$ is defined by the number of cars on $e$ at time $t$ divided by the length $\ell_e$ of $e$. Hence, the distribution of the density $\rho_e^{\text{syst}}(t)$ at time $t$ is updated as follows:

$$\bar{\rho}_e^{\text{syst}}(t) \leftarrow \bar{\rho}_e^{\text{syst}}(t) + q_e(t)/\ell_e \tag{13}$$

$$\tilde{\rho}_e^{\text{syst}}(t) \leftarrow \tilde{\rho}_e^{\text{syst}}(t) + q_e(t)(1 - q_e(t))/\ell_e^2, \tag{14}$$

which follows from treating the distribution $\rho_e(t)$ for each edge $e$ and for each time $t$ as an independent Poisson binomial distribution consisting of a number of cars each with a different probability of contributing to the density. When the number of cars gets large, the Poisson binomial distribution is well approximated by a normal distribution $\mathcal{N}(\bar{\rho}_e(t), \tilde{\rho}_e(t))$. This justifies the assumption in the planning algorithm of Fig. 3 that $\rho_e(t)$ is a normal distribution.

As the time axis is discrete, we only need to update a finite set of density distributions along the route planned for the car. We use the same discretization to compute the integrals in Equation (12). We use the updated mean and variances for the densities to route subsequent cars. This cycle of routing cars and updating densities continues indefinitely.

## Empirical Results

In this section, we present our empirical study of the performance of our approach. Our hypothesis is that our system plans routes that have, on average, lower travel times than routes planned using the shortest path metric or stochastic-historical prediction method, increasingly so when the proportion of users of our system increases. This reinforces the essential claim of our paper: by taking into account the routes planned by the system itself, a planning system can find routes with substantially shorter travel time.

We have validated our approach by calculating plans for a fixed population of cars and queries using varying route planning methods. The validation was done in four parts. First, we compare our method with using a single path in a network. Next, we compare the performance of our adaptive route planning algorithm to using shortest path A* searching. Next, we compare our algorithm to using stochastic-historical prediction. Finally, we investigate the performance of our algorithm as the percentage of total number of cars that are controlled varies.

### Traffic Simulation

To simulate the travel times of the cars in these experiments, we use the same derivation as above. We calculate the estimated travel times using the fundamental diagram and the time-varying density data. This density data is then updated for every car that travels the network. For these experiments, we have added a *cutoff-capacity* to our road network edges. This is the maximum density value the edge will be assigned, regardless of how many cars are routed on it. This ensures that the planners we compare against do not plan routes with infinite time duration.

### Avoiding Congestion on a Single Path

We designed this benchmark to showcase the basic premise of our approach. The road network is a rectangular grid of 5 $\times$ 5 intersections, connected by a road segments with equal maximum speeds, 22.35 m/s, capacities, 0.09 cars per meter,

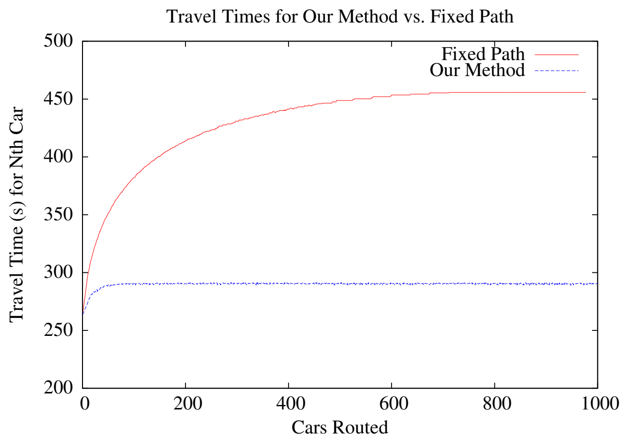


Figure 4: We highlight the performance of our algorithm compared with routing cars along a single path. The flow of cars quickly leads to congestion and long travel times for the single path. Our approach distributes the cars and settles to a constant travel time.

and cut-off capacities, 0.085 cars per meter. Each road segment is approximately 1000m. We assume the road network is initially empty (i.e. there is no traffic). Then, we begin routing a set $S$ of cars, each defined by a tuple $(s, g, t)$ of a starting vertex $s$, an ending vertex $g$, and a starting time $t$, enter the road network and traverse the route given by a route planner. We assign the starting vertex for each car to be the bottom left corner of the grid and the goal to be the top right corner. Each car is given a starting time $t = 2 * i$ seconds, where $i$ is the car index in $S$.

We ran this experiment for both planners, the route planner that simply returns the shortest path between the start and goal vertex, independent of traffic conditions, and our adaptive route planning system.

The result of this can be seen in Figure 4. Obviously, since all cars have the same start and goal vertices, they are all assigned the same route by the shortest-path planner, quickly causing growing congestion on this route. The incoming flow is enough to cause significant congestion, but not to cause a complete traffic jam. Our method distributes the cars along multiple paths to the goal based on densities predicted by earlier planned routes. By doing so, it can handle the flow of vehicles at a relatively constant travel time.

### Comparison with Shortest Path Planner

In the second benchmark, we compare the behavior of our algorithm to the shortest path planner while routing a set of cars $S$ with random initial and goal intersections. This scenario takes place on a grid road network with 15 $\times$ 15 intersections and a road length of  100m. The road network is initially empty. As above, we perform the experiment for both planners: first, we route each car in $S$ using a shortest path planner and calculate the resulting travel times. Second, we do the same assuming all cars are routed using our route planner. To best illustrate the effect of network load in this scenario, each car has a starting time of zero. The parame-

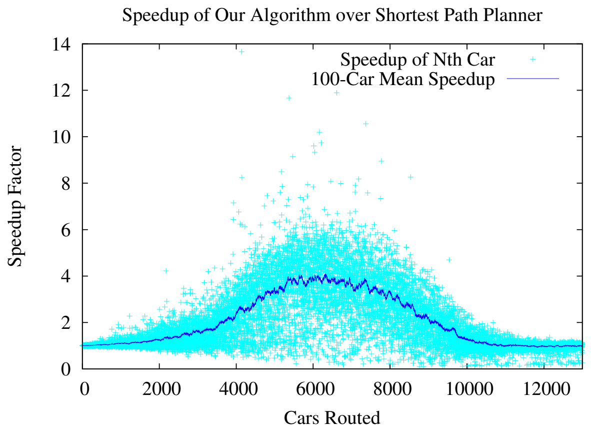


Figure 5: This figure shows the speedup factor that our method achieves over a shortest path planner for a series of cars. Each car has a random start and goal position.

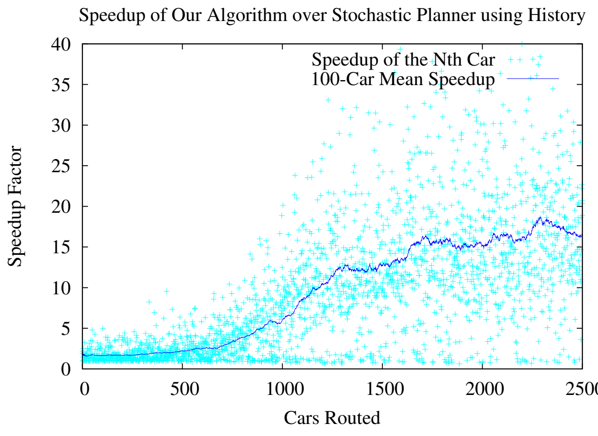


Figure 6: This figure shows the speedup factor (up to 20 for the 100-car mean) that our method achieves over a stochastic planner using only historical data for a series of cars, indexed from 0 to 2500. Each car has a random start and goal position.

ters used for the experiment were capacity = 0.09 cars per meter, maximum velocity = 22.35 m/s, and cut-off capacity = 0.085. For storing the density information for each edge, a time discretization of 15s was used.

Figure 5 shows the result of this experiment. The result is given in terms of the speedup of our method over the shortest path planner, i.e. the ratio of the travel time planned by the shortest path planner over the travel time planned by our method. The speedup for every car routed is displayed as the cyan scatter plot, and the average speedup factor for each cohort of 100 cars is displayed as the blue line. As can be seen, initially the speedup factor is negligible, since the road network has low traffic densities that do not significantly slow down traffic. Hence, in these cases the shortest path is indeed also the time-optimal path. However, as more an more cars have entered the road network, the average speedup in travel time by using our planner rather than a shortest path planner increases, peaking at a factor of approximately 4 after 6,000 cars have been planned. Eventually, the average speedup decreases again, since the road network has become so congested that alternative routes do not provide any benefit in terms of travel time.

## Comparison with Stochastic Planner Using Historical Data

In this experiment, we compare the behavior of our algorithm to a stochastic planner using historical data (SPUHD) when routing a population of cars $S$. This experiment takes place on the same grid road network as above, with 15 $\times$ 15 intersections. We generate *historic* traffic data by defining a set of cars $S$ with random start and goal intersections, routing each car in $S$ using a shortest path planner, and calculating the resulting densities. The cars are created in 40 batches of 200, with each batch having a starting time of 5 seconds later than the preceding batch. This creates a maximum average network density, using the shortest path routing, of 0.054 cars per meter, and areas of full congestion, i.e. 0.085 cars per meter. These time-varying, edge specific stochastic densities are the *history* for the scenario. We assume that $S$ represents the typical traffic flow. We compare our method and the SPUHD by planning for all of $S$, given the calculated stochastic *history*. In our routing algorithm, the planned routes are used to update the traffic densities for plans of future cars, but our planner is unaware of the *history*. In the SPUHD, each car in $S$ is planned for assuming the *history* is a valid prediction, and these predicted densities are not updated based on the SPUHD's planned routes.

The results are shown in Figure 6. We can see that only using the stochastic *history* is not an effective strategy when all cars are being navigated by the planner. This is an extreme example, but it illustrates a basic motivating problem with using stochastic prediction. If the planner were routing one car in $S$, then the *history* would be almost perfect; as the planner is routing all of $S$, the predictions based on the *history* are not valid. As the SPUHD *believes* a certain congestion pattern will occur, due to the *history*, it routes cars around that congestion pattern. However, as the SPUHD is controlling all the cars, the predicted congestion pattern does not occur, but instead the planner creates congestion in other areas. The SPUHD assigns cars sub-optimal routes due to its belief that the typical, historical traffic flows will remain constant. Our method, on the other hand, distributes car routes and achieves a 100-car mean speedup of up to a factor of 20.

The high speedup factors here illustrate how unsuitable pure historical prediction is when the entire set of cars is being routed. In attempting to avoid predicted densities, the stochastic planner using historical data irrationally prefers domains of the road network, which then causes congestion and traffic jams in those areas. Even at low network densities, approximately 0.014 cars per meter, the SPUHD causes traffic jams, with some roads being saturated to their cutoff capacity, 0.085 cars per meter.

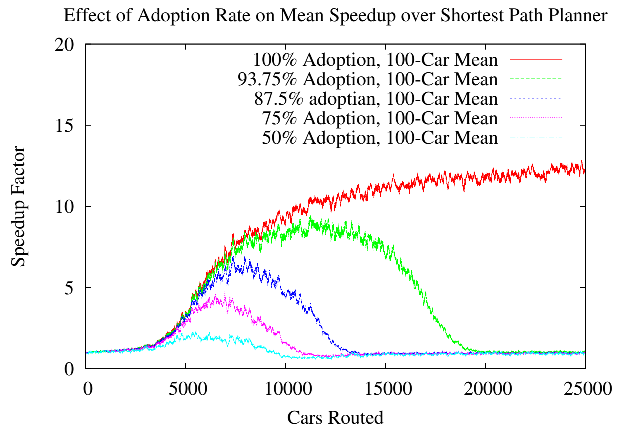


Figure 7: The 100-car mean speedup of our method over the simple planner for varying adoption percentages.

## Effect of Adoption Rate

In our final benchmark, we analyze the effect of the adoption rate of our system in scenarios in which part of the cars use our route planner system, and the other part of the cars use a shortest path planner. We let a set $S$ of cars with random start and goal intersections enter the (initially empty) road network at a rate such that over time heavy congestion is likely to be created on the road network. We use the same road network as above. A percentage $\alpha$ of the cars (randomly sampled from $S$) use our adaptive traffic route planner to plan their routes, whereas $1 - \alpha$ of the cars use a shortest path planner. For the sake of the simplicity, our adaptive route planner ignores the portion of cars it does not control; in reality this data can be estimated from historical data (see Section **Blending Historical and System Data**). We repeat this experiment for various values of $\alpha$. The proportion of cars, $\alpha$, that are routed by our method are chosen using a consistent random number seed: this implies that a car routed for a low $\alpha$ will also be routed for a high $\alpha$, preserving features of the graphs for each value of $\alpha$. The cars enter the road network at a rate of 50 per second.

Figure 7 shows the results, a graph depicting the 100-car mean speedup for various adoption values from $50\%$ to $100\%$. We see a peek speedup of over 10 when $100\%$ of the cars are controlled: our system avoids the creation of heavy congestion and large scale traffic jams. However, the maximum speedup and the integral of the speedup curve decrease rapidly with $\alpha$, showing a strong sensitivity to uncontrolled cars creating traffic jams. At the $50\%$, a maximum speedup of 2 is observed, and for smaller $\alpha$ values, a similarly small speedup is observed. These results clearly show that the benefit of our system increases with the adoption rate.

## Conclusion and Future Work

State of the art approaches can handle stochastic planning and can make use of traffic predictions, but they ignore a useful source of information, i.e. the existing planned routes. As routing systems become more pervasive, the routes they plan will begin to significantly influence the future state of traffic. Prior plans then become relevant to future plans. Our approach addresses this issue. We provide a method to update stochastic traffic predictions with existing planned routes. Given stochastic predictions of future traffic states, we plan for cars within this space. For each path planned, we update the stochastic predictions based on the routes planned for each car. The density of each edge is updated according to the estimated arrival and departure times for the car. The velocity of each edge is then updated according to the fundamental diagram, an empirical relationship between density and velocity. In our simulations, the improved routing algorithm results in better utilization of the road network, reduces congestion and the travel time for each car.

There are many avenues for future work. We would like to perform more analysis and validate the performance of our algorithm on actual traffic data. It would be useful to relax some of our assumptions in terms of normal distributions along each edge of the road network. Finally, it may be useful to develop a decentralized version of our adaptive traffic planning algorithm.

## Acknowledgments

This work was supported in part by ARO Contract W911NF-10-1-0506, NSF awards 0917040, 0904990 and 1000579, and RDECOM Contract WR91CRB-08-C-0137. The earlier version of this paper was presented at AAAI 2011 (Wilkie et al. 2011).

## References

Brakatsoulas, S.; Pfoser, D.; Salas, R.; and Wenk, C. 2005. On map-matching vehicle tracking data. In *Proceedings of the 31st international conference on Very large data bases*, 853–864. VLDB Endowment.

Chabini, I., and Lan, S. 2010. Adaptations of the A* Algorithm for the Computation of Fastest Paths in Deterministic Discrete-Time Dynamic Networks. *IEEE Transactions on Intelligent Transportation Systems* 3(1):60–74.

Florian, M.; Mahut, M.; and Tremblay, N. 2008. Application of a simulation-based dynamic traffic assignment model. *European Journal of Operational Research* 189(3):1381–1392.

Greenshields, B., et al. 1935. A study of traffic capacity. In *Highway Research Board Proceedings*, volume 14, 448–477.

Horvitz, E.; Apacible, J.; Sarin, R.; and Liao, L. 2005. Prediction, Expectation, and Surprise: Methods, Designs, and Study of a Deployed Traffic Forecasting Service. *Conf. on Uncertainty in Artificial Intelligence*.

Hua, M., and Pei, J. 2010. Probabilistic path queries in road networks: traffic uncertainty aware path selection. In *Proceedings of the 13th International Conference on Extending Database Technology*, 347–358. ACM.

Lim, S.; Balakrishnan, H.; Gifford, D.; Madden, S.; and Rus, D. 2009. Stochastic Motion Planning and Applications to Traffic. *Algorithmic Foundation of Robotics VIII* 483–500.

Min, W.; Wynter, L.; and Amemiya, Y. 2007. Road traffic prediction with spatio-temporal correlations. In *Proceedings of the Sixth Triennial Symposium on Transportation Analysis, Phuket Island, Thailand (June 2007)*.

Nikolova, E.; Kelner, J.; Brand, M.; and Mitzenmacher, M. 2006. Stochastic shortest paths via quasi-convex maximization. *Algorithms–ESA 2006* 552–563.

Nikolova, E.; Brand, M.; and Karger, D. 2006. Optimal route planning under uncertainty. In *Proceedings of International Conference on Automated Planning and Scheduling*.

Nikolova, E. 2010. High-Performance Heuristics for Optimization in Stochastic Traffic Engineering Problems. *Large-Scale Scientific Computing* 352–360.

Peeta, S., and Ziliaskopoulos, A. 2001. Foundations of dynamic traffic assignment: The past, the present and the future. *Networks and Spatial Economics* 1(3):233–265.

Siebel, F., and Mauser, W. 2005. On the fundamental diagram of traffic flow. *Arxiv preprint cond-mat/0503290*.

Wilkie, D.; van den Berg, J.; Lin, M. C.; and Manocha, D. 2011. Self-aware traffic route planning. *Proc. of AAAI*.

Work, D.; Blandin, S.; Tossavainen, O.; Piccoli, B.; and Bayen, A. 2010. A traffic model for velocity data assimilation. *Applied Mathematics Research eXpress*.

# Constraint based Berth Allocation and Ship Scheduling with Upstream Supply Planning

**S Kameshwaran** and **Alfiya Tezabwala** and **Vinayaka Pandit**
IBM Research - India, Bangalore
{kameshwaran.s, alfiya.tezabwala, pvinayak}@in.ibm.com

**Abstract**

Allocation of ships to berths significantly contributes to the operational efficiency and revenue at the ports. Popularly called as the *berth allocation problem*, many of its variants have been solved using scheduling algorithms and math programming techniques in the literature. Our work is motivated by a real world scenario where the port belongs to a mining company. In this case, the berth allocation is equivalent to scheduling of demand fulfilled by the mining company. The scheduling of demand is contingent on the upstream supply of materials from the mine area to the port, through a rail network. Given a schedule of incoming ships with required demand for next two to three weeks, our problem is to allocate berths and schedule the ships such that the required demand can be fulfilled with appropriate upstream supply planning. The problem has both the features of detailed scheduling at the port and aggregate multi-period, multi-stage planning of the upstream supply chain. In this paper, we propose a constraint based modeling for the integrated planning-scheduling problem using IBM ILOG Constraint Programming Optimizer.

## Introduction

The berth allocation problem is concerned with the allocation of berth space to ships in the ports. Many variants of the problem have been studied based on one or more of the following cases:

- *Berth layout*: Discrete, continuous, hybrid;
- *Ship arrival time*: Static, dynamic;
- *Ship handling time*: Static, dynamic;
- *Port type*: Container, bulk;

The berth layout can be classified as discrete, continuous, and hybrid (Bierwirth and Meisel 2010). Discrete berth layout defines a finite set of berths to which ships can be *allocated*, whereas a continuous case defines continuous berthing space in which the ships can be *packed*. Port with a hybrid layout will have both the discrete set of berths and continuous berth space. In the static ship arrival problem, ships are already at the port, whereas in the dynamic case, only a subset of the ships to be scheduled are present. In the static handling time problem, ship handling times are considered as input, whereas in the dynamic case, they are decision variables. In a container port, the cargo to be loaded or unloaded are packed inside standard containers. Thus the handling is independent of the cargo type and will not constrain the ship berth allocation. On the other hand, bulk terminals use different types (dry, liquid) and varieties (minerals, food, oil, gas) of cargo that require specialized equipments and storage areas, which constrain the feasibility of berthing a ship depending on the cargo.

In general, the berth allocation problem is NP-hard (Imai, Nishimura, and Papadimitriou 2001; Umang, Bierlaire, and Vacca 2011). Math programming is the predominant solution approach (Buhrkal et al. 2011), while constraint programming has also been showed to be successful (Kelareva et al. 2012). The berth allocation problem we consider in this paper is for a bulk port with discrete berth layout, dynamic ship arrival time, and static handling time. The problem we consider in this paper is motivated by mining supply chains with their own ports for private use.

Some large mining companies own the end-to-end supply chain starting from the mines, rail network, and the port. The bulk ore from the mine is moved through the rail and stocked at the port. Ships arrive with customers demands and the port schedules the ships on the berth and loads the required orders. The main difference in the problem considered here is that the mining company (port) has to make sure that there is enough bulk in the port yard to fulfill the demand. This is not the case in other ports, where the availability of cargo to be loaded and removal of the unloaded cargo from the port are handled by the customers. Also, note that there is no unloading of cargo in our case. Thus, the problem of berth allocation in this case is equivalent to scheduling of demand fulfillment, which is contingent on the availability of the materials stocked at the yard.

The berth allocation is usually solved for a horizon of two to three weeks. With limited capacity of the yard at the port for stocking, the bulk needs to be continuously replenished from the mine area through the rail. There are multiple bulk ore types, with different storage location and capacity in the yard. Hence the choice and quantity of the bulk to be moved from the mine area to the port should closely be synchronized with the fulfillment of the demand at the port.

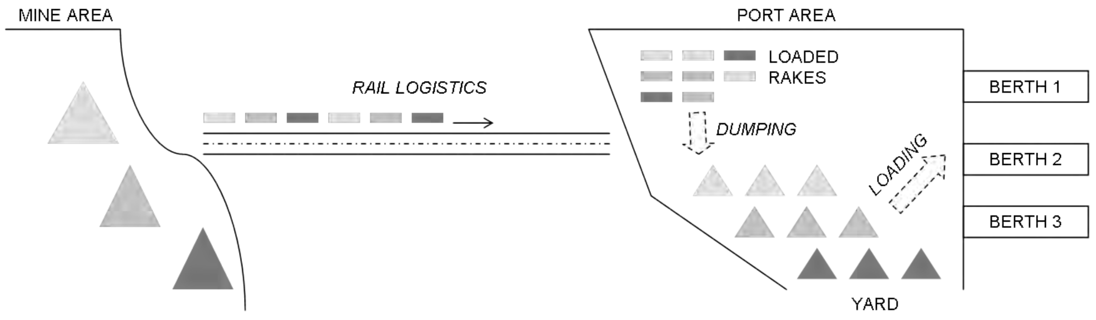


Figure 1: Schematic of moving bulk from mine to port

We consider the berth allocation problem at the port, integrated with the upstream planning of supply from the mines. The upstream supply planning is done at aggregate level from mine area to port, whereas the berth allocation is detailed scheduling of ships at the port. Hence our problem is *integrated* planning and scheduling, where the integration is across vertical stages in the supply chain. We propose a constraint based modeling for the above problem using IBM ILOG CP Optimizer.

## Berth Allocation with Upstream Supply Planning

The schematic of the mining supply chain that moves the bulk ores from mine to ships is illustrated in figure 1. Our primary interest is in the berth allocation and ship scheduling problem at the port. Given the set of ship arrivals in the two to three weeks horizon, the berth allocation problem schedules the ships on the berths and loads the ships with the demanded materials. Failure to schedule and load a ship results in penalty for not fulfilling the demand. Also the port is liable to pay penalty if there is a delay in the loading of ship beyond its demurrage time. Thus the berth allocation problem directly influences the port utilization and revenue.

From the perspective of the mining company, the berth allocation problem is equivalent to demand scheduling and fulfillment. The bulk to be loaded on to the ships is stored in the open yard at the port area. The capacity of yard can only cater to few days of demand. Hence, for a horizon of two to three weeks, the materials required should be continuously replenished at the yard in the port area from the mine. There are different types of materials in the supply chain and hence the upstream supply from the mine to the port will be driven by the demand at the port. Similarly the scheduling of demand fulfillment should take into account the upstream supply capability. The problem we study in this paper is motivated by this real world scenario of berth allocation problem in the mining industry.

The movement of materials from the mine to the port is a series of following interdependent scheduling problems: A) rail logistics, B) yard operations (dumping of bulk from rail rakes to yard and loading of bulk to the ships), and C) berth allocation. Each of the above scheduling problems is individually hard. Thus an end-to-end integrated scheduling across the entire supply chain would be challenging. Further, some mining companies use third party rail services and hence cannot schedule the rail operations. Taking into account the above factors, we define a new variant of the berth allocation problem - *berth allocation with upstream supply planning*. The upstream supply from the mine is not handled at detailed operations scheduling level, but at *aggregate planning* level.

### Detailed Scheduling versus Aggregate Planning

There are two key differences between detailed scheduling and aggregate planning. Firstly, the unit time in the horizon is different - detailed scheduling is at the granularity of minutes, whereas the aggregate planning is done at hours. Secondly, scheduling takes into account all the resources and tasks. The aggregate planning ignores the finer details. For example, consider the rail logistics stage that moves the bulk from mine area to port area. The operations scheduling of rail logistics should take into account the following: number of rakes available at the mine, loading of bulk from mine to the rakes, formation of trains with rakes, minimum time gap between two trains, number of trains in transit, etc.

On the other hand, the *aggregate planning* problem for rail logistics can be expressed by a single planning parameter - *number of rakes that can be supplied from the mine per planning period*. In order to meaningfully approximate the detailed scheduling constraints by a single planning parameter, the planning period should be long enough. The planning period of six hours and above is usually used in practice.

Aggregate planning for the upstream supply from mine area to port area thus replaces large number of finer scheduling constraints with few planning parameters, thereby reducing the problem complexity. The plan generated at this level is then used to drive the dedicated rail scheduling system that comes with a detailed schedule to meet the plan. The multilevel approach (Caimi et al. 2011) in rail logistics is popular in planning and scheduling of manufacturing as *hierarchical decomposition* (Pinedo 2009, Ch. 8). In the reminder of the paper, we use *planning* and *aggregate planning* interchangeably.

### Upstream Supply Planning

The upstream supply planning problem for the supply chain depicted in figure 1 consists of three activities: A) rail logistics, B) dumping, and C) loading, where the dumping and loading happen at the port area. There are three inventories in the supply chain: A) yard in the mine area, B) loaded rakes in the port area, and C) yard in the port area. We briefly describe them below.

**Rail Logistics** The mine and ports could be separated by hundreds or even thousands of miles requiring several hours or days of transportation time. The transportation through rail, carries the bulk in rakes of fixed capacities. The aggregate planning parameter is the *number of rakes supplied from the mines per period*. The decision to be taken is the number of rakes per material that can be supplied from the mines per planning period. Note that the reverse logistics of the trains back to the mine need not be considered at planning level. If the third party rail service is used for transportation, then the service level agreement from the service provider - *maximum number of rakes between mine and port in six hours* - can be interpreted as the planning parameter.

**Dumping** Dumping is *rail car dumping* activity where the material from the loaded rakes in the port area are dumped on to the stockpiles at the open yard in the port. The aggregate planning parameter is the *number of rakes that can be dumped per period*, and the decision to be taken is the number of rakes dumped per material per planning period.

**Loading** Loading is the activity of loading the bulk ore from the stockpiles in the open yard to the incoming ships. At the aggregate level, we characterize the loading activity by *loading rate* at each berth. The decision to be taken is the quantity of each material that needs to be loaded from the yard.

**Yard in the Mine Area** There are different types of bulk ores, which require separate stockpiles to avoid contamination and quality issues. We are looking at a horizon of two to four weeks of demand fulfillment. The supply from the mine is scheduled and is considered as input. For the time horizon under consideration, we can assume the inventory capacity at the mine yard to be infinite, with addition of material to the inventory happening as per the given supply schedule.

**Loaded Rakes in Port** The rail logistics activity pack the bulk ore into discrete units of rakes and transport them to the port area. The loaded rakes once arriving at the port area, wait to be dumped to the open yard. The inventory of loaded rakes is capacitated with an upper bound on the number of rakes.

**Yard in the Port Area** The bulk dumped from the loaded rakes are stacked to stockpiles in the yard. The yard in the port area is similar to that of at the mine area with separate stockpiles for different materials, but the capacity of stockpiles in the port area is limited.

The upstream supply planning problem is to determine each of the following *per planning period*: A) number of rakes per material supplied from the mine, B) number of rakes per material dumped at the port area, C) amount of material loaded on to the incoming ships. This is a *multi-stage multi-period* flow and inventory problem across the supply chain. The decision of loading the materials on to the incoming ships is closely tied to the berth allocation problem in terms of scheduling of ships and the material demanded. Thus we have a scheduling problem integrated with a supply planning problem.

There are two challenges in modeling the integrated planning and scheduling as single monolithic optimization problem. Firstly there are two different time units in the problem, where the planning period aggregates over large scheduling time units. Secondly, the problem has a generous mix of combinatorial and logical constraints of scheduling with the linear structure of planning. We propose in this paper a constraint based modeling of the integrated scheduling and planning problem, using the IBM ILOG CP Optimizer. We illustrate the technique with a simple example.

## Problem Definition

The problem is formally defined in this section. For the purpose of illustration, we have considered a simpler problem definition, ignoring many real world constraints, which are discussed in the later part of the paper.

### Berth Allocation and Ship Scheduling

Given a set of ships arriving at the port with demand for different materials, the problem is to allocate ships to compatible berths and schedule the loading of required materials. We consider just one material per ship with a penalty for failure to berth and load the material (loss of demand). Another penalty incurred is the delay in completion of loading of ships beyond the agreed upon demurrage time (tardiness cost) and a bonus (negative penalty) for early completion. The objective is to minimize the total penalty.

**Data** The given input data regarding the berths and ships are:

| | |
|---|---|
| $H$ | Scheduling horizon with index $h = 0, 1, \ldots, H$; |
| $N$ | Set of ships with index $i = 1, \ldots, \|N\|$; |
| $M$ | Set of berths with index $j = 1, \ldots, \|M\|$; |
| $K$ | Set of materials with index $k = 1, \ldots, \|K\|$; |
| $SB$ | = $\{(i, j)\}$, set of compatible ships and berths; |
| $l_j$ | Loading rate at berth $j$; |
| $a_i$ | Arrival time of ship $i$; |
| $e_i$ | End lay time of ship $i$; |
| $d_i$ | Demurrage time $\in (a_i, e_i)$ for ship $i$; |
| $o_i$ | Material $k \in K$ to be loaded in ship $i$; |
| $q_i$ | Quantity of material $O_i$ to be loaded in ship $i$; |
| $lt_{ij}$ | $= q_i/l_j$ for $(i, j) \in SB$, (Loading time for ship $i$ at berth $j$); |
| $p_i$ | Penalty incurred per unit time for ship $i$ sailing out later than the demurrage time $d_i$; |
| $b_i$ | Bonus accrued per unit time for ship $i$ sailing out earlier than the demurrage time $d_i$; |
| $r_i$ | Penalty for rejecting ship $i$; |

The berths are of varying capacity and hence only accommodate ships of acceptable capacity. We capture this in the tuple $SB$. As there is only one material per ship, the loading time is equivalent to the berthing time, but can change depending on the berth.

**Decisions** Scheduling decisions are:
(*SD1*) Accept/reject for servicing each ship;
(*SD2*) Allocation of accepted ships to berths;
(*SD3*) Scheduling of accepted ships on relevant berths;

**Constraints** Following are the mandatory constraints to be satisfied:
(*SC1*) Ship $i$ can only be berthed during its interval of availability $(a_i, e_i)$;
(*SC2*) Ship $i$ can only berthed at exactly *one* compatible berth $j$, such that $(i, j) \in SB$;
(*SC3*) Berthing/loading time for ship $i$ at berth $j$ is $lt_{ij}$;
(*SC4*) A berth cannot service more than one ship at a time;

**Objective** The objective is to *minimize* the sum of the following costs:
(*O1*) Total demurrage penalty, $DP$;
(*O2*) Negative of total demurrage bonus, $DB$;
(*O3*) Total penalty due to rejected ships, $RP$;

### Upstream Supply Planning

**Data** The input data for upstream supply planning consists of the maximum flows per period between the stages and inventory capacities.

| | |
|---|---|
| $P$ | Duration of a planning period; |
| $T$ | $= H/P$, Planning horizon with index $t = 0, 1, \dots, T$; |
| $s_t^k$ | Supply of material $k$ at period $t$ to the source; |
| $rq$ | Rake capacity; |
| $sp$ | Maximum number of rakes that can be supplied from source per period; |
| $dp$ | Maximum number rakes that can be dumped to the yard per period; |
| $lp$ | Maximum quanity of materials that can be loaded from the yard to the ships; |
| $tp$ | Number of planning periods required to transport the rakes from the source to the port; |
| $yc^k$ | Inventory capacity for material $k$ at the yard; |
| $rc$ | Inventory capacity for loaded rakes at the port; |

**Decisions** The decisions for the planning problem is to determine the flow of materials between each stage and the inventory at each for all the time periods in the planning horizon.
(*PD1*) Rakes with material $k$ supplied from source at period $t$: $x_t^k$;
(*PD2*) Rakes with material $k$ dumped to yard at period $t$: $y_t^k$;
(*PD3*) Quantity of material $k$ loaded to ships from yard at period $t$: $z_t^k$;
(*PD3)* Inventory of material $k$ in the source at period $t$: $ms_t^k$;
(*PD3*) Inventory of rakes loaded with material $k$ at period $t$: $rk_t^k$;
(*PD4*) Inventory of material $k$ in the yard at period $t$: $yd_t^k$;

**Constraints** The constraints in the planning are related to the *flow* and *inventory* at all the stages.
(*PC1*) The flow between the stages are capacitated:

$$\textstyle\sum_k x_t^k \le sp \quad \forall t \tag{1}$$

$$\textstyle\sum_k y_t^k \le dp \quad \forall t \tag{2}$$

$$\textstyle\sum_k z_t^k \le lp \quad \forall t \tag{3}$$

(*PC2*) Inventory balance constraints:

$$ms_t^k = ms_{t-1}^k + s_t^k - rq \times x_t^k \quad \forall k, t \tag{4}$$

$$rk_t^k = rk_{t-1}^k + x_{t-tp}^k - y_t^k \quad \forall k, t \tag{5}$$

$$yd_t^k = yd_{t-1}^k + rq \times y_t^k - z_t^k \quad \forall k, t \tag{6}$$

In the above inventory balance constraints, inventory parameters with index $t = 0$ denote the initial inventory. Parameters and variables with $t < 0$ are by default defined as 0.
(*PC3*) Inventory capacity constraints:

$$ms_t^k \ge 0 \quad \forall k, t \tag{7}$$

$$rq \times x_t^k \le ms_{t-1}^k \quad \forall k, t \tag{8}$$

$$\textstyle\sum_k rk_t^k \in [0, rc] \quad \forall t \tag{9}$$

$$y_t^k \le rk_{t-1}^k \quad \forall k, t \tag{10}$$

$$yd_t^k \in [0, yc^k] \quad \forall k, t \tag{11}$$

$$z_t^k \le yd_{t-1}^k \quad \forall k, t \tag{12}$$

There is no upper bound on inventory in the mine area as given by (7). The constraints (8), (10), and (12) model the constraint that the flow in period $t$ is bounded by the inventory in period $t-1$, to ensure inventory capacity constraints (by forbidding *consumption* and *production* in the same period).

### Linking Scheduling and Planning

As per the scheduling problem, the ship $i$ at berth $j$, requires $lt_{ij}$ time units to load the $q_i$ quantity of material $o_i \in K$ demanded by the ship. There is no restriction or information on the availability of material in the yard to be loaded on to the ships. In the planning problem, $z_t^k$ quantity of material $k$ is being loaded from the yard to the ships at the planning period $t$. To make the upstream planning and demand scheduling consistent, the $z_t^k$ should be equal to the total quantity of material $k$ loaded to the ships during the period $t$.

**Constraint** Planning - scheduling consistency constraint:
(*PSC*) Loading of material $k$ from the yard at period $t$ should be equal to the total quantity of material $k$ loaded on the ships during the scheduling horizon $[(t-1)P, tP)$.

## Constraint based Formulation with ILOG CP Optimizer

### ILOG CP Optimizer

IBM ILOG CP Optimizer is a software library which provides a constraint programming engine targeting both constraint satisfaction problems and optimization problems. The engine is designed to be used in a *model and run* development process, which contains powerful methods for finding feasible solutions and improving them. The strength of the optimizer removes the need for the user to write and maintain a search strategy. CP Optimizer provides a library of re-usable and maintainable classes that can be used just as they are, or extended to meet special needs. The classes define objects in the problem domain in a natural and intuitive way, enabling the user to clearly distinguish the problem representation from the problem resolution.

CP Optimizer provides integer decision variables with finite domain, along with a set of predefined constraint constructs, for modeling the problem. Arithmetic, relational, logical, and reification expressions with real number coefficients can be modeled as constraints. Other special constraints include the well known *allDifferent*, *count*, *packing*, *element*, and *allowedAssignments*. They are useful for modeling and solving a wide variety of combinatorial and constraint satisfaction problems. In addition, CP Optimizer provides a new scheduling language supported by a robust and efficient automatic search. This new-generation scheduling model is based on ILOGs experience in applying constraint-based scheduling to industrial applications (Laborie 2009). We use the ILOG CP Optimizer for modeling and solving the integrated scheduling and planning problem.

## CP Optimizer Model

We briefly outline the constraint based model for the problem using CP Optimizer. The scheduling model in CP Optimizer is primarily based on the *interval formalism* introduced in (Laborie and Rogerie 2008; Laborie et al. 2009). An *interval* variable $a$ is a decision variable whose domain dom($a$) is a subset of $\{\perp\} \cup \{[s,e)|s,e \in \mathcal{Z}, s \leq e\}$, where $s$ and $e$ are start and end of the activity, respectively. By default, the interval is *present*, which means that it is assigned to $\{[s,e)\}$ where $e - s$ denotes its length. If the variable is declared as optional, then it can also be *absent* ($=\perp$). When an interval variable is absent, then it is not considered by any constraint that involves the variable. The interval variable aptly models a scheduling activity.

**Berth Allocation** We model the berth allocation and ship scheduling using two interval variables.
**dvar interval** *ship*[$i$] **optional in** $a_i$ . . $e_i$;
**dvar interval** *shipLoad*[$< i, j >$] **optional size** $lt_{ij}$;
**alternative** (*ship*[$i$],
**all** ($< i, j >$ **in** $SB$) *shipLoad*[$< i, j >$]);

The above sample code in OPL shows the declaration of the two interval decision variables *ship* and *shipLoad* with a special constraint **alternative**. Declaration of *ship*[$i$] as **optional** allows to model the decision (*SD1*). The **in** option models constraint (*SC1*) by restricting the scheduling to be within the range of ship's arrival and end lay time. Note that as the variable is optional, the above constraint is valid only if it is present. In other words, if the ship is accepted for servicing, then it has to be scheduled within its arrival - departure time range.

In order to choose a berth to schedule the ship, the interval variable *shipLoad*[$< i, j >$] is declared for all $(i, j) \in SB$. The option **size** declares that the activity requires $lt_{ij}$ of time. The **alternative** constraint between *ship*[$i$] and the set of *shipLoad*[$< i, j >$] variables states that *ship*[$i$] is executed if and only if exactly *one* of the interval variables in the set *shipLoad* is executed, where both the activities are synchronized (modeling constraints (*SC2*) and (*SC3*)). In order to ensure (*SC4*), we use the cumulative function expression.

The cumulative function expression is a specialized constraint that allows to model cumulative resources and inventories. The usage of a cumulative resource and the level of a inventory are functions of time. A cumulative resource like a berth is used by ships, where its usage is increased at the start of ship berthing and and is decreased at the end of ship berthing.
**cumulFunction** *berthUsage*[$j$] =
**sum**($< i, j > \in SB$) **pulse**(*shipLoad*[$< i, j >$], 1);
*berthUsage*[$j$] $\leq 1$;

The **pulse**(*shipLoad*[$< i, j >$], 1) is a elementary cumulative function that contributes a *pulse* function with height 1, during the start and end of the interval variable *shipLoad*[$< i, j >$]. Note that as interval variable is optional, if it is absent, then there is no contribution. The *berthUsage*[$j$] is the sum of the elementary functions. If two ships overlap in their berthing schedule on berth $j$, then the value of the above expression is two during the period of overlap. By restricting the *berthUsage*[$j$] $\leq 1$, only one ship can be berthed at a time in a berth (constraint (*SC4*)).

**Upstream Supply Planning** The upstream planning problem is modeled in CP Optimizer directly as linear equations given by (1) - (12). The decision variables $x_t^k$, $y_t^k$, and $z_t^k$ are declared as integer decision variables.

**Linking Scheduling and Planning** The key aspect of the model is to be able to link the planning and scheduling seamlessly such that both parts fit together to form a single model. Also the scheduling problem is solved using the special interval constructs of CP Optimizer, whereas the multi-period, multi-stage planning problem is solved using integer decision variables and linear constraints.

The point of linking planning and scheduling is decision variables *shipLoad*[$< i, j >$] in scheduling and $z_t^k$ in planning, since both represent loading of material from stockyard to ship. We use the special scheduling constraint, **overlapLength** to achieve this linking.
$z_k^t$ **== sum**($< i, j > \in SB$ : $o_i == k$)
**overlapLength**(*shipLoad*[$< i, j >$], $(t-1) \times P$, $t \times P$)
$\times l_j$;

The **overlapLength** expression finds the length of interval *ship*[$< i, j >$] overlapping with the time interval [$(t-1) \times P$, $t \times P$], where $P$ is the duration of planning period. This overlap length when multiplied with the loading rate $l_j$ gives the material loaded to ship in the $t^{th}$ planning period. The expression **sum**($< i, j > \in SB$ : $o_i == k$) then sums up this quantity for all ships having demand for material $k$ to give $z_k^t$, and handles the constraint (*PSC*).

**Objectives** The objectives of the model are written as *decision expressions*. The three decision expressions are:
(*O1*) **dexpr float** $DP$ =
**sum**($i \in N$)**maxl**(0, **endOf**(*ship*[$i$],0) - $d_i$)) * $p_i$ ;
(*O2*) **dexpr float** $DB$ =
**sum**($i \in N$)**maxl**(0, $d_i$-**endOf**(s̄hip[$i$],0)) * $b_i$;
(*O3*) **dexpr float** $RP$ =
**sum**($i \in N$) (1 - **presenceOf**(*ship*[$i$])) * $r_i$;

The demurrage penalty or bonus is a function of end of the ship scheduling activity. The function **endOf**(*ship*[$i$],0) returns the end time of the interval *ship*$[i]$, if present, or returns 0 otherwise. The penalty for rejecting a ship is modeled using the **presenceOf** logical function, which returns 1 if the interval is present in the solution, 0 otherwise. The objective function is modeled as:

**minimize** $DP - DB + RP$;

## Illustrative Example

In this section, we show the applicability of the constraint based approach proposed above with a simple illustrative example. It is worth noting that the primary goal of constraint based solvers is to quickly come up with feasible solutions and improve them progressively. Proving optimality is a computationally intensive procedure and in practice, the algorithm is terminated with a time or memory limit. The alternative to benchmark optimality is to use a mathematical programming model. As we currently do not have such a model, we test the performance of the integrated planning scheduling model against the scheduling model. Firstly we solve only the berth allocation problem and compare it with the integrated upstream supply planning version.

### Berth Allocation Problem

The dataset considers a horizon of two weeks with 30 incoming ships to be serviced in four berths. The berths have capacity constraints and not all ships can be berthed at all berths. We set the granularity to *one* minute. The other relevant data are shown below.

| | |
|---|---|
| $H$ | $h = 0, 1, \ldots, 20160$ (2 weeks, 1 min granularity); |
| $\lvert N\rvert$ | 30 Ships; |
| $\lvert M\rvert$ | 4 Berths; |
| $\lvert K\rvert$ | 3 Materials; |
| $l_j$ | 10 kT/hr $\forall j \in M$; |

Given this data, we do the berth scheduling only without planning the supply, assuming that once a ship is berthed the order can be always be loaded (the material is already available). With this assumption we get the following best solution in CP Optimizer for a 30 second time limit set on the IBM ILOG CPLEX Optimization Studio 12.4, running on a Windows 7 laptop, with Intel i7 and 4 GB RAM.

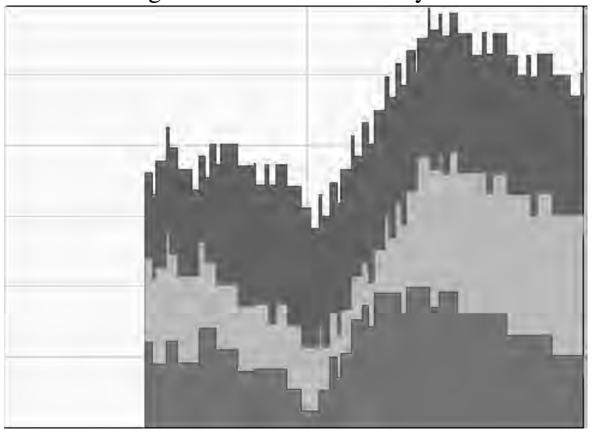
| | |
|---|---|
| Number of Ships berthed | 30 |
| Objective fun. Value | $-953000$ |
| $\sum_{i \in N} p_i$ | \$0 |
| $\sum_{i \in N} b_i$ | \$953000 |
| $\sum_{i \in N} r_i$ | \$0 |

### Berth Allocation with Upstream Supply Planning

We use the same dataset above to solve the integrated problem, with the following aggregate parameters for planning:

| | |
|---|---|
| $P$ | 6 Hours; |
| $T$ | $t = 0, 1, \ldots, 56$; |
| $rq$ | 16 kT; |
| $sp$ | 15 rakes; |
| $dp$ | 15 rakes; |
| $lp$ | 240 kT; |
| $tp$ | 2 periods; |
| $yc^k$ | 1000 kT ; |
| $rc$ | 40 rakes; |

Figure 2: Material Inventory at Yard



The above data ensures that the maximum flow across various stages has consistent throughput. The maximum material that can flow at the loading stage is $lp =$ 4berths$*$10kT/hr$*$6hrs$=$ 240kT/period. This flow is matched with supply from source and dumping as $sp,\ dp =$ 15rakes/period$*$16kT$=$ 240kT/period. The travel time for material to arrive at port is 2 periods, hence the initial inventory required to support supply of rakes is $rc = 40 \geq$ 2periods$*$15rakes/period and material at yard is $yc^k = 600 \geq$ 2periods$*$240kT. At the mine we have very high initial inventory for all materials so that supply is unconstrained.

The goal is to check the quality of the solution of the integrated problem against that of the berth allocation problem. As both the problems share the same ship and berth data, the *best* solution to the berth allocation problem can be used the lower bound to the integrated problem. The above dataset ensures that the upstream supply is indeed unconstrained. However, it does not make the upstream supply stages *redundant*, as the port area has only materials to cater to four to five days of demand. The model indeed has to find a solution that moves the materials from the mine to the port area. With the same set of computational resources and a 30 seconds time limit, the same solution was found.

| | |
|---|---|
| # Ships berthed | 30 |
| Objective fun. Value | $-953000$ |
| $\sum_{i \in N} p_i$ | \$0 |
| $\sum_{i \in N} b_i$ | \$953000 |
| $\sum_{i \in N} r_i$ | \$0 |

The material inventory at the yard over the planning periods is as shown in the fig. 2. It shows how the level of inventory for various material changes over planning periods, with incoming supply and loading. This simple example illustrates the applicability of the model. However, extensive set of computational experiments are required with varying demand and supply conditions to study the model performance and its sensitivity to different characteristics of the problem.

## Extensions

The problem illustrated in this paper had only the basic constructs of the real world problem. We list here some of the constraints that were not considered in the paper.

**Berth Allocation Problem** A) Channel capacity for sailing in and out; B) Sail out can happen only during high tides; C) Demand for multiple materials in a ship; and D) Directly loading of materials from loaded rakes to ship in the port area;

**Upstream Supply Planning** We considered aggregate parameters at each stage that constrain the flow between stages in a planning period using (1) - (3). The parameters are independent of the materials. In practice, there will be restrictions for each material. For example, consider the dumping activity. The number of rakes that can be dumped with the same material will be restricted as it would require access to same set of equipments that connect to same set of stock piles containing the specific material. In particular, following aggregate parameters per planning period are common: A) Maximum number of rakes per material that can be supplied from mine; B) Maximum number of rakes per material that can be dumped; C) Maximum number of simultaneous loading of same material.

## Conclusions and Future Work

We modeled a new variant of the berth allocation problem in this paper, which integrates detailed scheduling of ships with upstream supply planning. Motivated by a real world mining supply chain, the problem essentially models demand scheduling coupled with supply planning. The upstream supply planning is a generic multi-stage multi-period aggregate planning problem, which is commonly observed in many supply chains. Thus the proposed problem that integrates aggregate planning problem with a detailed scheduling problem across the vertical stages of the supply chain could possibly have several applications in different supply chains.

We proposed a constraint based modeling of the integrated problem using IBM ILOG CP Optimizer. The next step is a mathematical programming model that would help in benchmarking the performance of the constraint based model. As the problem has both the features of planning and scheduling, we believe that neither the constraint based model, nor the mathematical programming model would perform well. Our goal is to build a hybrid optimization model using Benders like decomposition that leverages both the constraint programming (for berth allocation) and mathematical programming (for upstream supply planning).

## References

Bierwirth, C., and Meisel, F. 2010. A survey of berth allocation and quay crane scheduling problems in container terminals. *European Journal of Operational Research* 202(3):615–627.

Buhrkal, K.; Zuglian, S.; Ropke, S.; Larsen, J.; and Lusby, R. 2011. Models for the discrete berth allocation problem: A computational comparison. *Transportation Research Part E: Logistics and Transportation Review* 47(4):461 – 473.

Caimi, G.; Fuchsberger, M.; Laumanns, M.; and Schupbach, K. 2011. A multi-level framework for generating train schedules in highly utilised networks. *Public Transport* 3(1):3 – 24.

Imai, A.; Nishimura, E.; and Papadimitriou, S. 2001. The dynamic berth allocation problem for a container port. *Transportation Research Part B: Methodological* 35(4):401 – 417.

Kelareva, E.; Brand, S.; Kilby, P.; Thiebaux, S.; and Wallace, M. 2012. CP and MIP methods for ship scheduling with time-varying draft. In McCluskey, L.; Williams, B.; Silva, J. R.; and Bonet, B., eds., *ICAPS*. AAAI.

Laborie, P., and Rogerie, J. 2008. Reasoning with conditional time-intervals. In *Proceedings of the 21st International Florida Artificial Intelligence Research Society Conference (FLAIRS 2008)*, 555–560. AAAI Press.

Laborie, P.; Rogerie, J.; Shaw, P.; and Vilím, P. 2009. Reasoning with conditional time-intervals. part ii: An algebraical model for resources. In *Proceedings of the 22nd International Florida Artificial Intelligence Research Society Conference (FLAIRS 2009)*. AAAI Press.

Laborie, P. 2009. IBM ILOG CP Optimizer for detailed scheduling illustrated on three problems. In van Hoeve, W. J., and Hooker, J. N., eds., *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 6th International Conference, CPAIOR 2009, Pittsburgh, PA, USA, May 27-31, 2009, Proceedings*, 148–162. Springer.

Pinedo, M. L. 2009. *Planning and Scheduling in Manufacturing and Services*. Springer.

Umang, N.; Bierlaire, M.; and Vacca, I. 2011. The Berth Allocation Problem in Bulk Ports. Swiss Transport Research Conference, STRC, Monte Verita / Ascona, Switzerland, May 13, 2011.

# Query Optimization Revisited: An AI Planning Perspective

**Nathan Robinson**
Dept. of Computer Science
University of Toronto
Toronto, Canada

**Sheila A. McIlraith**
Dept. of Computer Science
University of Toronto
Toronto, Canada

**David Toman**
School of Computer Science
University of Waterloo
Waterloo, Canada

**Abstract**

The generation of high quality query plans is at the heart of query processing in traditional database management systems as well as in heterogeneous distributed data sources on corporate intranets and in the cloud. A diversity of techniques are employed for query plan generation and optimization, many of them proprietary. In this paper we revisit the problem of generating a query plan using AI automated planning. Characterizing query planning as AI planning enables us to leverage state-of-the-art planning techniques, as well as supporting the longer-term endeavor of purposeful information gathering as part of a larger data-intensive, task-driven system. While our longterm view is broad, here our efforts focus on the specific problem of cost-based join-order optimization, a central component of production-quality query optimizers. We characterize the general query planning problem as a delete-free planning problem, and query plan optimization as a context-sensitive cost-optimal planning problem. We propose algorithms that generate high quality query plans, guaranteeing optimality under certain conditions. Our approach is general, supporting the use of a broad suite of domain-independent and domain-specific optimization criteria. Experimental results demonstrate the effectiveness of AI planning techniques for query plan generation and optimization.

## 1. Introduction

Informally, a *query plan* or a *query execution plan* is an ordered set of physical operations used to access information. *Query optimization* endeavors to find a query plan that maximizes the efficiency of execution, where efficiency may be measured in terms of minimizing space, latency, or other properties associated with the execution of the plan (e.g., (Ioannidis 1996; Chaudhuri 1998; Haas et al. 2009)). Traditionally the information being accessed by a query plan has resided in a relational database management systems, but as information management has evolved, query optimization has broadened to address plans that are executed over network accessible federated databases. Most recently, with the preponderance of structured and unstructured data in distributed information sources, there has been increasing interest in querying information sources that exist over the web and in the cloud, and extending beyond relational databases to linked data (e.g., (Ladwig and Tran 2010)).

In this paper we examine whether AI automated planning has anything of substance to contribute to the generation and optimization of plans for information gathering in general, and specifically for relational queries. Indeed there is a body of previous AI planning research related to query planning, including (e.g., (Kambhampati and Gnanaprakasam 1999; Nie and Kambhampati 2001; Kambhampati et al. 2004), (Knoblock 1996; Ambite and Knoblock 1997; 2000; Barish and Knoblock 2008), (Friedman and Weld 1997) ). Many of these works use planning or plan rewriting to construct query plans using simple physical operations, some relying on extensive processing outside the planner. Much of this (excellent) work is older work, little of it benefiting from advances in the state of the art in planning and plan optimization in the last decade. We were originally interested in revisiting this problem with the broadened perspective of advances in delete-free, cost-optimizing, and preference-based planning, and with a view to the integration of optimized information gathering into decision-making.

We are motivated by the task of the generating optimized information gathering plans in all of their guises, but for the purposes of this paper, our algorithms and empirical evaluation are tailored to the task of query optimization in relational database systems, and specifically to cost-based join-order optimization – the optimization of the ordering of join operations employed in conjunctive query evaluation.

The original 1979-published work on query optimization was with respect to System R, and used dynamic programming techniques (Selinger et al. 1979). Modern-day systems are proprietary and embedded within commercial systems, but reportedly some continue to use dynamic programming, while others use time-limited branch and bound search. Here we cast the general problem of information gathering as a delete-free planning problem, and the problem of optimizing the quality of information gathering as a cost-optimizing delete-free planning problem.

Unfortunately, the delete-free property of information gathering is not universally applicable. In particular, when we delve into the details of the particular relational database query optimization task of cost-based join-order optimization, we immediately observe two things. First, that the cost models that are employed in join-order optimization are context sensitive. The cost of an action is predicated on what has preceded it. Further, we observe that while information gathering is delete free, some of the physical operations employed to realize efficient query plans can have a component that deletes a property of our plan state. (The sorting of a table as part of some physical operations is one such example.)

We develop three somewhat diverse planning algorithms to address our query optimization problem: a delete-free algorithm, an optimal A* algorithm, and a greedy algorithm, together with a suite of domain-specific heuristics. We analyze their properties and assess their computational effectiveness. Of particular note is our ability to generate query plans that are guaranteed optimal on problems that are highly competitive to those reputed to be solved to optimality by commercial systems.

The work presented here is in its early stages, but is sufficiently advanced that there are interesting results and lessons to share. It introduces an interesting application to the AI Planning community, and a challenging problem to those interested in applications for cost-optimal (delete-free) planning and the more specific unaddressed problem of context-sensitive cost-optmal (delete-free) planning.

## 2. Preliminaries

We begin with a review of necessary relational database background and terminology. Conjunction queries (CQ) in SQL take the following form

$\texttt{select } x_1, \ldots, x_k \texttt{ from } R_1\, r_1, \ldots, R_n\, r_n \texttt{ where } C$

where $C$ is a conjunction of equalities of the form $r_i.a = x_l$ for $a$ an attribute (column) of the relation (table) $R_i$. This can be equivalently written as a *predicate calculus*-style comprehension of the form:

$$\{x_1 \ldots, x_k \mid \exists r_1, \ldots, r_k, x_{k+1}, \ldots, x_m. \\ R_1(r_1) \wedge \ldots \wedge R_n(r_n) \wedge \bigwedge R_i a_j(r_i, x_l)\}$$

where, conceptually, $R_i(r_i)$ atoms bind the variables $r_i$ to record id's of records in the instance of $R_i$ and $R_i a_j$ are binary relations that represent attributes of $R_i$ (attribute relations). Note that the *tuple variables* ($r_i$) are separate from the *value variables* ($x_j$). We allow some of the variables $x_i$ in the `select` list to be designated as *parameters*.

A typical query compiler and optimizer in modern relational database systems performs several steps to produce a *query plan*, that is instructions to the *query execution phase* that is ultimately responsible for retrieving the data and answering the user's query. The query optimizer's phases range from parsing, type-checking, view expansion, etc., to rule-based query rewriting and cost-based *query optimization*. In the following we focus on *cost-based* optimization for *conjunctive queries*, that roughly correspond to the most common queries in SQL, the so called `SELECT`-blocks. Indeed, this part of optimizing queries is commonly considered the corner stone of relational query optimization since the original System R (Selinger et al. 1979).

### 2.1 Operators for CQ Query Plans

The *query plans* for conjunctive queries are responsible for accessing the data relevant to the query answers that are stored in (possibly disk-based) data structures, called the *access paths*. The results of these primitive operations are then combined using *join* operators to form the ultimate query plan. Indeed, the crux of query optimization for conjunctive queries lies in the appropriate choice of appropriate access paths for the relations involved in the query and in the *ordering* of how the results of these operations are combined using joins – hence this part of query optimization is often dubbed *join-order selection*.

Additional relational operators, such as selections and projections are commonly *not* considered at this time—either they are fully subsumed by joins (such as in the case of constant selections) or can be added in post-processing (projections[1]).

[1]While we do not explicitly deal with duplicates in this presentation, all the techniques are fully compatible with SQL's duplicate semantics for conjunctive queries.

**Access Paths** The primitive relational operations are the *access paths* (APs), operators responsible for retrieving the *raw* data from relational storage (typically, disks). Every user relation (table) is typically associated with several access paths that support efficient search for tuples based on various conditions – e.g., find all Employee tuples in which the name attribute is "John". Note that the access paths used for search *expect* some of their attributes (the *inputs*) to be bound (i.e., associated with a constant value obtained earlier in the course of execution of the query plan). Formally, we can describe the access paths for a relation $R$ as triples of the form

$$\texttt{name}(r, x_1, \ldots, x_k) : \langle R(r) \wedge C, \{x_{i_1}, \ldots, x_{i_k}\}\rangle$$

where $C$ is a conjunction of equalities (similar to those in conjunctive queries) only using attributes of $R$ and variables $r$ and $x_1, \ldots, x_k$ out of which $x_{i_1}, \ldots, x_{i_k}$ denote the *input parameters* of this access method.

**Base File (scan and record fetch):** In the basic setting, for each relation $R$ we always have the following two access paths:

$$R\texttt{Scan}(r, x_1, \ldots, x_k) : \\ \langle R(r) \wedge Ra_1(r, x_1) \wedge \ldots \wedge Ra_k(r, x_k), \{\}\rangle \\ R\texttt{Fetch}(r, x_1, \ldots, x_k) : \\ \langle R(r) \wedge Ra_1(r, x_1) \wedge \ldots \wedge Ra_k(r, x_k), \{r\}\rangle$$

where $a_1, \ldots, a_k$ are all the attributes of $R$; these two paths are used to retrieve all tuples of a relation $R$ and to retrieve a *particular* tuple given its tuple id (note that the tuple id $r$ is the *input* to the access path and has to be bound before a record can be fetched).

**Indices:** In addition to the basic access paths we typically have additional access paths, called *indices*, that are used to speed up lookups for tuples based on certain search conditions (that are again captured by specifying inputs for the access path). Note also that the indices typically store only a few attributes of the indexed relation (the remaining ones can be retrieved using the `Fetch` access path). We capture this by declaring an access path

$$R\texttt{xxxIndex}(r, Y) : \langle R(r) \wedge C, X\rangle$$

for each *index* on $R$ (called generically `xxx` here) where $C$ is a conjunction of attribute relations (for attributes of $R$), is $X$ a set of names of variables that correspond to parameters of the index, and $Y$ is a set of variables that correspond to the attributes actually stored in the index (typically $X = Y$).

**Example 1** Given a relation `Emp(Id, Name, Boss)` we will have the following access paths:

$$\texttt{EmpScan}(r, x_1, x_2, x_3) : \langle \texttt{Emp}(r) \wedge \\ \texttt{EmpId}(r, x_1) \wedge \texttt{EmpName}(r, x_2) \wedge \texttt{EmpBoss}(r, x_3), \{\}\rangle \\ \texttt{EmpFetch}(r, x_1, x_2, x_3) : \langle \texttt{Emp}(r) \wedge \\ \texttt{EmpId}(r, x_1) \wedge \texttt{EmpName}(r, x_2) \wedge \texttt{EmpBoss}(r, x_3), \{r\}\rangle \\ \texttt{EmpIdIndex}(r, x_1) : \langle \texttt{Emp}(r) \wedge \\ \texttt{EmpId}(r, x_1)\{x_1\}\rangle \\ \texttt{EmpNameIndex}(r, x_1, x_2) : \langle \texttt{Emp}(r) \wedge \\ \texttt{EmpName}(r, x_1) \wedge \texttt{EmpId}(r, x_2), \{x_2\}\rangle$$

that allow retrieving all employee records, finding a record by record id, finding record ids using employee id, and finding record ids using employee name, respectively. Note that `EmpNameIndex` has an extra variable $x_2$ for `Id`; we will see later how this can be used for so-called *index only* query plans.

**Join Operators** To combine the results of access path invocations into query results, the *join* operators (that essentially implement *conjunctions*) are used. We consider the following two implementations of these operators:

**Nested Loops Join (NLJ):** The most basic join operator is based on the idea that for each tuple retrieved from its left argument it probes its right argument to find matching tuples. When the right argument is an access path with an input parameter present in the above tuple, the value is passed to the access path to facilitate search (in this case the join is often called the *IndexJoin*).

**Merge Sort Join (MSJ):** Another approach to implementing the join operator is to *sort* each of its arguments on the join attribute and then merge the results. While algorithmically preferable, the overhead of sorting often makes this method inferior to the plain NLJ. On the other hand, knowledge of *order properties* of the underlying access paths may allow the sorting step to be avoided.

Many other join implementations and algorithms have been investigated, such as the *Hash join* (based on creating a temporary hash-based index); for the purposes of this paper we focus on the above two joins without loss of generality.

To simplify the presentation we only consider left-deep query plans in this paper (this is similar to System R and is fully general for iterator-based plans that do not materialize intermediate results).

**Example 2** For a query:

$\texttt{select } x_1, x_2 \texttt{ from Emp } e$
$\texttt{where } e.\texttt{Id} = x_1 \texttt{ and } e.\texttt{Name} = x_2$

written as a comprehension as

$\{x_1, x_2 \mid \texttt{Emp}(r) \wedge \texttt{EmpId}(r, x_1) \wedge \texttt{EmpName}(r, x_2)\}$

we expect the following query plans, based on whether $x_1$ or $x_2$ (or neither) is a query parameter:

- None: $\texttt{EmpScan}(r, x_1, x_2, x_3)$
- $x_1$: $\texttt{EmpIdIndex}(r, x_1) \bowtie_{\text{NLJ}} \texttt{EmpFetch}(r, x_1, x_2, x_3)$
- $x_2$: $\texttt{EmpNameIndex}(r, x_1, x_2)$

Note that the last plan is an *index-only* plan. Also note that replacing *EmpFetch* access path that retrieves employee records based on record id by three paths one for each employee attribute would simulate how *column stores* execute queries. This relies on our representation of queries and access paths using tuple ids and attribute relations; indeed, this representation supports many advanced features that go far beyond textbook approaches and often generalizes hard-coded solutions present in production relational systems (such as unclustered indexing, index-intersection search, etc.).

## 2.2 Cost Model

The optimality of a query plan is judged with respect to a *cost model* based on summary (statistical) information about the relations and the access paths (that store the actual data); we follow a simple System-R style cost model to illustrate the approach (however, more advanced cost models can be easily used as well). We collect the following:

- for every relation $R$ (table): the number of tuples and, for each attribute, the number of distinct values;
- for each access path (index): the cost (no. of disk pages read) of retrieving all the tuples that match the access path's input parameters (reading the whole data set if none);

These estimates are then combined, using arithmetic formulas associated with particular join algorithms, to estimate the cost and cardinality of query plans (in disk page reads).

## 3. Mapping into PDDL Actions

We map join-order selection to an automated planning problem by combining the choice of the next access path with the appropriate implementation of the join operation in a single PDDL action (note that this is sufficient for our left-deep plans). We use the fluents $\texttt{needs-}R$, $\texttt{has-}R$, and $\texttt{bound}$ to capture the fact that the query needs to access a certain relation, that the current query plan has already accessed a certain relation, and that a variable has been bound to a value in the current plan, respectively.

### 3.1 Nested Loop Joins

First considering plans that use NLJ only (note that this also covers index-join based plans in the cases where NLJ is coupled with an index access path). For each AP

$\langle R\text{AP}, R(r) \wedge Ra_1(r, x_1) \wedge \ldots \wedge Ra_k(r, x_k), \{x_{i_1}, \ldots, x_{i_l}\}\rangle$

there is an action:

Action $\texttt{NLJ-}R\texttt{AP}$
pre: $\texttt{needs-}R(?r)$,
$\texttt{bound}(?x_{i_1})$, ..., $\texttt{bound}(?x_{i_l})$
post: $\texttt{has-}R(?r)$ $\texttt{has-}Ra_1(?r, ?x_1)$ ... $\texttt{has-}Ra_k(?r, ?x_k)$
$\texttt{bound}(?x_1)$ ... $\texttt{bound}(?x_k)$

Next, for the query:

$\{x_1 \ldots, x_k \mid \exists r_1, \ldots, r_k, x_{k+1}, \ldots, x_m.$
$R_1(r_1) \wedge \ldots \wedge R_n(r_n) \wedge \bigwedge Ra_i(r_i, x_l)\}$

we have an initial state $s_0$ such that:

$\texttt{needs-}R_1(r_1)$, ..., $\texttt{needs-}R_n(r_n) \in s_0$
$\texttt{needs-}Ra_i(r_i, x_l) \in s_0$ for all conjuncts in $\bigwedge Ra_i(r_i, x_l)$, and
$\texttt{bound}(x_j) \in s_0$ for all parameters in the query.

and a goal $\mathcal{G}$ such that:

$\texttt{has-}R_1(r_1)$, ..., $\texttt{has-}R_n(r_n) \in \mathcal{G}$ and
$\texttt{has-}Ra_i(r_i, x_l) \in \mathcal{G}$ for all conjuncts in $\bigwedge Ra_i(r_i, x_l)$.

**Example 3** For the query

$\{x_1, x_2 \mid \texttt{Emp}(r) \wedge \texttt{EmpId}(r, x_1) \wedge \texttt{EmpName}(r, x_2)\}$

with parameter $x_1$ we have a possible plan:

$\langle \texttt{NLJ-EmpIdIndex}(r, x_1), \texttt{NLJ-EmpFetch}(r, x_1, x_2, x_3)\rangle$

Note that the initial NLJ "joins" with a single tuple of parameters, in this example with the value for $x_1$. In the planner this corresponds to exploring the following sequence of states:

1. $\texttt{needs-Emp}(r)$, $\texttt{needs-EmpId}(r, x_1)$, $\texttt{needs-EmpName}(r, x_2)$, $\texttt{bound}(x_1)$
2. $\texttt{needs-Emp}(r)$, $\texttt{needs-EmpId}(r, x_1)$ $\texttt{needs-EmpName}(r, x_2)$, $\texttt{bound}(x_1)$, $\texttt{has-Emp}(r)$, $\texttt{has-EmpId}(r, x_1)$, $\texttt{bound}(r)$
3. $\texttt{needs-Emp}(r)$, $\texttt{needs-EmpId}(r, x_1)$, $\texttt{needs-EmpName}(r, x_2)$, $\texttt{bound}(x_1)$, $\texttt{has-Emp}(r)$, $\texttt{has-EmpId}(r, x_1)$, $\texttt{bound}(r)$ $\texttt{has-EmpName}(r, x_2)$, $\texttt{has-EmpBoss}(r, x_3)$, $\texttt{bound}(x_2)$, $\texttt{bound}(x_3)$

Another plan is $\langle \texttt{EmpScan}(r, x_1, x_2, x_3)\rangle$ . This produces the following sequence of states:

1. $\texttt{needs-Emp}(r)$, $\texttt{needs-EmpId}(r, x_1)$, $\texttt{needs-EmpName}(r, x_2)$, $\texttt{bound}(x_1)$
2. $\texttt{needs-Emp}(r)$, $\texttt{needs-EmpId}(r, x_1)$, $\texttt{needs-EmpName}(r, x_2)$, $\texttt{bound}(x_1)$, $\texttt{has-Emp}(r)$, $\texttt{has-EmpId}(r, x_1)$, $\texttt{has-EmpName}(r, x_2)$, $\texttt{has-EmpBoss}(r, x_3)$, $\texttt{bound}(x_2)$, $\texttt{bound}(x_3)$

This plan is however, less efficient given our cost model.

### 3.2 Adding Merge Sort Joins

While we could naively add MSJ to the above approach, we would miss opportunities arising from additional understanding of ordered properties of the access paths in order to avoid sorting steps in the plan.

We use the fluent $\texttt{asc}(x)$ to indicate that the values of the variable $x$ are sorted (ascending) in the output of the (current) query plan (again we use only single-variable orderings, but extending to other interesting orders is a mechanical exercise). Note, however, that unlike, e.g., the `bound` fluent, the sorted properties of variables may disappear after executing the next join, causing the encoding to lose its delete-free character.

To take advantage of order of access paths and results of partial query plans we use the following three actions that correspond to sorting the result of the current query plan, to merge-joining with an appropriately ordered access path, and to merge-joining with an access path that was sorted prior to the join, respectively:

Action $\texttt{Sort-on-?}x$: (sort results of the current plan on $x$)

pre: $\texttt{bound}(?x)$
post: $\texttt{asc}(?x)$ $\neg\texttt{asc}(?y)$ for all other variables $?y$

Action $\texttt{MJ-on-?}x\texttt{-}AP$: (add a *merge-join* on variable $x$ with the access path AP, assuming AP is also sorted on $x$)

pre: $\texttt{bound}(?x)$ $\text{asc}(?x)$
post: effects of AP as for NLJ
$\neg\texttt{asc}(?y)$ for all other variables $?y$

Action $\texttt{MSJ-on-?}x\texttt{-}AP$: (add a *sort-merge-join* on variable $x$ with the access path AP, assuming AP is not sorted on $x$)

pre: $\texttt{bound}(?x)$ $\texttt{asc}(?x)$
post: effects of AP as for NLJ
$\neg\texttt{asc}(?y)$ for all other variables $?y$

We also add $\texttt{asc}(x)$ to the initial state for each bound variable $x$. (This is sound since there is only a single "tuple" of parameters and constants.)

Finally, for a given query problem, we take an initial description of the problem instance, together with the schemas described here, and generate an query-specific PDDL planning instance. In addition to the information in the schemas, each individual action has an action cost that is a computation that relies on the variable bindings in the current state and as such is *context specific*. While PDDL supports context-specific action costs, few planners actually accommodate them, we therefore solve the previously described problems with the domain specific solvers presented in Section 5.

In more detail, the cost of our actions (that represent joining the next access path to the current query plan) depends on the number of tuples so far, captured as $size(s)$ for the current state $s$, the size and structure of the relation to be joined, and the particular join algorithm. These values are used to estimate the cost and size in successor states. For example, the cost of executing $\texttt{NLJ-}R\texttt{Scan}$ in state $s$ is $pages(R)ceil(size(s)/\texttt{buf-sz})$, that is we must join every page of tuples in the current store with every page of tuples in $R$ (where `buf-sz` is the number of tuples we buffer before scanning $R$). For brevity, we omit a full description of the cost functions we employ, noting instead that they are closely based on those used in System R.

## 4. Query Planning as AI Planning

In the previous section, we saw how to encode the join-order query optimization problem in terms of a PDDL initial state, goal, and a set of PDDL action schemas that are translated, together with their cost model, into a set of instance-specific ground actions. We refer to the problem of generating a query plan with the NLJ PDDL encoding as a J-O *query planning problem* and when augmented with MSJ APs as a J-O+ *query planning problem*. By inspection, we make the following observations:

**Obs 1** J-O *query planning is a delete-free planning problem.*

**Obs 2** J-O *query optimization is a context-sensitive cost-optimizing delete-free planning problem.*

**Obs 3** J-O+ *query planning is not a delete-free planning problem and as such,* J-O+ *query optimization is simply a context-sensitive cost-optimizing planning problem.*

We highlight these seemingly straightforward observations because they suggest the potential for exploiting advances in delete-free cost-optimizing delete-free planning techniques (e.g., (Gefen and Brafman 2012; Haslum, Slaney, and Thiébaux 2012; Pommerening and Helmert 2012)). Perhaps less encouraging is the observation that delete-free planning owes much of its computational advantages to the property that partial plans expand monotonically and a final ordering of groupings of actions can be extracted quite easily. Unfortunately, with a *context sensitive* cost model, order matters, and many of the gains afforded by cost-optimizing delete-free planning are lost at least with respect to current implementations. Nevertheless on the positive side, as we move beyond relational cost-based join-order optimization and consider other criteria for defining plan quality and other physical operators for realizing a query/information gathering plan, we see that the models of cost are often context *independent* and thus, again by inspection, we make the following observation:

**Obs 4** *A number of query and information-gathering optimization problems are context-independent cost-optimizing delete-free planning problems.*

This bodes well for the application of delete free planners to the generation of some classes of optimized information-gathering plans.

## 5. Generating Query Plans

Following from the observations in Section 4, we propose three algorithms together with a suite of domain-dependent heuristics for generating optimized query plans. The first algorithm, DF, exploits the delete-free nature of our problem, greedily generating cost-minimizing delete-free plans. The second is a classical A* algorithm, which we ran with three different admissible heuristics. The third, GR, is a greedy best-first search that does not consider partial plan cost in its evaluation function, but that uses an admissible heuristic, together with cost, in order to do sound pruning. The latter two algorithms can be guaranteed to produce optimal plans under certain conditions, which is notable relative to the state of the art in query planning. Note that while the heuristics have elements that are specific to the domain of join-order optimization, the general structure of the algorithms can be used for a diversity of information gathering/query plan optimization tasks simply by changing the cost function and the heuristic.

## 5.1 Fast Delete-Free Plan Exploration

Algorithm DF computes plans that do not include sorting actions. This prevents actions that merge on variables other than input variables. For certain problems, this precludes DF from finding optimal solutions, but the costs of the plans it finds are guaranteed to be upper bounds on the optimal cost allowing them to be used as initial bounds for the algorithms A* and GR.

The decision not to allow sorting actions means that in any state $s$, a subset of all actions can be efficiently determined that will move the planner towards the goal. We call such actions *useful* and denote the set of useful and applicable actions in state $s$ as $A_u(s)$. The algorithm proceeds by heuristically generating sequences of useful actions which achieve a goal state. Throughout its fixed runtime, it remembers the best such plan generated. Details of the algorithm follow.

**Algorithm 1** DF

```
$\pi_* \leftarrow \langle\rangle$; $c_* \leftarrow \inf$
while not time out do
    $s \leftarrow s_0$; $c \leftarrow 0$; $\pi \leftarrow \langle\rangle$
    while $\mathcal{G} \not\subseteq s$ do
        if random fraction of $1 \leq 0.9$ then
            $a \leftarrow a' \in A_u(s)$ with minimal resulting size,
                breaking ties with action cost
        else
            $a \leftarrow a' \in A_u(s)$ randomly selected with a
                likelihood inversely proportional to the
                resulting size and then action cost
        $c \leftarrow c + cost(a)$; $\pi \leftarrow \pi + a$
        if $c \geq c_*$ then break
        $s \leftarrow s \cup add(a)$
    if $c < c_*$ then $c_* \leftarrow c$; $\pi_* \leftarrow \pi$
return $\pi_*$ and $c_*$
```

Clearly DF does not guarantee to compute an optimal query plan, but it has the capacity to generate a multitude of plans very quickly. In practice, our Python implementation of DF can generate hundreds to thousands of candidate plans per second. In Section 6 we examine how effective this approach is empirically, at finding a high-quality plan. As noted below, this algorithm is also useful in providing a quality pruning bound for pruning of partial plans in our algorithms A* and GR.

## 5.2 A*

This algorithm A* is an eager A* search that uses a number of domain-dependent admissible heuristics. The code for A* is based on the eager search algorithm in Fast Downward (Helmert 2006). The primary difference from existing heuristic-search planning algorithms is that, as our action costs are context-dependent, we compute them lazily when expanding a state. While this increases the cost of expanding each state, it eliminates the need to pre-compute actions with all possible costs, which is prohibitively expensive.

We now examine the three heuristics that were used with this algorithm and show their admissibility and consistency and therefore the optimality of the solutions returned by A*. The first heuristic, $h_{blind}$ evaluates a state $s$ as follows:

- $h(s) = 0$, if $\mathcal{G} \subseteq s$; and
- $h(s) = 1$, otherwise.

**Proposition 1** *The heuristic $h_{blind}$ is admissible and consistent for the query-planning problems we consider.*

The consistency and admissibility of the heuristic $h_{blind}$ follow from the fact that 1 is a lowed-bound on the cost of any action and that the heuristic is clearly monotone.

The next heuristic, $h_{admiss}$ evaluates a state $s$ by counting the number of unsatisfied relations and assuming that $size(s)$ and all subsequent states is 1 to get a lower bound on the cost of achieving the goal.

In a given state $s$ let $\mathcal{R}(s)$ be the unsatisfied relations and $\mathcal{R}_I(s)$ be the (partially) unsatisfied relations for which, we have a bound tuple id – i.e. those relations for which a partial index action has been executed, which can be satisfied with a fetch action.

$h_{admiss}$ evaluates a state $s$ as follows:

- $h(s) = 0$, if $\mathcal{G} \subseteq s$, and
- $h(s) = |\mathcal{R}_I(s)| + \sum_{r \in \mathcal{R}(s) \setminus \mathcal{R}_I(s)} max(1, ceil(log_{200}(pages(R))))$

**Proposition 2** *The heuristic $h_{admiss}$ is admissible and consistent for for the query-planning problems we consider.*

To prove Proposition 2, we require to show that $h_{admiss}$ is guaranteed to not over-estimate the cost of reaching the goal from $s$ and is monotone. The former point can be seen by noting that the minimum cost of satisfying any relation $R$ is 1 when we can currently execute a fetch action on $R$, that is when $R \in \mathcal{R}_I(s)$ and otherwise the cost is at-least $max(1, ceil(log_{200} R.pages))$, from the System R-based cost model. $h_{admiss}$ is monotone because whenever we execute an action $a$ we either reduce the size of the sets $\mathcal{R}(s')$ and $\mathcal{R}(s')$ by 1 or leave them unchanged and if $|\mathcal{R}_I(s')| < |\mathcal{R}_I(s)|$ then $|\mathcal{R}(s')| < |\mathcal{R}(s)|$.

The final heuristic that we used, $h_{admissLA}$ builds upon the $h_{admiss}$ heuristic by performing one step of look ahead to take into account the size of the current state $s$. Let $A_u(s)$ be the set of applicable and useful actions in state $s$, including sort actions. $h_{admissLA}$ evaluates a state $s$ as follows:

- $h(s) = 0$, if $\mathcal{G} \subseteq s$, and
- $h(s) = min_{a \in A_u(s)} c(a) + h_{admiss}(s \cup add(a))$

**Proposition 3** *The heuristic $h_{admissLA}$ is admissible and consistent for the query-planning problems we consider.*

To see that Proposition 3 holds, see that $h_{admissLA}$ has a value for non-goal states that is the minimum over all successors of the cost to reach that successor $s'$ and the admissible and monotone estimate given by $h_{admiss}$ from $s'$.

**Theorem 1** *If* A* *terminates on a query planning problem of the type we consider, then it returns an optimal solution.*

We observe that our dynamic cost can be seen just as a proxy for a large number of ground actions with fixed-costs, hence the Theorem follows as the heuristics are consistent. Note that all of our heuristics are admissible. We explored the development of more informative inadmissible heuristics, which could be used with admissible heuristics for sound pruning. Unfortunately, as of this writing, we have found no superior inadmissible heuristic.

### 5.3 Greedy Best-First Search

This algorithm GR is an eager greedy best-first search that uses the same heuristics and code-base as A*. The sole difference between GR and A* is that GRorders the states on its open list solely on the basis of their heuristic values, that is by the function $f(s) = h(s)$. Expanded states $s$ are pruned when $g(s) + h(s)$ exceeds the current bound.

**Theorem 2** *An expanded state $s$ can be safely pruned when $g(s) + h(s)$ exceeds the current bound without sacrificing optimality.*

The proof idea is based on observing that the bounds are sound and heuristics admissible. The next Theorem follows as either we consider successors of a node or the node is pruned; lack of further nodes implies that an optimal solution was found (or none exists).

**Theorem 3** *If* GR *terminates on a query planning problem of the type we consider, then it returns an optimal solution.*

## 6. Evaluation

The question we'd like to address with our experimental evaluation is how AI planning techniques perform relative to the state of the art in relational database query planning. In particular, we'd like to know whether AI planning techniques have the potential to compute higher quality query plans faster than the state of the art. Unfortunately, the state of the art in relational database technology is embedded within proprietary systems, precluding systematic comparison. Similarly there are no suitable benchmarks for objective comparison of underlying algorithms. Instead what we do know and can leverage is that typical relational database systems employ time-limited planning algorithms and commonly trade plan optimality for reducing the time to find a reasonable query plan. For example, the classical System R query optimizer only considers plans that utilize Cartesian products as a last resort despite the fact that there are well-known examples where this heuristic leads to suboptimal query plans (this happens, e.g., when two relations with small cardinality are joined with a large relation indexed on attributes retrieved from both of these small relations). Other commercial systems utilize variants of time-constrained branch-and-bound algorithm in which the system attempts to allocate approximately the same time quota to exploring the "upper part" of the tree of possible plans as to the "lower part" (that is commonly much larger). Also, commercial query optimizers commonly take advantage of additional schema information, such as the key and foreign key constraints, that can be used to improve the cost estimation (this, however, makes the optimization problem easier as the additional information tends to make the cost difference between varying query plans more pronounced).

As the time to optimize queries is quite constrained in existing database systems the size of join-order problems that relational database optimizers currently solve (to optimality), in terms of the number of relations in a query, and the ratio of the number of variables to number of relations is rather small (again, in the absence of additional schema information). The actual algorithms and their performance in most commercial optimizers is a closely guarded business secret and only anecdotal evidence—that optimality is only guaranteed for about up to 10 way joins—is available.

With this information in hand, the purpose of our experiments was 1) to evaluate the relative effectiveness of the different approaches to query plan search and plan optimization that we examined, with the general objective of determining the relative merits and shortcomings of the algorithms and heuristics, and 2) to get some sense of whether AI automated planning techniques held some longer-term promise for cost-based join order optimization, in particular, relational database query optimization and more generally, and beyond that to the general problem of optimizing the quality of information gathering from disparate sources.

We evaluated 3 different algorithms: DF, our delete-free planning algorithm, A*, our A* search algorithm, and GR, our greedy search algorithm. The latter two algorithms were each evaluated with three different heuristics: $h_{blind}$, $h_{admiss}$, and $h_{admissLA}$. Our specific purpose was twofold. First, we aimed to determine how many problems each of A* and GR could solve optimally. Second, and more pragmatically, we aimed to determine the quality of the plans found by DF and A* as a function of time. Proprietary database systems usually allocate a short period of time for query planning (on the order of seconds) and for our algorithms to be practically useful they must find high-quality plans in this time frame.

In the absence of existing query planning benchmarks, we tested our planning systems on randomly generated database schemata and queries. Each generated schema consists of tables with between 2 and 10 attributes (not greater than half of the number of variables in the associated query). Each table has a random size of between 10k and 500k tuples and 200 tuples are assumed to fit into a page of memory. The first attribute of every table is assumed to be the primary key and has a number of distinct values equal to the table size. Every other attribute has a random number of distinct values up to $10\%$ of the table size.

Every query has a given number of relations $R = 5, 10, ..., 60$ and a given number of variables $V = 1.2, 1.5$, or 2 times $R$. Every query has 3 variables set as constants and 10 other variables selected (less if there is not enough variables). For each relation in the query there is a $10\%$ chance of reusing an existing table, otherwise a new table is used. Variables are randomly assigned to relations and we ensure that queries are connected.

Ten problem instances were generated for each $R$ and $V$. All experiments were run on a 2.6GHz Six-Core AMD Opteron(tm) Processor with 2GB of memory per experiment. We performed the following experiments.

### 6.1 Experiment 1: Optimal Plans

An upper bound $B$ on plan cost was produced by running DF for 5 seconds and then A* and GR were run with the initial bound $B$ with a time limit of 30 minutes. Running DF for longer than 5 seconds, to get tighter initial bounds, did not allow more problems to be solved optimally.

The results of this experiment can be seen in Figure 1. The number of problems that can be solved to optimality quickly drops off for both planners when the number of re-

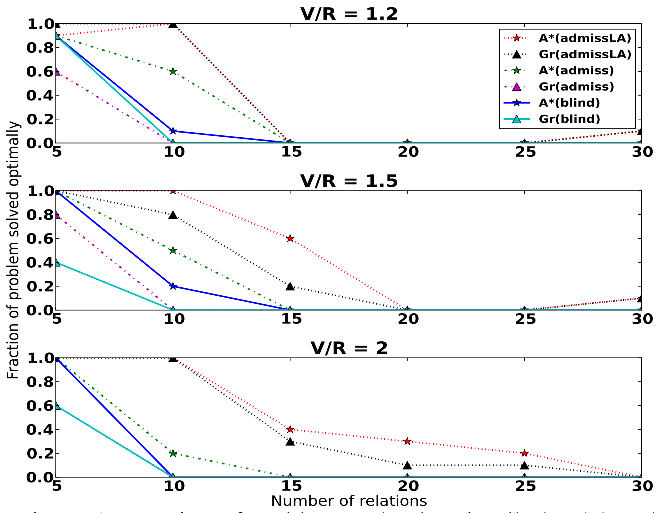


Figure 1: Fraction of problems solved optimally by A* and GR with different heuristics (2GB, 30 min time out).

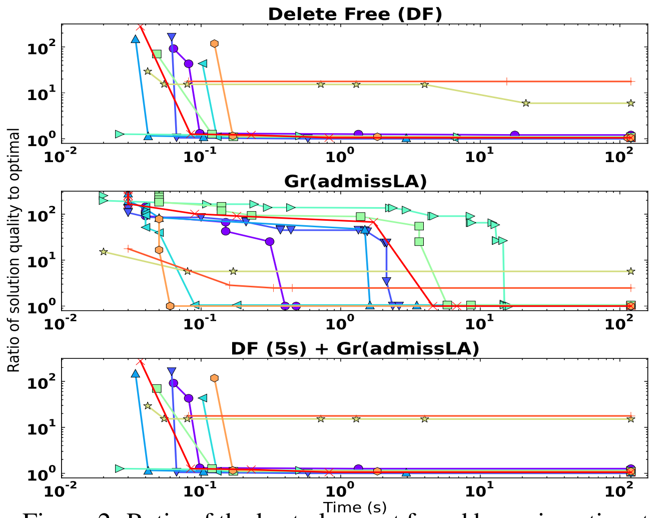


Figure 2: Ratio of the best plan cost found by a given time to the known optimal cost for problem instances with $R = 10$ and $V = 20$ for DF, GR, and DF run for 5 seconds, followed by GR with the resulting bound. GR used the $h_{admissLA}$ heuristic and all approaches had 2GB, 2 min time out.

lations is 15 or higher with A* and heuristic $h_{admissLA}$ performing the best. Problems with a higher $V/R$ ratio are somewhat easier than those with a low ratio. As expected, the different heuristics all give significantly different performance. $h_{blind}$ heuristic, due to its lack of guidance leads to the planners running out of memory on all but the smallest problems. $h_{admiss}$ leads to considerably better performance than $h_{blind}$, as it returns a value proportional to the number of relations left to satisfy and directs the planner towards the goal. However, as it ignores the size of the current and all intermediate states, it fails to distinguish between many plans. $h_{admissLA}$ provides improved performance by at least considering the size of the state that results from executing each action, even though it ignores the sizes of all subsequent states. An important remaining challenge is to develop an admissible heuristic that can take into account the sizes of states beyond one step ahead without doing full $k$ step lookahead, which was found to be too costly to be an effective heuristic.

In general, the planners solved those problems for which they could quickly find a sequence of cheap actions that bind a large portion of the variables while keeping the size small. Experiments show that those problems that can be solved optimally can be solved quickly, usually within a few seconds. On many of the remaining problems, the algorithms run out of memory exploring plateaus close to the goal.

## 6.2 Experiment 2: Fast High-Quality Plans

Given that our optimal algorithms do not scale acceptably for real world use, where an acceptable plan must be found within a few seconds, we also explored the use of several sub-optimal, any time algorithms.

In these experiments we ran DF and GR with no initial bounds and also GR with an initial bound generated by running DF for 5 seconds. Each of these algorithms was run for a total time of 2 minutes and plans were recorded as they were produced.

An important question to answer about these algorithms is how the quality of the plans that they find compares to the optimal. We only have optimal solutions for the smaller problems instances that could be optimally solved by DF and GR. Figure 2 shows a representative sample of these results using the $h_{admissLA}$ heuristic. It shows that for most problems, the sub-optimal algorithms find solutions that closely approach the optimal quality within a second. On the smaller problems, for which we could generate optimal solutions, DF more quickly approached high-quality solutions.

There was a small subset of problems for which these approaches are unable to find solutions within 10 times the cost of the optimal, even after 2 minutes. From the available evidence, this issue increases at larger problem sizes. Further analysis of the structure of these difficult problems is needed to determine what prevents the greedy approaches from finding high-quality solutions.

The approach of using DF to produce an initial bound for GR did not lead to a significantly better solution quality (after similar or longer run-times) and, due to the time required to find the initial bound, is impractical. This is not particularly surprising as GR very quickly finds upper bounds on plan cost anyway. We therefore omit this approach from further discussion here.

As well as comparing DF and GR to the optimal solutions, we performed extensive experiments to compare them to each other. Figure 3 shows the costs of the best plans found by DFand GR with all three heuristics after 0.5 and 5 seconds for all problem instances in our experiment set. Problems that could not be solved by an algorithm were assigned a cost of $10^5$ in that case.

From looking at the plots, it is clear that when the runtime is short, GR generally finds plans that are better than those found by DF, often by several orders of magnitude. As the experiment time increases, the quality of best plans found by DF improve relative to those found by GR. As can be expected, GR with the $h_{blind}$ heuristic fails almost all problems, usually running out of memory before any solutions are found. Over all run times there is a significant group of problems for which GR fails to compete with DF.

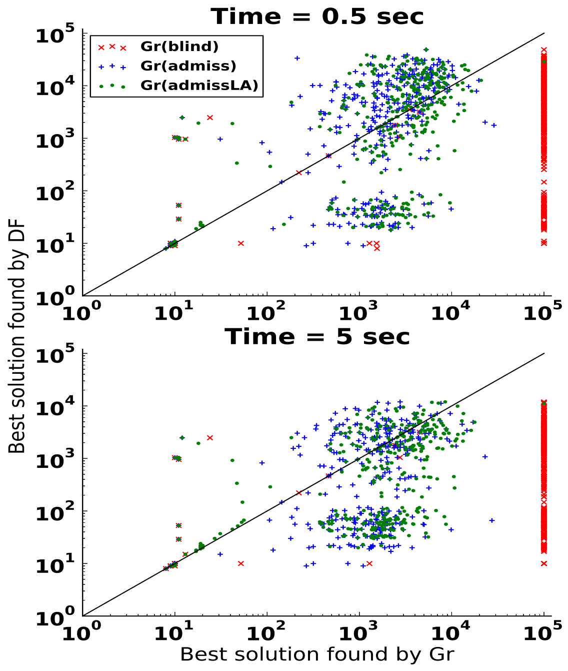


Figure 3: The costs of the best plans found by GR and DF after 0.5 and 5 seconds for all problem instances in our experiment set. Problems that could not be solved by an algorithm were assigned a cost of $10^5$ in that case.

This pattern of performance is not particularly surprising. GR initially performs better than DF because the $h_{admissLA}$ heuristic is considerably more informative than the action selection heuristic employed by DF when generating delete-free plans. This allows GR to very quickly find reasonably good plans. However, the greedy nature of GR means that expansions made early in the search can commit the algorithm to low quality parts of the search space. This behaviour can explain the cluster of problems on which GR consistently performs worse than DF.

## 7. Summary

This paper reports on preliminary findings relating to the applicability of AI automated planning techniques to the optimization of information gathering in general and to the generation of high quality cost-based join-order optimized query plans in particular. We observed that join-order query planning is a delete-free planning problem, and that query optimization is a context-sensitive cost-optimal delete-free planning problem. However, when we considered the broader problem that includes merge-joins, the delete free nature is lost. We developed delete-free, A*, and greedy planning algorithms which we combined with domain-dependent heuristics for generating optimized query plans. The latter two algorithms were guaranteed to produce optimal plans under certain conditions. Note that while the heuristics have elements that are specific to join-order optimization, the general structure of the algorithms can be used for a diversity of information gathering/query plan optimization tasks simply by changing the cost function and the heuristic. Experimental results are promising. Perhaps most notably, our planners could generate *optimal* query plans of a size that is highly competitive with those reputed to be solved by commercial systems. This work presents an interesting and challenging application domain for AI planning technology and a promising approach to solving a diversity of problems related to optimized information gathering.

**Acknowledgements:** The authors gratefully acknowledge funding from the Natural Sciences and Engineering Research Council of Canada (NSERC) and from the NSERC Business Intelligence Network program.

## References

Ambite, J. L., and Knoblock, C. A. 1997. Planning by rewriting: Efficiently generating high-quality plans. In *Proceedings of the 14th National Conference on Artificial Intelligence*, 706–713.

Ambite, J. L., and Knoblock, C. A. 2000. Flexible and scalable cost-based query planning in mediators: A transformational approach. *Artificial Intelligence Journal* 118(1-2):115–161.

Barish, G., and Knoblock, C. A. 2008. Speculative plan execution for information gathering. *Artificial Intelligence Journal* 172(4-5):413–453.

Chaudhuri, S. 1998. An overview of query optimization in relational systems. In *Proceedings of the 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 34–43.

Friedman, M., and Weld, D. S. 1997. Efficiently executing information-gathering plans. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, 785–791.

Gefen, A., and Brafman, R. I. 2012. Pruning methods for optimal delete-free planning. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling*, 56–64.

Haas, P. J.; Ilyas, I. F.; Lohman, G. M.; and Markl, V. 2009. Discovering and exploiting statistical properties for query optimization in relational databases: A survey. *Statistical Analysis and Data Mining* 1(4):223–250.

Haslum, P.; Slaney, J. K.; and Thiébaux, S. 2012. Minimal landmarks for optimal delete-free planning. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling*, 353–357.

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

Ioannidis, Y. E. 1996. Query optimization. *ACM Computing Surveys* 28(1):121–123.

Kambhampati, S., and Gnanaprakasam, S. 1999. Optimizing source-call ordering in information gathering plans. In *Proceedings of the 16th International Joint Conference on Artificial Workshop on Intelligent Information Integration*.

Kambhampati, S.; Lambrecht, E.; Nambiar, U.; Nie, Z.; and Gnanaprakasam, S. 2004. Optimizing recursive information gathering plans in EMERAC. *Journal of Intelligent Information Systems* 22(2):119–153.

Knoblock, C. A. 1996. Building a planner for information gathering: A report from the trenches. In *Proceedings of the 3rd International Conference on Artificial Intelligence Planning Systems*, 134–141.

Ladwig, G., and Tran, T. 2010. Linked data query processing strategies. In *Proceedings of The 9th International Semantic Web Conference*, 453–469.

Nie, Z., and Kambhampati, S. 2001. Joint optimization of cost and coverage of query plans in data integration. In *Tenth International Conference on Information and Knowledge Management*, 223–230.

Pommerening, F., and Helmert, M. 2012. Optimal planning for delete-free tasks with incremental lm-cut. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling*, 363–367.

Selinger, P. G.; Astrahan, M. M.; Chamberlin, D. D.; Lorie, R. A.; and Price, T. G. 1979. Access path selection in a relational database management system. In *Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data*, 23–34.

# A generic constraint-based local search library for the management of an electromagnetic surveillance space mission

**Cédric Pralet** and **Guillaume Infantes** and **Gérard Verfaillie**
ONERA - The French Aerospace Lab, F-31055, Toulouse, France
{Cedric.Pralet,Guillaume.Infantes,Gerard.Verfaillie}@onera.fr

## Abstract

This paper presents what has been done at the French Aerospace Lab (ONERA) to deal with a scenario of space mission defined by the French Space Agency (CNES). This space mission is dedicated to the surveillance from space of ground electromagnetic sources. It involves two satellites: one for source detection and another one for data acquisition and download. It presents two sources of uncertainty: the presence or not of electromagnetic sources and, in case of presence, the volume of data generated by acquisition. Due to these uncertainties and to limited communication windows with ground control stations, online planning and scheduling (P&S) is necessary on board the second satellite to make consistent and optimal decisions in terms of data acquisition and download. In this paper we show how a generic constraint-based local search library can be used to build the onboard planning and scheduling component. This library, called *InCELL*, has been developed at ONERA. It allows temporal constraints, resource constraints, arithmetic and logical constraints, and optimization criterion to be quickly and incrementally evaluated at each step of a local search algorithm. Already experimented to deal with simpler scenarios, this is the first time it is experimented on a complex scenario involving agile satellites. We show also how the generic simulation tool *Ptolemy* can be used to simulate the space system and evaluate its P&S component.

## Introduction

In the context of the CNES-ONERA *Agata* project about spacecraft autonomy (Charmeau and Bensana 2005), after working on a first mission scenario involving only one non agile Earth optical detection and observation satellite (Damiani, Verfaillie, and Charmeau 2004; Pralet and Verfaillie 2008), ONERA dealt with a more complex mission scenario defined by CNES and called *Agata-One*. The main objective was to assess whether or not the tools that were defined to deal with the first scenario can be easily adapted to deal with a more complex one.

The *Agata-One* scenario involves two agile Earth satellites placed on low altitude, circular orbits, on the same orbital plane. Agile satellites are able to perform very quick attitude movements along the three axes around their gravity center (roll, pitch, and yaw) generally thanks to gyroscopic actuators which are more efficient than usual reaction wheels. Thanks to regular roll attitude movements, the first satellite ($S_1$) scans a wide strip around its ground track. Thanks to its instruments, it is able to detect the presence of electromagnetic sources at the Earth surface and to localize them. In case of detection, it sends instantaneously information to the second satellite ($S_2$, which follows it at a small distance) via a permanent inter-satellite low-rate communication link. Satellite $S_2$ maintains a set of ground areas on which electromagnetic sources have been detected. Each time it overflies one of these areas, it can acquire data from it. To do that, it must perform a roll and pitch attitude movement to direct its acquisition instrument (a reception antenna) towards this area (the reception antenna is body-mounted on the satellite). When too many close areas must be handled, it must decide on those it will effectively handle and on the acquisition order. Once data from an area has been acquired, it is memorized in a mass memory and downloaded to ground reception and processing stations via a non permanent satellite-ground high-rate communication link. Downloading data to a ground station is only possible within one of the station visibility windows. Moreover, it is only possible when the satellite attitude is compatible with data download (as the reception antenna, the emission antenna is body-mounted on the satellite and, during the whole download period, the station must remain inside the satellite emission antenna cone). As for data acquisition, when too much data must be downloaded, satellite $S_2$ must decide on those it will download and on the download order. Fig. 1 summarizes the exchanges between satellites and ground reception stations.

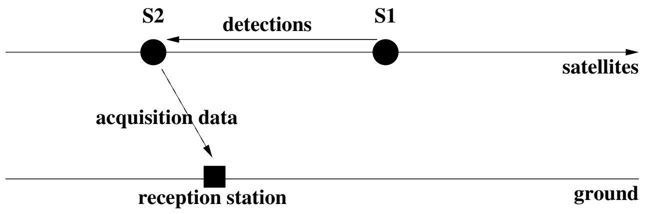


Figure 1: Exchanges between satellites and ground reception stations.

It must be stressed that the attitude of satellite $S_2$ allow-

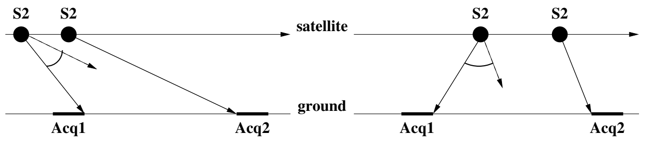


Figure 2: How the attitude movement to be performed by satellite $S_2$ to transit from a data acquisition to another one depends on the time at which the transition is triggered. In the second case (right), the angular movement to be performed is greater than in the first case (left).

ing it to direct its reception antenna towards a given area depends on the position of the satellite on its orbit and thus on time. In such conditions, the attitude movement necessary to transit from a data acquisition from a given area to a data acquisition from another area, and thus the time taken by this transition, depends not only on both areas, but also on the time at which the transition is triggered (time-dependent transition duration). See Fig. 2 for an illustration.

This mission scenario involves two main sources of uncertainty: the presence or not of electromagnetic sources and, in case of presence, the volume of data generated by acquisition (this volume is highly variable and can typically range from 1 to 1000). Due to these uncertainties and to the non permanent visibility of satellites by ground control stations, online decision-making on data acquisition and download is necessary on board satellite $S_2$. To make such decisions, it would be possible to use manually defined decision rules. However, decisions would be better informed if they could use the result of P&S: planning and scheduling regularly performed over a given horizon ahead, using the most up to date information about detections and data volumes; decisions made according to the first steps of the plans produced.

Due to the limited computing time available for P&S and due to the limited computing resources available on board (CPU and RAM), heuristic search (greedy and/or local search) seems to be the right option to build an anytime combinatorial search procedure, able to produce quickly good quality plans and to improve on them as long as time is available before making decisions. It is widely used in space missions that require online onboard P&S (Chien et al. 2000; 2005b; 2005a). We already used it in the context of Earth observation and surveillance missions (Lemaître et al. 2002; Beaumet, Verfaillie, and Charmeau 2011; Pralet and Verfaillie 2008; Pralet et al. 2011; Verfaillie et al. 2011). However, each time, we built specific heuristic search procedures, dedicated to the specific mission at hand and not directly reusable to handle other missions. To deal with the *Agata-One* scenario, we decided to change our approach and to use generic tools developed at ONERA in the context of the *Agata* project and, more specifically, the *Invariant-based Constraint EvaLuation Library* (*InCELL* (Pralet and Verfaillie 2013)).

*InCELL* draws its inspiration from the ideas of *Constraint-based local search* (CLS (Hentenryck and Michel 2005)). In CLS, the user defines a model of its problem in terms of decision variables, constraints, and optimization criterion. She/he defines also its local search procedure over the set of complete variable assignments (where every variable is assigned). Because the speed of each local move is one of the keys to local search success, the software uses so-called *invariants* which allow expressions and constraints to be quickly and incrementally evaluated after each move. In *InCELL*, multiple-input multiple-output invariants allow expressions, arithmetic and logical constraints, temporal and resource constraints to be expressed and efficiently handled. *InCELL* calls for *Simple Temporal Network* (STN (Dechter, Meiry, and Pearl 1991)) techniques which allow temporally flexible plans to be produced, and for *Time-dependent* STN (TSTN (Pralet and Verfailllie 2012)) techniques which allow time-dependent transition durations to be taken into account.

To deal with the *Agata-One* scenario, an *InCELL* model of the associated P&S problem (decisions about data acquisition and download by satellite $S_2$) was built, a simple non chronological greedy search procedure was designed, and the events that trigger a new call to P&S over a given horizon ahead were defined.

To simulate the space system and to evaluate its P&S component, an event-based model of the system, based on the notions of state, event preconditions and effects, and event activations, was built and implemented using the generic simulation tool *Ptolemy* (Eker et al. 2003). Whereas P&S allows only the utility of decisions over the planning horizon to be evaluated, this simulation allows the global utility of successive decisions over the simulation horizon to be evaluated.

Sect. 1 describes problem data and Sect. 2 presents the structure of possible decisions. In Sect. 3, a constraint-based model of the P&S problem is introduced. The main ingredients of the *InCELL* library, as well as its main reasoning mechanisms, are presented in Sect. 4. Sect. 5 describes the search procedure and Sect. 6 defines when P&S is called. Sect. 7 shows how the space system and its P&S component can be simulated and evaluated, using the *Ptolemy* tool.

## 1 Problem data

Permanent (static) problem data is the following:

- a finite set of ground areas that must be kept under surveillance;
- a finite sequence of priority levels;
- for each ground area, its priority level, its weight (to give more or less weight to areas of the same priority level), an acquisition duration, an expected, a minimum, and a maximum volume of data resulting from acquisition;
- a finite set of ground reception stations;
- a data download rate from satellite $S_2$ to any ground reception station.

Moreover, it is assumed that a function associates with each ground area $a$ and each time $t$ the attitude of satellite $S_2$ necessary to acquire data from $a$ at time $t$, when acquisition is possible, and that another function associates with each pair of attitudes of satellite $S_2$ the minimum time necessary to reach the second one, starting from the first one.

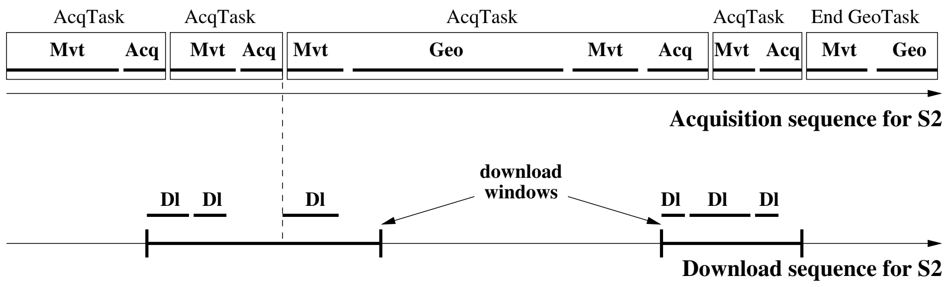


Figure 3: The two concurrent sequences of action on board satellite $S_2$. AcqTask = acquisition task; Geo Task = geocentric task; Mvt = attitude movement; Acq = data acquisition; Geo = geocentric pointing; Dl = data download.

Each time P&S is called, its complementary (dynamic) data is the following:

- a planning horizon ahead;
- an attitude of satellite $S_2$ at the beginning of the planning horizon;
- a set of ground areas from which electromagnetic sources have been detected by satellite $S_1$, but no acquisition by satellite $S_2$ has been performed yet;
- for each of these ground areas, its detection time and a finite sequence of acquisition windows by satellite $S_2$ over the planning horizon;
- a finite set of acquisitions that have been already performed, but whose data has not been downloaded yet (still present in memory);
- for each of these acquisitions, its detection and acquisition times and its actual volume in memory;
- a finite sequence of download windows by satellite $S_2$ over the planning horizon.

Acquisition windows are reduced in case of intersection with a download window, when acquisition is incompatible with download, in order to give priority to data download.

## 2 Possible decisions

On board satellite $S_2$, it is necessary to decide on two concurrent sequences of action:

- the sequence of data acquisitions;
- the sequence of data downloads.

The first sequence is made of acquisition tasks, each one following immediately the previous one. An acquisition task is, according to an HTN-like decomposition of tasks into sub-tasks (*Hierarchical Task Networks* (Nau et al. 2003)), itself made of:

- either an attitude movement immediately followed by a data acquisition;
- or an attitude movement immediately followed by a geocentric pointing, immediately followed by another attitude movement, immediately followed by a data acquisition (satellite geocentric pointing maintained towards Earth center is a waiting action, favourable to communication with Earth, data downloads, and energy recharging).

This sequence, possibly completed by an attitude movement followed by a geocentric pointing at the end of the planning horizon, entirely defines the attitude trajectory of satellite $S_2$.

The second sequence is made of data downloads, each one being performed within a download window. In this sequence, a download may not immediately follow the previous one. This is the case when it is necessary to wait for the end of an acquisition before downloading resulting data within a download window.

Fig. 3 illustrates the two concurrent sequences of action. Both sequences are not independent from each other because data download requires preceding acquisition.

## 3 A constraint-based model

P&S problem is a kind of over-constrained scheduling problem (over-constrained because it may be impossible to schedule all the candidate tasks (Kramer and Smith 2003)) which can be modeled using only constraints over intervals. An interval is defined by its presence, its starting date, its ending date, and its duration. Its presence is a boolean, equal to 1 if and only if the interval is effectively present in the schedule.

The model associates:

- with each ground area from which electromagnetic sources have been detected by satellite $S_1$, but no acquisition by satellite $S_2$ has been performed yet, an acquisition interval, a geocentric pointing interval, and a download interval;
- with each acquisition that has been already performed by satellite $S_2$, but whose data has not been downloaded yet, a download interval.

These intervals may be present or absent. Constraints to be satisfied are the following:

- each acquisition interval must be, when present, included in one of the acquisition windows of the associated ground area; its duration is the acquisition duration of the associated ground area, defined in the problem data;
- each download interval must be, when present, included in one of the download windows; if the acquisition has been already performed at the P&S time, download duration is equal to the actual volume in memory divided by

the download rate; if it has not been performed yet, it is equal to the maximum volume resulting from acquisition divided by the download rate (pessimistic assumption allowing the produced schedule to be surely executed);

- for each ground area from which electromagnetic sources have been detected, absence of the acquisition interval implies absence of the geocentric pointing and download intervals; presence of the geocentric pointing interval implies that it must precede the acquisition interval and follow the previous acquisition interval; presence of the download interval implies that it must follow the acquisition interval;
- there must be no overlapping between present acquisition and geocentric pointing intervals and enough time between successive intervals to allow attitude movements; moreover movements to or from geocentric pointings must be performed in minimum time in order to give geocentric pointing as much time as possible;
- there must be no overlapping between present download intervals.

The criterion to be optimized is a vector of global utilities, one per priority level. The global utility associated with a priority level $p$ is equal to the sum of the local utilities associated with each of the ground areas of priority $p$. The local utility associated with a ground area is equal to its weight multiplied by two functions which both take a value between $0$ and $1$: a decreasing function of the time between detection and acquisition and another decreasing function of the time between acquisition and download. These functions tend to encourage quick acquisition and quick delivery of information on the ground. Two vectors of global utilities, resulting from two schedules, are lexicographically compared from the highest priority level to the lowest one.

## 4 The InCELL library

*InCELL* (*Invariant-based Constraint EvaLuation Library*) is a software library, dedicated to the quick incremental evaluation of expressions and constraints.

*InCELL* draws its inspiration from the ideas of *Constraint-based local search* (CLS (Hentenryck and Michel 2005)). In CLS, the user defines a model of its problem in terms of decision variables, constraints, and optimization criterion. She/he defines also its local search procedure over the set of complete variable assignments (every variable assigned). Because the speed of each local move is one of the keys to local search success, the software uses so-called *invariants* which allow expressions and constraints to be quickly and incrementally evaluated after each move. An invariant is a one-way constraint of the form $x \leftarrow exp$, where $x$ is a variable and $exp$ a function of other variables, such as for example $x \leftarrow \sum_{i=1}^{N} y_i$. On this example, when the value of $y_j$ for some $j$ is modified, it is not necessary to recompute $\sum_{i=1}^{N} y_i$ from scratch. It suffices to add to the previous value of $x$ the new value of $y_j$, minus its old value. The only condition is the absence of cycles in the definition of invariants (no variable directly or indirectly function of itself).

*InCELL* extends the definition of invariants by allowing multiple-input multiple-output invariants. Invariants allow expressions, but also constraints, to be represented. Constraints, such as for example $\sum_{i=1}^{N} y_i \leq K$, are specific invariants whose evaluation stops when they are violated. In *InCELL*, a constraint optimization problem (variables, constraints, and criterion) takes the form of a DAG (*Directed Acyclic Graph*) of invariants. Each time the value of some atomic variables (variables that are not functions of other variables and are roots of the DAG) is modified, the DAG of invariants is lazily reevaluated according to a DAG topological order: any invariant is reevaluated only when necessary and at most once.

On top of these basic concepts and mechanisms, *InCELL* offers some constructs dedicated to scheduling: time point variables, interval variables (defined by two time point variables and a distance constraint between them), unary and binary distance constraints (of the form $x \leq K$ or $x-y \leq K$). All temporal constraints are managed using a special STN invariant (*Simple Temporal network* (Dechter, Meiry, and Pearl 1991)) which has as inputs a set of unary and binary distance constraints and as outputs the earliest dates of all the time point variables involved in the constraints. Classical STN techniques are used to handle the STN: constraint propagation, maintenance of propagation chains, decomposition of the distance graph into strongly connected components. Moreover, STN concepts and techniques are extended in *InCELL* to deal with so-called *time-dependent scheduling* (Gawiejnowicz 2008), that is with time-dependent distance constraints (Pralet and Verfailllie 2012) where the minimum distance is not a constant, but a function of the involved time points (of the form $x - y \leq F(x, y)$ with some assumptions about Function $F$). All the constraints over intervals, defined in the previous section, can be managed by *InCELL*, including the minimum transition times between successive acquisition and geocentric pointing intervals, thanks to time-dependent distance constraints.

*InCELL* allows also resource constraints to be defined and profiles of resources (with piecewise constant or linear evolutions) to be quickly and incrementally maintained, taking into account the earliest dates produced by the STN. This would allow memory (piecewise constant evolution) and energy (piecewise linear evolution) constraints to be managed. However, these constraints are ignored in our problem: energy because it is not limiting and memory because of the uncertainty about the volume of data generated by acquisition. When planning acquisitions, we prefer not to limit acquisitions because of possible large volumes of data. However, when executing the acquisition plan, before triggering an acquisition, in case of possible memory overflow, we remove from memory lower priority data and, when it is not sufficient, we cancel the acquisition.

One of the key features of *InCELL* is its ability to work on dynamic models (a new model each time P&S is called) using a unique static model which is recycled to build dynamic models. This allows any dynamic memory allocation to be avoided on board: a key requirement when building embedded reactive control software.

See (Pralet and Verfaillie 2013) for more details about the

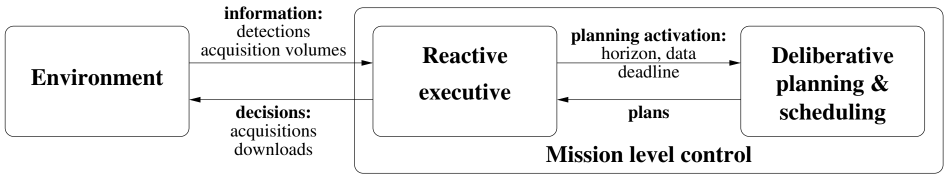


Figure 4: Exchanges between the environment, the reactive executive, and the deliberative P&S component.

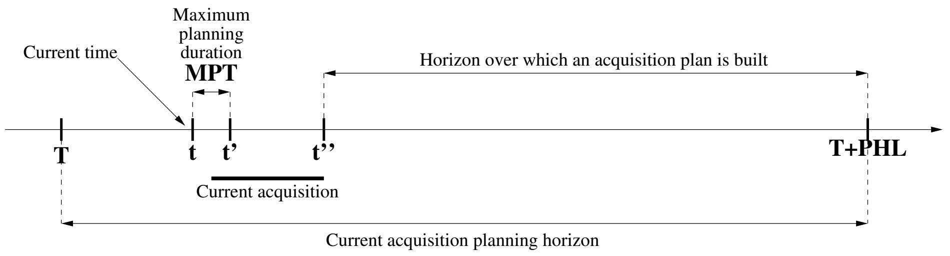

Maximum planning duration
MPT
Current time
Horizon over which an acquisition plan is built
T
t
t'
t''
T+PHL
Current acquisition
Current acquisition planning horizon

Figure 5: Horizon over which an acquisition plan is built at a given time $t$ on board satellite $S_2$.

*InCELL* library.

## 5 A non chronological greedy search

To build online acquisition and download plans on board satellite $S_2$, we defined a very simple greedy search procedure, although more sophisticated local search procedures could be considered (Aarts and Lenstra 1997).

At each step of this procedure, an acquisition (resp. download) of highest priority level and of highest utility at this priority level is selected and added to the acquisition (resp. download) plan, when addition is possible. It is added in the best acquisition (resp. download) window and at the best position in the acquisition (resp. download) sequence in terms of utility. Once the acquisition sequence is defined, geocentric pointings are added between acquisitions, when possible.

## 6 Calls to planning and scheduling

In case of online P&S, it is not only necessary to define the P&S model and the reasoning and search mechanisms. It is necessary to define when the executive calls to P&S, in order to get a plan over some planning horizon ahead and to follow it until a new call to P&S. See Fig. 4 for a global view of the exchanges between the environment, the reactive executive, and the deliberative P&S component.

In our problem, as far as acquisitions are concerned, we define the length $PHL$ of the planning horizon (horizon over which P&S is called; typically some hours) and the maximum planning time $MPT$ (maximum time taken by P&S; typically some seconds). The planning horizon of length $PHL$ is regularly shifted (typically every half an hour). P&S is called again when a new acquisition opportunity appears over the planning horizon. This happens either when electromagnetic sources are detected by satellite $S_1$ on some ground area, or when the planning horizon is shifted and a new acquisition window for some ground area appears over the new planning horizon. In such a case, we consider the current time $t$, the time $t' = t + MPT$ at which a plan will be surely available, the time $t''$ from which decisions can be made, taking into account acquisition or attitude movement possibly in progress at $t'$ (acquisitions and attitude movements are not interruptible, but geocentric pointings are), and we call to P&S over the planning horizon from $t''$. See Fig. 5 for an illustration.

As far as downloads are concerned, P&S is called $MPT$ before each download window (or group of windows that overlap or are very close to each other) over the whole window (or group of windows).

## 7 Simulation

We used the simulation tool *Ptolemy* to simulate the space system. *Ptolemy* (Eker et al. 2003) (see http://ptolemy.eecs.berkeley.edu/) is a generic tool dedicated to the simulation of dynamic systems, with an emphasis on hybrid simulation. Among many other possibilities, it is possible within *Ptolemy* to simulate a system whose dynamics involves both discrete events and continuous evolutions of resources. To express in *Ptolemy* the dynamics of the space system, we particularly relied on the *Ptera* framework (Feng, Lee, and Schruben 2010) which is based on the notions of state, events, event preconditions and effects, and conditional activations by events of other events (possibly with some delay and some probability). See Fig. 6 for an illustration of the several temporal horizons that are handled in the simulation (simulation, commitment, planning, and decision horizons).

This simulation was run on scenarios built by CNES. Fig. 7 shows a screenshot of the simulation tool at the end of a five day simulation horizon, where one can see:

- at the top left, the current acquisition requests over the whole world (small circles) and the visibility circle of the unique ground reception station (in blue);
- at the top right, an artist view of satellites $S_2$ (in front) and $S_1$ (behind) with the pointing direction of the former;

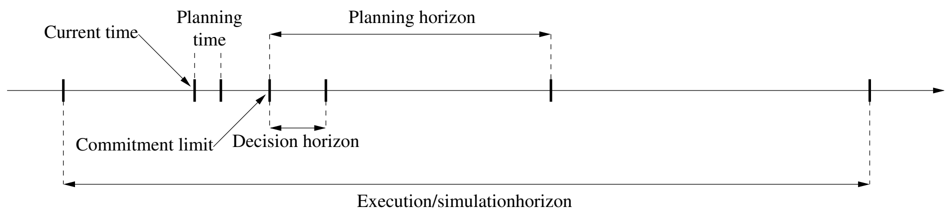
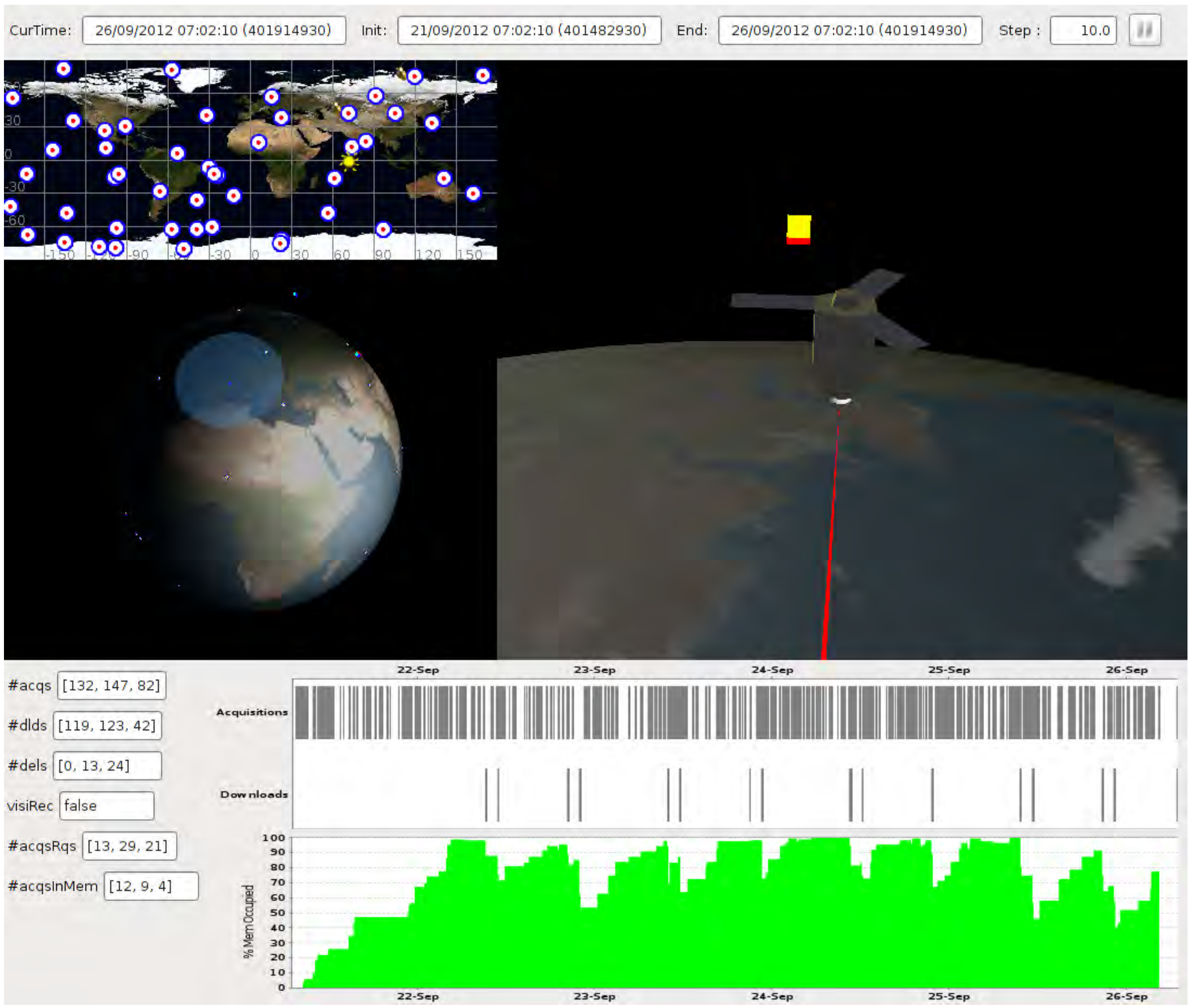


Figure 6: Illustration of the several temporal horizons that are handled in the simulation.



CurTime: 26/09/2012 07:02:10 (401914930) Init: 21/09/2012 07:02:10 (401482930) End: 26/09/2012 07:02:10 (401914930) Step : 10.0

#acqs [132, 147, 82]
#dlds [119, 123, 42]
#dels [0, 13, 24]
visiRec false
#acqsRqs [13, 29, 21]
#acqsInMem [12, 9, 4]

Figure 7: Screenshot of the *Agata-One* simulator at the end of a five day simulation horizon.

| **Prio** | 1 | 2 | 3 |
|---|---|---|---|
| **NAcq** | 132 | 147 | 82 |
| **NDl** | 119 | 123 | 42 |
| **NRm** | 0 | 13 | 24 |

Table 1: Global results over the five day simulation horizon: **Prio** = priority level, **NAcq** = number of performed acquisitions; **NDl** = number of downloaded acquisitions, **NRm** = number of performed acquisitions that have been removed from memory to free space.

- at the bottom right, the sequence of acquisitions, the sequence of downloads, and the evolution of memory on board.

Over this simulation horizon, acquisition planning is called 680 times, each time over a four hour horizon ahead. Download planning is called 16 times, each time over the next download window. Each time it is called, acquisition (resp. download) planning must manage some tens of acquisitions to be performed (resp. downloaded). Acquisition (resp. download) planning takes on average 432 ms (resp. 2258 ms) on an i5-520 Intel processor with 1.2 GHz and 4 GBRAM.

The global results per priority level are shown on Tab. 1 The mean utilization percentage of the downloads windows is of 85.39%.

A demonstration of the space system simulation is presented in the ICAPS 2013 Application Showcase.

## Conclusion

The first result of this study is the demonstration that the generic *InCell* library allows the planning problem associated with a new complex space mission to be easily modeled and efficiently solved. Beyond the necessary improvements of the library in terms of modeling power and algorithm efficiency, the next steps should be the management of other missions, the implementation of the executive and of the P&S component on actual space processors, and the effective use on board an autonomous spacecraft, for example to manage data downloads in presence of uncertainty about volumes.

## References

Aarts, E., and Lenstra, J., eds. 1997. *Local Search in Combinatorial Optimization.* John Wiley & Sons.

Beaumet, G.; Verfaillie, G.; and Charmeau, M. 2011. Feasibility of Autonomous Decision Making on board an Agile Earth-observing Satellite. *Computational Intelligence* 27(1):123–139.

Charmeau, M.-C., and Bensana, E. 2005. AGATA: A Lab Bench Project for Spacecraft Autonomy. In *Proc. of the 8th International Symposium on Artificial Intelligence, Robotics, and Automation for Space (i-SAIRAS-05).*

Chien, S.; Knight, R.; Stechert, A.; R.Sherwood; and Rabideau, G. 2000. Using Iterative Repair to Improve the Responsiveness of Planning and Scheduling. In *Proc. of the 5th International Conference on Artificial Intelligence Planning and Scheduling (AIPS-00)*, 300–307.

Chien, S.; Cichy, B.; Davies, A.; Tran, D.; Rabideau, G.; Castano, R.; Sherwood, R.; Mandl, D.; Frye, S.; Shulman, S.; Jones, J.; and Grosvenor, S. 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.; Davies, A.; Mandl, D.; Frye, S.; Trout, B.; Shulman, S.; and Boyer, D. 2005b. Using Autonomy Flight Software to Improve Science Return on Earth Observing One. *Journal of Aerospace Computing, Information, and Communication* 2:196–216.

Damiani, S.; Verfaillie, G.; and Charmeau, M.-C. 2004. An Anytime Planning Approach for the Management of an Earth Watching Satellite. In *Proc. of the 4th International Workshop on Planning and Scheduling for Space (IWPSS-04).*

Dechter, R.; Meiry, I.; and Pearl, J. 1991. Temporal Constraint Networks. *Artificial Intelligence* 49:61–95.

Eker, J.; Janneck, J.; Lee, E.; Liu, J.; Liu, X.; Ludvig, J.; Neuendorffer, S.; Sachs, S.; and Xiong, Y. 2003. Taming Heterogeneity: the Ptolemy Approach. *Proceedings of the IEEE* 91(1):127–144.

Feng, T.; Lee, E.; and Schruben, L. 2010. Ptera: An Event-oriented Model of Computation for Heterogeneous Systems. In *Proc. of the 10th International Conference on Embedded Software (EMSOFT-10)*, 219–228.

Gawiejnowicz, S. 2008. *Time-dependent Scheduling*. Springer.

Hentenryck, P. V., and Michel, L. 2005. *Constraint-based Local Search.* MIT Press.

Kramer, L., and Smith, S. 2003. Maximizing Flexibility: A Retraction Heuristic for Oversubscribed Scheduling Problems. In *Proc. of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)*, 1218–1223.

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

Nau, D.; Au, T.; Ilghami, O.; Kuter, U.; Murdock, W.; Wu, D.; and Yaman, F. 2003. SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research* 20:379–404.

Pralet, C., and Verfaillie, G. 2008. Decision upon Observations and Data Downloads by an Autonomous Earth Surveillance Satellite. In *Proc. of the 9th International Symposium on Artificial Intelligence, Robotics, and Automation for Space (i-SAIRAS-08).*

Pralet, C., and Verfaillie, G. 2013. Dynamic Online Planning and Scheduling using a Static Invariant-based Evaluation Model. In *Proc. of the 23rd International Conference on Automated Planning and Scheduling (ICAPS-13).*

Pralet, C., and Verfailllie, G. 2012. Time-Dependent Simple Temporal Networks. In *Proc. of the 18th International Conference on Principles and Practice of Constraint Programming (CP-12)*, 322–338.

Pralet, C.; Verfaillie, G.; Olive, X.; Rainjonneau, S.; and Sebbag, I. 2011. Planning for an Ocean Global Surveillance Mission. In *Proc. of the 7th International Workshop on Planning and Scheduling for Space (IWPSS-11)*.

Verfaillie, G.; Infantes, G.; Lemaître, M.; Théret, N.; and Natolot, T. 2011. On-board Decision-making on Data Downloads. In *Proc. of the 7th International Workshop on Planning and Scheduling for Space (IWPSS-11)*.