ICAPS 2011

21th International Conference on Automated Planning and Scheduling

Proceedings of the

Scheduling and Planning Application woRKshop (SPARK'11)

Freiburg, Germany – 13 June 2011

Edited by

Gabriella Cortellessa, Minh Do, Riccardo Rasconi, and Neil Yorke-Smith

The 2011 ICAPS conference is sponsored by

Adventium Labs
Artificial Intelligence Journal (AIJ)
ATRiCS Advanced Traffic Solutions
David E. Smith
European Coordinating Committee for Artificial Intelligence (ECCAI)
European Office of Aerospace Research & Development
European Space Agency (ESA)
Florida Institute for Human & Machine Cognition (IHMC)
IBM Research
Institute for Computational Sustainability (ICS)
National ICT Australia Limited (NICTA)
National Science Foundation (NSF)
Robert Bosch GmbH
SICK AG
SIFT
TRACLabs Inc.
University of Freiburg, Faculty of Engineering

and held in cooperation with the Association for the Advancement of Artificial Intelligence

Twenty-First International Conference on Automated Planning and Scheduling

Proceedings of the

Scheduling and Planning Application woRKshop (SPARK'11)

Edited by
Gabriella Cortellessa, Minh Do,
Riccardo Rasconi, and Neil Yorke-Smith

SPARK Series Website: http://decsai.ugr.es/~lcv/SPARK/

Proceedings design courtesy of Piergiorgio Bertoli.

Programme Committee

- *Susanne Biundo*, Universitat Ulm, Germany
- *Luis Castillo*, IActive Intelligent Solutions, Spain
- *Steve Chien*, NASA JPL, USA
- *Gabriella Cortellessa*, ISTC-CNR, Italy (co-chair)
- *Andrew Davenport*, IBM T.J. Watson Research Center, USA
- *Mathijs de Weerdt*, TU Delft, The Netherlands
- *Minh Do*, PARC, USA (co-chair)
- *Heng-Soon Gan*, University of Melbourne, Australia
- *Alexandra Kirsch*, Technical University of Munich, Germany
- *Jana Koehler*, Lucerne University of Applied Sciences and Arts, Switzerland
- *Tim Menzies*, West Virginia University, USA
- *Nicola Policella*, ESA-ESOC, Germany
- *Rong Qu*, University of Nottingham, UK
- *Riccardo Rasconi*, ISTC-CNR, Italy (co-chair)
- *Biplav Srivastava*, IBM Research, India
- *Patrik Haslum*, Australian National University, Canberra
- *Roman van der Krogt*, Cork Constraint Computation Centre, Ireland
- *Gerard Verfaillie*, ONERA, France
- *Neil Yorke-Smith*, American University of Beirut, Lebanon, and SRI International, USA (co-chair)

Auxiliary Reviewers

- *Felix Müller*
- *Scott Sanner*
- *Bernd Schattenberg*

Contents

Papers

Preface

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'11 is the fifth 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/

In the attempt to cover the whole spectrum of the efforts in P&S Application-oriented Research, this year's SPARK edition will categorize all contributions in three main areas, namely *P&S Under Uncertainty and Execution*, *Novel Domains for P&S*, and *Emerging Applications for P&S*. 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 Programme Committee and the auxiliary reviewers for their commitment in reviewing. We thank the ICAPS'11 workshop and publication chairs for their support. Finally, we thank the commentators who agreed to lead discussion during the workshop.

Gabriella Cortellessa, ISTC-CNR, Italy
Minh Do, Palo Alto Research Center, USA
Riccardo Rasconi, ISTC-CNR, Italy
Neil Yorke-Smith, American University of Beirut, Lebanon, and SRI International, USA

Co-chairs, SPARK'11

Scheduling a Dynamic Aircraft Repair Shop

Maliheh Aramon Bajestani and **J. Christopher Beck**
Department of Mechanical & Industrial Engineering
University of Toronto
Toronto, Ontario M5S 3G8, Canada
{maramon, jcb}@mie.utoronto.ca

Abstract

We study a dynamic repair shop scheduling problem in the context of military aircraft fleet management. A number of flights, each with a requirement for a specific number and type of aircraft, are already scheduled over a long horizon. The overall scheduling horizon is considered to be composed of multiple time periods. The goal is to maintain a full complement of aircraft. We need to assign aircraft to flights and schedule repair activities while considering the repair capacity limit and aircraft failures. The number of aircraft awaiting repair dynamically changes over time because of aircraft failures. We designed three re-scheduling policies using different optimization and heuristic techniques to solve the dynamic problem over successive time periods. Experimental results demonstrate that the wave coverage is higher if the optimization technique, logic-based Benders decomposition, is used to solve the problem over longer time periods more frequently.

Introduction

In industries using expensive machinery, it is common to repair rather than replace a machine when it breaks down. For example, it is far too expensive for a railroad or airline company to keep stock on-hand to replace failed machines. The need for repair, however, generates a set of new decisions: "How many repair resources (e.g., repairpersons) should be allocated?", "Where should repairs take place?", and "When should they be done and using which resources?". In this paper, we study an aircraft repair shop. When aircraft fail, the management process must dynamically react to failures by scheduling and re-scheduling repair activities to maximize aircraft availability. A high-quality schedule capable of dealing with uncertainty and adjusting to unexpected events leads to an efficient repair operation.

Motivated by the case study in Safaei et al. (Safaei, Banjevic, and Jardine 2010), we address the problem of aircraft fleet management where a number of flights are planned over a long horizon consisting of several time periods. Every flight, called "wave", has a requirement for a specific number of aircraft of different types. Aircraft flow over a long horizon is illustrated in Figure 1.

The goal is to construct a repair schedule that will maximize wave coverage while allowing for aircraft failures in systematic pre- and post-flight checks. Each aircraft failure requires a set of repair activities with known processing times and resource requirements to be scheduled using resources with a limited capacity.

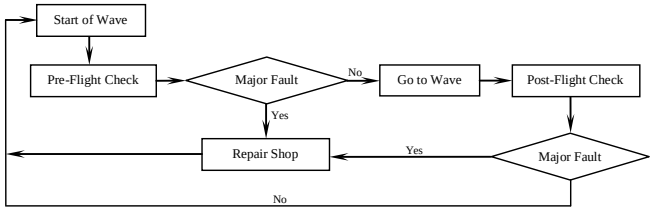


Figure 1: A flow chart representing aircraft flow among waves, checks, and the repair shop over a long horizon.

Simply stated, we view the dynamic problem as successive scheduling problems over time periods. We use three scheduling techniques including logic-based Benders decomposition (LBBD), mixed-integer programming (MIP), and a dispatching heuristic developed in our previous work (Aramon Bajestani and Beck 2011) to solve the scheduling problem in each time period. To revise the schedule, we design three different policies based on when and how the re-scheduling is done.

Empirical studies indicate that solving the scheduling problems more frequently over longer time periods using LBBD results in the best performance and that solving the scheduling problems more frequently is more important than solving them over longer time periods.

The main contributions of our paper are:

- generalizing an offline scheduling problem studied in (Aramon Bajestani and Beck 2011) to its dynamic counterpart;
- demonstrating how to adapt existing solution techniques to a dynamic problem;
- empirically analyzing the impact of applying different re-scheduling strategies to determine when and how to respond to real-time events.

The following section defines our problem, reviews the scheduling algorithms used for each time period and discusses the literature on scheduling a repair system. We go on to describe the proposed re-scheduling strategies, present our experiments and results, suggest possible directions for future work, and provide a conclusion.

Background

In this section, we present the formal definition of the problem, review the solution approaches for scheduling the repair shop, and discuss the literature on scheduling a repair system.

Problem Definition

The problem at time 0 is shown schematically in Figure 2. The circles represent aircraft. It is assumed that the total number of aircraft is constant over a long horizon. In our example, at the beginning, three aircraft are ready for the pre-flight check; others are in the repair shop awaiting repair before they can proceed to the pre-flight check. A number of waves (five are shown) and their corresponding pre- and post-flight checks are already scheduled over a long horizon.

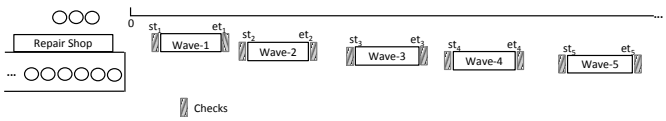


Figure 2: Snapshot of the problem at time 0 over a long horizon.

At the beginning, we schedule the repair activities over a time period, for example the interval containing the first three waves. The schedule may not be executed as is. We dynamically react to actual aircraft failures by re-scheduling the repair activities after, for example, each wave.

The goal in each time period is to assign aircraft to waves to maximize coverage while respecting constraints on maintenance capacity. The coverage is the extent to which the aircraft requirements of the waves are met. The scheduling problem is under the constraints that the repair shop has limited capacity and the aircraft are subject to breakdown which can be detected in pre- or post-flight checks. We assume that once an aircraft fails, it goes to the repair shop and waits until its repair operations are performed.

We use the following notation to represent the problem.

- N is the set of aircraft. λ_n is the failure rate of the aircraft $n \in N$.
- K is the set of aircraft types. For each aircraft type $k \in K$, there are A_k aircraft ready (i.e., not in the repair shop) at the beginning of the first time period. $\bar{\lambda}_k$ is the mean failure rate over all aircraft of type k.
- R is the set of repair resources (called "trades"). The maximum capacity of trade $r \in R$ is C_r.
- W is the set of waves and D is an ordered set of due dates in the time period. D consists of the wave start-times plus a big value, B sorted in ascending order. Each wave, $w \in W$ has a start-time, $st_w \in D$, and an end-time et_w. Each wave requires a_{kw} aircraft of type k.
- J is the set of jobs in the time period. Each job is associated with a specific aircraft, and I_k denotes the set of repair jobs for aircraft type k. M_r is the set of jobs requiring trade r. Each job might require more than one trade to be completed. The processing time of job j on trade r is p_{jr} and c_{jr} is the capacity of trade r required by job j.

To understand how the dynamic problem over the long horizon can be viewed as scheduling problems over successive time periods, assume that we start repairing the failed aircraft and assigning them to the waves based on the computed schedule at time 0. A wave might start while a repair is under way in the repair shop. If some aircraft fails the pre-flight check, it goes to the repair shop. Each failed aircraft requires a set of repair activities with known processing times and resource requirements. At the repair shop, some of the previously failed aircraft might be already repaired, some might be under repair, and others might be awaiting repair. Once the failed aircraft enter the repair shop, we have a new repair scheduling problem with a new set of jobs, including the recently failed aircraft and the previously failed aircraft whose repairs are still under way or are not yet started. The new problem has an added constraint, namely that the repairs currently under way cannot be disrupted.

Scheduling for a Time Period

This section reviews the details of mixed-integer programming, logic-based Benders decomposition, and the dispatching heuristic presented in (Aramon Bajestani and Beck 2011).

Mixed Integer Programming The variables are defined in Table 1 and the model is shown in Figure 3.

Var.	Definition
Z_{kw}	The number of aircraft of type k assigned to fly in wave w
x_{ij}	$x_{ij} = 1$ if the ith due date is assigned to job j
st_{jr}	The start-time of job j on trade r
U_{kw}	The number of aircraft of type k whose repair due date is st_w
E_{kw}	The expected number of available aircraft type k for wave w
et_{jr}	The end-time of job j on trade r

Table 1: The decision variables (top) and inferred variables (bottom) for the MIP model.

The objective function (1) maximizes the number of aircraft assigned to a wave subject to a limit on the number of aircraft required and the expected number available (Constraint (5)). Equation (2) calculates the number of aircraft of type k whose repair due date is st_w. Equation (3) calculates the expected number of available aircraft for the first wave. Equation (4) calculates the expectation of availability for the other waves. The first term includes those aircraft available but not used for the previous wave and those newly arrived from the repair shop. The second term includes all aircraft that are available because they have completed waves since the previous wave started. ξ_k^{pre} and ξ_k^{post} denote the probability of failure associated with aircraft type k in pre- and post-flight checks, respectively: $\xi_k^{pre} = (1 - e^{-\alpha\bar{\lambda}_k})$ and $\xi_k^{post} = (1 - e^{-\beta\bar{\lambda}_k})$, where $\alpha < \beta$ to reflect deterioration of the aircraft through use. As tracking the history of the aircraft is prohibitive to find the actual probability of failure for each, they are distinguished based on their type and the failure rate of each aircraft type $\bar{\lambda}_k$ is used to estimate the probability of failure. Constraint (6) ensures that exactly one due date is assigned to each job. Equation (7) calculates

$$\text{max.} \sum_{w=1}^{W} \sum_{k=1}^{K} Z_{kw} \quad (1)$$

$$\text{s.t. } U_{kw} = \sum_{j \in I_k} x_{ij}, \text{ if } d_i = st_w \quad (2)$$

$$E_{k1} = (A_k + U_{k1})(1 - \xi_k^{pre}), \ \forall k \quad (3)$$

$$E_{kw} = (E_{k(w-1)} - Z_{k(w-1)} + U_{kw})(1 - \xi_k^{pre}) + \sum_{v=1}^{w-1} Z_{kv}(1 - \xi_k^{post})(1 - \xi_k^{pre}),$$
$$\text{if } st_{w-1} < et_v \leq st_w, \forall w \neq 1, k \quad (4)$$

$$Z_{kw} \leq \min(E_{kw}, a_{kw}), \ \forall k, w \quad (5)$$

$$\sum_{i=1}^{|D|} x_{ij} = 1, \ \forall j \quad (6)$$

$$st_{jr} + p_{jr} = et_{jr}, \forall j, r \quad (7)$$

$$et_{jr} \leq \sum_{i=1}^{|D|} x_{ij} d_i, \ \forall j, r \quad (8)$$

$$\sum_{j \in M_r} c_{jr} \leq C_r, \text{if } st_{jr} \leq t < et_{jr}, \ \forall t, r \quad (9)$$

$$x_{ij} \in \{0, 1\}, \ \forall i, j \quad (10)$$

$$0 \leq E_{kw} \leq |N|, \ \forall k, w \quad (11)$$

$$st_{jr}, et_{jr} \in \mathbb{Z}^+ \cup \{0\}, \ \forall j, r \quad (12)$$

$$Z_{kw} \in \mathbb{Z}^+ \cup \{0\}, Z_{kw} \leq |N|, \ \forall k, w \quad (13)$$

Figure 3: The global MIP model for one time period.

the end-time of the jobs. The end-time of each job is guaranteed to be less than or equal to the assigned due date by constraint (8). Constraint (9) enforces the capacity limit of each trade, where t denotes the discrete time during which job j is under way.

Logic-based Benders Decomposition A logic-based Benders decomposition (LBBD) method can be formulated where the master problem assigns aircraft to waves to maximize wave coverage over the current time period and the sub-problems create the maintenance schedules given the due dates assigned by the master problem solution. The master problem is solved using MIP, while constraint programming (CP) is used for the scheduling sub-problems.

The Due-Date Assignment Master Problem (DAMP): MIP Model To formulate the master problem as a MIP model, we use a binary variable x_{ij} for each $j \in J$ and $i \in D$ with the same meaning as in the global MIP model. A MIP formulation of DAMP is as follows:

max. Objective (1)

s.t. Constraints (2) to (6), (10), (11), (13)

$$\sum_{j \in M_r, \ \sum_{i=1}^{|D|} x_{ij} d_i \leq st_w} c_{jr} p_{jr} \leq st_w C_r, \quad \forall r, w \quad (14)$$

MIP cuts (15)

The master problem incorporates a number of the constraints in the global MIP model. It does not represent the start-times of jobs nor does it fully represent the capacity of the trades. As is common in Benders decomposition, the master problem includes a relaxation of the sub-problems (Constraints (14)) and Benders cuts (Constraints (15)).

The Sub-problem Relaxation Constraint (14) is the relaxation of the capacity of a trade, expressing a limit on the area of jobs that can be executed. The limit is defined using the area bounded by the capacity of the trade and the time intervals $[0, st_w]$ for each wave w, plus $[0, B]$ where B is the maximum due date assigned to the jobs on the trade. The area of each interval must be greater than or equal to the sum of the areas of the jobs that finish by the end of the interval.

The Benders Cuts We demonstrate the intuition with an example before defining the cut. Assume that for a given trade with five jobs and a due date set, $D = \{14, 17, 20, 100\}$, the current master solution is: $x_{21} = 1, x_{12} = 1, x_{43} = 1, x_{14} = 1,$ and $x_{15} = 1$. Job 1 is assigned to the second due date, 17, Job 2 has the first due date, 14, and so on. If the current solution is infeasible due to the resource capacity of the trade, then we know that at least one of the jobs must have a later due date. We can, therefore, constrain the sum of the consecutive x_{ij} up to and including the ones assigned to 1 to be one less than the number of jobs. In our example, the cut would be:

$$(x_{11} + x_{21}) + (x_{12}) +$$
$$(x_{13} + x_{23} + x_{33} + x_{43}) + (x_{14}) + (x_{15}) \leq 5 - 1$$

Formally, assume that in iteration h, the solution of the DAMP assigns a set, Q, of due dates to the jobs on trade r. Assume further that there is no feasible solution on trade r with the assignments in Q.

The cut after iteration h is:

$$\sum_{j \in M_r} \sum_{i \in I_{jh}^r} x_{ij} \leq |M_r| - 1, \quad \forall r \quad (16)$$

where $I_{jh}^r = \{i' | i' \leq i, \text{ and } x_{ij}^h = 1\}$ is the set of due dates indices less than or equal to the due date index assigned to job j and $|M_r|$ is the number of jobs on trade r.

Job Scheduling Sub-problem Given a set of due dates assigned to the jobs on a trade, the goal of the job scheduling sub-problem (JSSP) is to assign start-times to the jobs to satisfy the due dates and the trade capacity. The JSSP for each trade can be modeled using cumulative constraints (Hooker 2005). We use a CP formulation:

$$\text{Cumulative}([t_j|d_j^h],[p_{jr}|d_j^h],[c_{jr}|d_j^h],C_r),\ \forall r$$
$$0 \le t_j \le d_j^h - p_{jr},\ \forall j,r \quad (17)$$

where t is an array of variables such that t_j is the start-time of job j, d is an array of values such that d_j^h is the due date assigned to job j in master problem in iteration h. The variables p_{jr}, c_{jr}, C_r are as defined above. Constraint (17) enforces the time windows: the job cannot be started later than $d_j^h - p_{jr}$.

A Dispatching Heuristic The dispatching heuristic, inspired by the Apparent Tardiness Cost (ATC) heuristic (Pinedo 2005), is a list-scheduling heuristic. It prioritizes repair activities based on how early the corresponding aircraft type is needed, the processing time of each job, and the relative type demand. The ranking index we use is as follows:

$$I_j = ST(k_j)\exp(-\frac{FN_j}{FC_j}), \quad \forall j$$

If we let k_j denote the type of aircraft j, then $ST(k_j)$ is the start-time of the first wave that requires an aircraft of type k_j. FN_j is the fraction of the total number of aircraft of type k_j required by the first wave that requires k_j, and FC_j is the maximum proportion of the capacity needed by job j over all its required trades, as follows.

$$FC_j = \max_r(\frac{p_{jr}c_{jr}}{ST(k_j)C_r}), \quad \forall r$$

The heuristic sorts the jobs in ascending order of the index and then iterates through the jobs, scheduling each job at its earliest available time.

Literature Review

Queuing theory is often used to model repair systems [(Iravani, Krishnamurthy, and Chao 2007) and the references therein]. Queuing theory has a long-term definition of optimality resulting in some sort of repair policy commonly assuming that the repair resources are unary capacity (i.e., one repair is carried out at a time). A repair policy determines the order under which the repair activities should be carried out.

Queuing theory does not model the combinatorics of the scheduling problem. In our problem, the optimization of scheduling performance at discrete time points (i.e., before each flight) is of interest, and the repair resources have a discrete capacity. Therefore, we believe that better performance can be achieved by dealing directly with the combinatorics and explicitly scheduling the repair shop to meet the waves.

Dynamic scheduling is well-suited to handle the uncertain and combinatorial structure of the scheduling problem. Dynamic scheduling concerns the allocation of resources to activities over time when the real-time events occur during the execution of previously determined schedule (Aytug et al. 2005).

The real-time event studied in this paper is a job-related event (Vieira, Hermann, and Lin 2003) because of the uncertainty involved in the systematic pre- and post-flight checks. Some of the repaired aircraft cannot accomplish their assigned flight: they are diagnosed as failed and must return to the repair shop.

When and how to respond to the real-time events are two independent variables in dynamic scheduling problems addressed in (Vieira, Hermann, and Lin 2003; Aytug et al. 2005; Bidot et al. 2009). In our problem, the length of each time period, and the frequency of re-scheduling determine how and when we react to the aircraft failures, respectively.

Re-scheduling Strategies

Our strategies have three main parts: scheduling the repair activities, observing the aircraft failures while executing the computed schedule, and responding to dynamic events by re-scheduling the repair activities. They start by scheduling the repair activities over one time period at time 0. The length of time period defines the scheduling horizon over which the repair activities are scheduled. The re-scheduling strategies start executing the repair schedule while observing the aircraft failures. The frequency of re-scheduling determines when our strategies dynamically respond to the aircraft failures.

We use the techniques reviewed in the Background section to schedule the repair activities over one time period. Three different policies denoted as P_{ij} are designed in which i and j define the length of scheduling horizon and the frequency of re-scheduling in number of waves, respectively.

The three policies discussed here are:

- P_{11}: This policy has a scheduling horizon with a length of one wave and re-schedules after every wave. In Figure 4, we show that P_{11} schedules one wave at a time ($i = 1$) and re-schedules after each wave ($j = 1$).
- P_{31}: This policy has a scheduling horizon with a length of three waves and re-schedules after every wave. In contrast to P_{11}, for P_{31} (Figure 5), the scheduling horizon is three waves but re-scheduling is still done after each wave.
- P_{33}: This policy has a scheduling horizon with a length of three waves and re-schedules after every third wave (Figure 6).

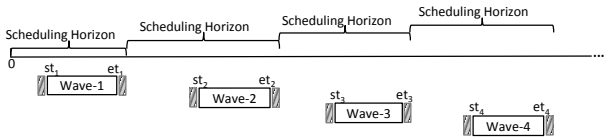


Figure 4: The P_{11} policy.

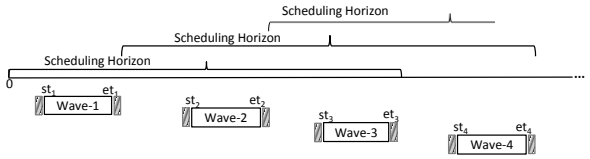


Figure 5: The P_{31} policy.

To model the dynamic events, we simulate the aircraft failures in pre- and post-flight checks. Every aircraft either passes or fails each check. If the aircraft fails, a new

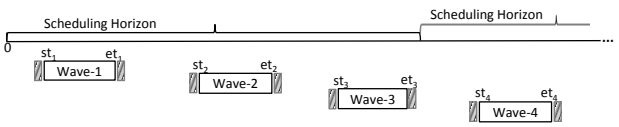


Figure 6: The P_{33} policy.

set of repair activities with known processing times and resource requirements is added to the repair shop. If the aircraft passes, it flies the wave. To model the aircraft deterioration, we increase the failure rate of the aircraft by γ percent each time it flies a wave. For example, consider that λ_n is the initial failure rate of the aircraft $n \in N$. Its failure rate after flying t waves will be equal to $\lambda_n(1+\gamma)^t$.

The observed wave coverage for each wave is the number of aircraft flying the wave divided by the number required.

Experimental Results

The next sub-section describes the problem instances and the experimental details. We then compare the impact of using different scheduling techniques and re-scheduling policies on the observed wave coverage.

Experimental Setup

For our problem instances, the number of aircraft, the number of trades, and the total number of waves are set to $\{10, 15, 20, 25, 30\}$, $\{4\}$, and $\{30\}$ respectively. Each combination has 5 instances for a total of 25 instances. Each instance is simulated 10 times.

Aircraft The number of aircraft types is equal to $\frac{|N|}{5}$, where $|N|$ is the number of aircraft. The aircraft are randomly assigned to different types, and the initial failure rate for each aircraft is randomly chosen from the uniform distribution of $[0, 0.5]$. The failure rate of an aircraft is increased by $\gamma = 5\%$ each time it is used. The values of α and β are 1 and 3, respectively (see the Mixed Integer Programming section).

Trades The capacity limit for each trade is $C_r = 10$.

Repair Jobs The repair jobs at time 0 each require half the trades, on average. Subsequent repair jobs require all trades. This difference was done to have enough repair jobs for the successive scheduling problems. The capacity of trade r used by job j, c_{jr}, is drawn from $[1, 10]$ while the processing time, p_{jr}, is drawn from $[r, 10r]$. At time 0, having 80% of the aircraft in the repair shop results in $|J| = 0.8|N|$ repair jobs.

Waves The plane requirement for each wave is randomly generated from the integer uniform distribution $[1, a_k]$ where a_k denotes the number of aircraft of type k. The length of each wave is drawn with uniform probability from $[3, 5]$. To find an appropriate start time for the first wave (not too early or too late) and subsequent waves, $T = 1.2 \times LB$ is defined where $LB = \max_r(S_r)$. The sum of the processing areas of the jobs in each trade, r, divided by the trade capacity is denoted by S_r. The processing areas in each trade are summed over the jobs in the repair shop at time zero. The start time of each wave is generated as $st_1 = rand[\frac{T}{3}, \frac{T}{2}]$ for the first wave, and $st_w = et_{w-1} + rand(0, 40)$ for $1 < w \leq 30$. As mentioned earlier the total number of waves is 30.

Dynamic events To simulate an aircraft failure, we generate a random value from the uniform distribution $[0, 1]$ for each aircraft at each check. If the random value is less than the aircraft's probability of failure, the aircraft fails; otherwise, it passes. The aircraft's probability of failure in pre- and post-flight checks are calculated using $(1 - e^{-\alpha\lambda_n})$ and $(1 - e^{-\beta\lambda_n})$, respectively. As mentioned earlier, λ_n is the failure rate of aircraft $n \in N$ which increases by $\gamma = 5\%$ each time the aircraft flies a wave. Note that, passing the pre-flight check of a wave does not necessarily mean that the aircraft flies the wave. If the number of available aircraft are more than the requirements, the aircraft that fly are randomly selected to meet the requirements.

The time-limit to schedule the repair activities in each scheduling horizon is 600 seconds. We execute the best feasible schedule found before the time-limit if MIP times out. In the case that LBBD times out, the schedule created by the dispatching heuristic is executed as LBBD cannot create a feasible schedule when it times-out.

The scheduling uses IBM CPLEX 12.1 and IBM ILOG Solver/Scheduler 6.7, and the simulation is coded in C++.

Computational Results

In this section, we discuss our results to answer a number of different questions.

Question 1 What is the impact of using a complete technique vs. the dispatching heuristic on the mean observed wave coverage?

We expect a complete technique to achieve higher wave coverage because it incorporates known information on uncertainty into scheduling the repair activities, while the dispatching heuristic does not have this property.

Figure 7 shows the mean observed coverage up to wave $w \in \{1, 2, ..., 25\}$ for different scheduling techniques over all three policies. The mean observed coverage up to wave w is $O_w = \frac{\sum_{i=1}^{w} \nu_i}{w}$, where ν_i denotes the coverage of wave i. As illustrated, LBBD achieves about a 40% higher mean coverage over all waves than either MIP or the dispatching heuristic. The MIP algorithm also takes the probabilistic information into account creating a repair schedule but it times out on 72% of scheduling problems without finding a feasible solution. The dispatching heuristic then is used to create the repair schedule. Therefore, using MIP and the dispatching heuristic results in waves with almost the same coverage as the dispatching heuristic alone.

Table 2 presents further data for all scheduling techniques: the mean observed coverage, the percentage of waves with less than or equal to 0.3 coverage, and the percentage of waves with more than or equal to 0.7 coverage. The data indicate the clear superiority of LBBD over the dispatching heuristic.

Question 2 Does P_{31} policy provide the waves with higher coverage than P_{33} and P_{11} policies using the optimization

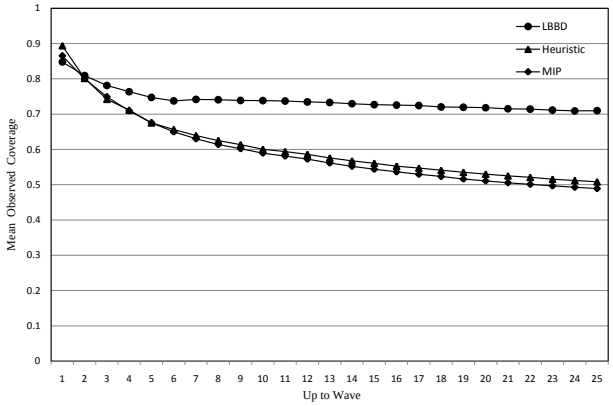


Figure 7: Mean observed coverage for different scheduling techniques.

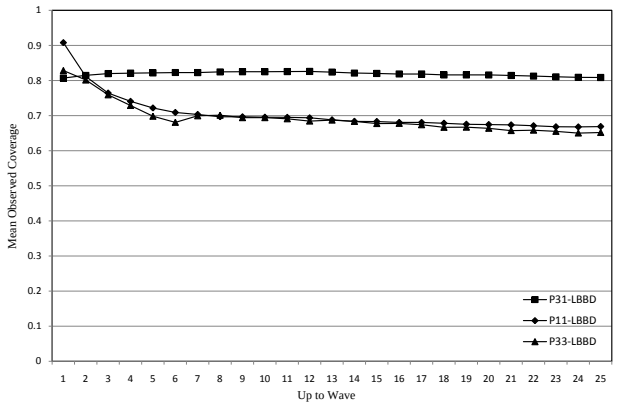


Figure 8: Mean observed coverage for different policies using LBBD.

Scheduling Technique	Mean Observed Coverage	% Waves with a Coverage ≤ 0.3	% Waves with a Coverage ≥ 0.7
LBBD	0.71	6.07	55.80
MIP	0.49	27.31	21.40
Heuristic	0.51	26.39	26.43

Table 2: The mean observed coverage, the percentage of waves with less than or equal to 0.3 coverage, and the percentage of waves with more than or equal to 0.7 coverage up to wave 25 over all the policies.

technique LBBD?

We expect that P_{31} with LBBD will produce better coverage because it schedules over a longer horizon and adjusts the schedule as soon as aircraft failures occur. Although P_{31} with the dispatching heuristic also responds quickly to the aircraft failures, it does not incorporate the length of the scheduling horizon into the ranking index for repair activities and always repair the aircrafts for the earliest future.

Figure 8 shows the mean observed coverage for different policies using LBBD up to wave $w \in \{1, 2, .., 25\}$. The P_{31} policy leads to consistently higher coverage.

Figure 9 displays the cumulative percentage of the waves with a coverage less than or equal to ω for LBBD and the dispatching heuristic, where ω denotes the values on the x-axis. The best performing approach will have a fewer waves with a low coverage and more waves with a high coverage. Therefore, its curve will be closer to the lower right-hand corner. As illustrated, in LBBD, P_{31} performs better than the two other policies. In contrast, in the dispatching heuristic, P_{31} results in waves with the same coverage as P_{11}.

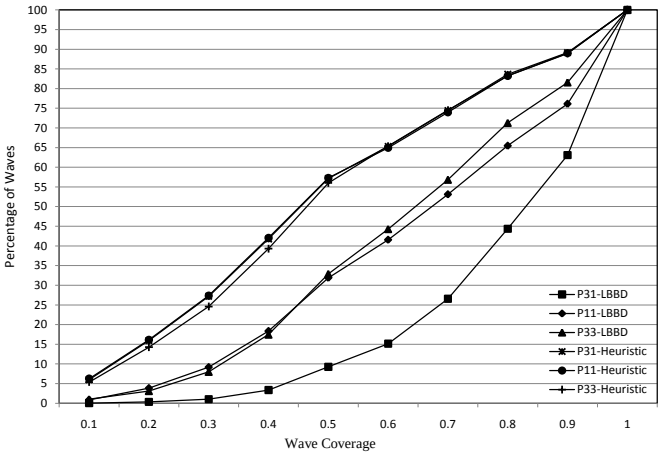


Figure 9: The percentage of waves with a coverage less than or equal to ω, where ω denotes the values on the x-axis.

Question 3 Does P_{11} have more waves with very low coverage than P_{33} using LBBD?

We expect that P_{33} will result in fewer waves with very low coverage beacuse it takes the possibility of a more distant future into account when creating the repair schedule, while P_{11} policy repairs the aircraft at the earliest possible time.

Figure 10 demonstrates that our intuitions are correct that the P_{33} policy has fewer waves with very low coverage than the P_{11} policy using LBBD.

Question 4 Is a quicker reaction to the dynamic events more important than scheduling over a longer horizon?

The P_{31} policy changes the repair schedule after each wave and trades-off the coverage among three consecutive waves by scheduling over a longer horizon. In contrast, the P_{11} policy schedules for one wave and reacts after each wave while the P_{33} policy reasons over a longer term without a quick response to the dynamic events.

As already shown in Figure 8, the P_{31} policy results in a higher mean coverage. The superiority of policy P_{31} indicates that both features of quick response to the dynamic events and long-term reasoning contribute to the overall performance, but the question is which one contributes more.

To answer the question, the waves are partitioned into buckets of size 3. We expected that P_{33} would achieve a higher mean coverage over each bucket than P_{11} because it reasons about the trade-off among the three waves. However, Figure 11 demonstrates that both policies achieve equal performance until wave 15 and the policy P_{11} then does better. This observation indicates that quick reaction to the air-

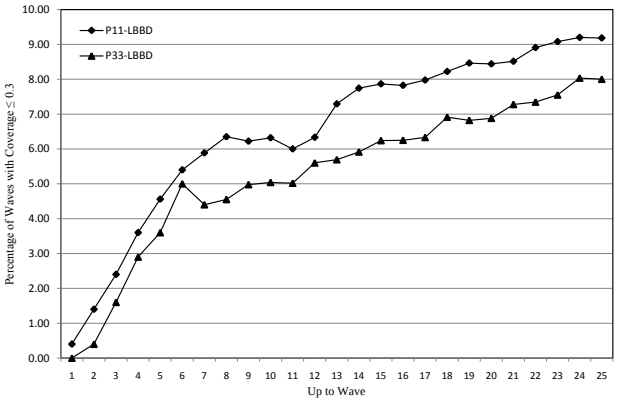


Figure 10: Percentage of waves with a coverage less than or equal to 0.3 using LBBD.

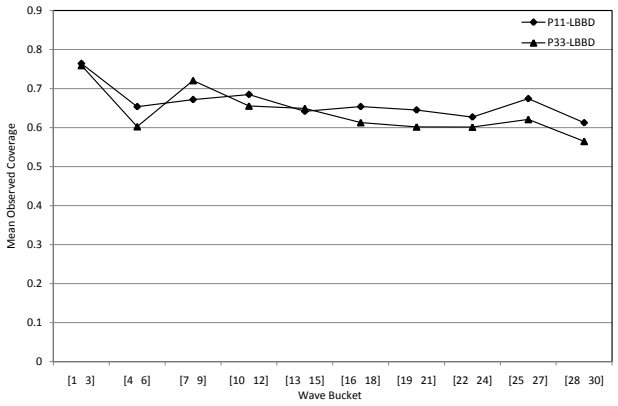


Figure 11: Mean observed coverage over each wave bucket using LBBD.

craft failures has more significant impact on the observed coverage than long-term reasoning.

To obtain more insight, the mean observed coverage for the first, second, and third waves in each bucket is shown in Figures 12, 13 and 14, respectively. As illustrated, P_{11} results in a higher coverage than P_{33} for the waves scheduled later in each bucket. The mean difference between two policies ($P_{11} - P_{33}$) is equal to -5, 2, and 10 percent for the first, second, and third waves in buckets, respectively. Intuitively, we expect that P_{11} provides the earlier waves in buckets with a higher coverage than P_{33} because the later policy trades off between three waves when assigning the aircraft to the waves. However, the observation contradicts our expectation and is an evidence that the quick reaction is more important than scheduling over longer horizon.

Summary The following conclusions are supported by empirical observations:

- LBBD provides the waves with a higher coverage than the dispatching heuristic.
- The P_{31} policy results in waves with a higher coverage than P_{11} policy using LBBD and with the same coverage using the dispatching heuristic.
- The P_{33} policy is shown to result in fewer waves with very low coverage than P_{11} policy using LBBD.
- Scheduling over a longer horizon and quickly adjusting the schedule based on the real events are the features contributing to the increase in the observed coverage. Furthermore, it is shown that the quick reaction to the dynamic events is more important than the long scheduling horizon.

Future Work

Although the experiments show that scheduling over a longer horizon and responding quickly to disruptions using optimization techniques in a dynamic and uncertain environment yield better performance, the generality of this observation remains in question. One promising direction in future work would be to establish a formal framework to determine how long the scheduling horizon should be and how quickly we should respond to real events. Bidot (Bidot 2005) presented the first steps toward a generic theoretical framework in his PhD thesis.

In this paper, to measure system performance, we have focused on the value of mean observed coverage, not taking into account the computational cost of applying different policies. Although P_{33} is shown to be the dominant policy, it has a greater computational cost and potentially increases the unnecessary changes in the schedule (Aytug et al. 2005). Optimizing over a longer scheduling horizon and rescheduling repair activities once failed aircraft enter the repair shop explain the high computational cost and the increased unnecessary changes, respectively. How to quantify costs in order to evaluate different policies is another interesting direction for future work.

Conclusion

In this paper, we address a dynamic aircraft scheduling problem in a repair system. The goal is to meet the aircraft requirements for each wave by assigning the failed aircraft to the flights considering the maintenance capacity and the aircraft failures. The number of failed aircraft dynamically changes because of aircraft breakdowns. Our proposed solution approaches solves the dynamic problem as successive scheduling problems over multiple time periods. We use three different scheduling techniques developed in our previous work and three re-scheduling policies to schedule the repair activities on-line with dynamic reaction to the aircraft failures. The length of the scheduling horizon and the frequency of re-scheduling are the features defining our three policies.

The computational results show that an optimization approach using logic-based Benders decomposition, scheduling over a longer horizon, incorporating the known information on aircraft failures, and adjusting the repair schedule as soon as new jobs enter the repair shop yield higher mean coverage. The results also provide evidence that quick reaction to the aircraft failures is more important than scheduling over a longer horizon as the policy with a higher frequency of re-scheduling does better than the policy with

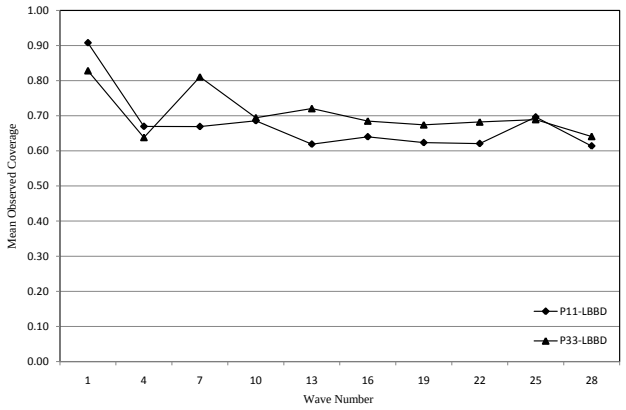


Figure 12: Mean observed coverage for the first waves in buckets.

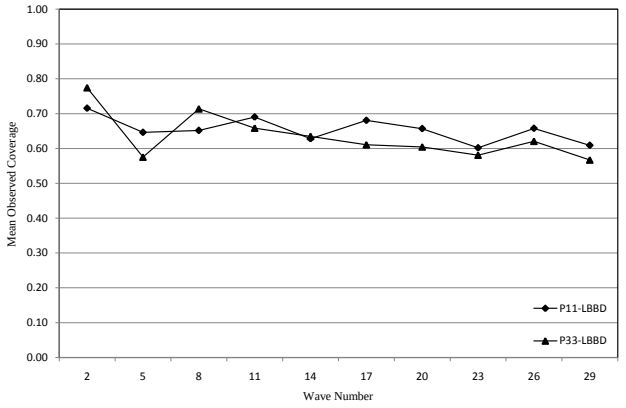


Figure 13: Mean observed coverage for the second waves in buckets.

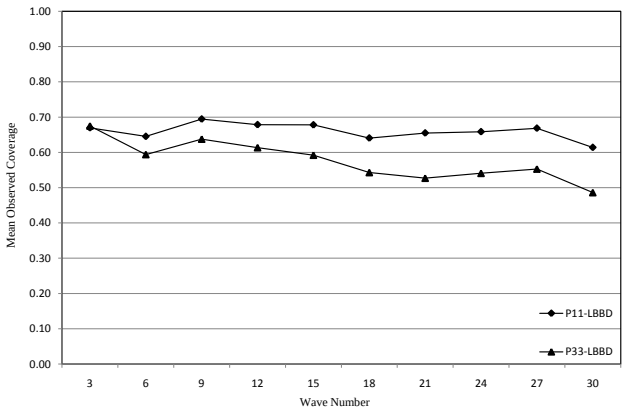


Figure 14: Mean observed coverage for the third waves in buckets.

longer scheduling horizon on the waves scheduled later in each time period.

References

Aramon Bajestani, M., and Beck, J. C. 2011. Scheduling an aircraft repair shop. In *Proceedings of the Twenty-First International Conference on Automated Planning and Scheduling (ICAPS'11)*.

Aytug, H.; Lawley, M.; McKay, K.; Mohan, S.; and Uzsoy, R. 2005. Executing production schedules in the face of uncertainties: A review and future directions. *European Journal of Operational Research* 161:86–110.

Bidot, J.; Vidal, T.; Laborie, P.; and Beck, J. 2009. A theoretical and practical framework for scheduling in a stochastic environment. *Journal of Scheduling* 12:315–344.

Bidot, J. 2005. *A general framework integrating techniques for scheduling under uncertainty*. Ph.D. Dissertation, Ecole Nationale d'Ingenieurs de Tarbes, France.

Hooker, J. 2005. A hybrid method for planning and scheduling. *Constraints* 10:385–401.

Iravani, S. M. R.; Krishnamurthy, V.; and Chao, G. 2007. Optimal server scheduling in nonpreemptive finite-population queueing systems. *Queueing System* 55:95–105.

Pinedo, M. L. 2005. *Planning and Scheduling in Manufacturing and Services*. Springer Series in Operations Research. Springer.

Safaei, N.; Banjevic, D.; and Jardine, A. K. S. 2010. Workforce constrained maintenance scheduling for aircraft fleet: A case study. In *Proceedings of Sixteenth ISSAT International Conference on Reliability and Quality in Design*, 291–297.

Vieira, G.; Hermann, J.; and Lin, E. 2003. Rescheduling manufacturing systems: a framework of strategies, policies and methods. *Journal of Scheduling* 6:36–92.

Opening the PANDORA-BOX: Planning and Executing Timelines in a Training Environment

Amedeo Cesta, Gabriella Cortellessa
& **Riccardo De Benedictis**
CNR – Consiglio Nazionale delle Ricerche
Istituto di Scienze e Tecnologie della Cognizione
Rome, Italy
{name.surname}@istc.cnr.it

Keith Strickland
Cabinet Office
Emergency Planning College
York, UK
keith.strickland@cabinet-office.x.gsi.gov.uk

Abstract

This paper introduces a novel use of timeline-based planning as the core element within a dynamic training environment designed for crisis managers. Training for crisis decision makers at the strategic level poses a number of challenges that range from the necessity to foster creative decision making to the need for the creation of engaging and realistic scenarios in support of experiential learning. This article describes our efforts to build an end-to-end system, called the PANDORA-BOX, that helps the trainer to populate and deliver a continuous 4-5 hours training session encompassing exercises that encourage a group of decision makers to achieve joint decisions. Specifically the emphasis is given to (a) the timeline-based representation as the core component for creating training sessions and unifying different concepts of the PANDORA domain; (b) the combination of planning and execution functionalities required to maintain and dynamically adapt a "lesson plan" on the basis of both trainee-trainer interaction and individual behaviors and performance; (c) the importance of keeping the trainer in close control of the activity loop.

Introduction

When a major incident or catastrophic event occurs, it is often human behavior alone that determines the speed and efficacy of the crisis response management arrangements. Indeed, all too often, shortcomings in the response to the emergency do not stem from ignorance of procedures but from difficulties inherent within the challenge presented when operating in traumatic circumstances, particularly when additional unexpected consequences arise. Effective crisis management is a key requirement to prevent an emergency from becoming a disaster. In recent years, poor management in response to an emergency has often resulted in critical situations becoming far worse. Furthermore, crisis events appear to occur more and more frequently and public expectation for an effective and immediate response grows at a similar pace. Thus managers, especially senior managers, have to cope almost routinely with crisis decision situations, given that we are now leaving in a "risk society" (Beck 1992). In these critical circumstances, there is a tremendous necessity to have effective leadership in place. Nevertheless, the ambiguity, urgency and high risk associated with crisis situations posits some constraint on the leadership capabilities. For example, given the need for an almost immediate and of course effective response to a crisis, there is little time to acquire and process effectively all the information that decision makers would wish to have available to them. As a consequence, they are required to assess information and make critical decisions under tremendous psychological stress and physical demands (Klann 2003; Leonard 2004), often caused by the difficulty of operating in a context where losses, including both human lives and critical resources, continue to rise until such time as the response can get ahead of the developing crisis.

Within this context training plays a crucial role in preparing crisis managers. Specifically, training for strategic decision making has to foster the leaders' ability to anticipate the possible consequences of poor decisions and to construct creative solutions to problems. In this light, experiential learning plays a crucial role. A great amount of investment is being devoted to the development of training procedures to increase the capability of crisis managers to deal with emergency situations. Two main modalities are mainly used: (a) the table top exercise (a group discussion guided by a simulated disaster); (b) a real world simulation exercise (field tests replicating emergency situations). Table top exercises are generally low cost and can be easily and frequently organized, but they cannot recreate the real atmosphere, in terms of stress, confusion and pressure. On the other hand, crisis managers trained through simulation exercises in the field can be very effective and can gain valuable skills, but such simulations are very expensive in both time and resources and cannot be easily and quickly organized.

The PANDORA project[1] aims to bridge the gap between tabletop exercises and real world simulation exercises by providing a near-real training environment at affordable costs. PANDORA's goal is to simulate all the dynamic elements contained within an entire disaster scenario within a training room setting that emulates an engaging, true-life environment. The system will be capable of presenting different evolving crisis scenarios, customized to meet specific and specified training needs according to the knowledge and experience levels among the participating students. A key aspect in PANDORA is the ability to create realistic consequence responses to the decisions taken by trainees thus reproducing realistic situations and facilitating the development of a comprehensive range of decision making skills. Additionally, the idea underpinning PANDORA is to take ac-

[1] http://www.pandoraproject.eu/

count of human behaviors and individual personalities in order to plan training sessions that recognize individual traits and training needs.

We have produced a first version of the whole system architecture, called the PANDORA-BOX, that fully demonstrates the feasibility of our approach. Central to the PANDORA system is an original use of the timeline-based planning (Muscettola 1994) to model a rich and unconventional domain. Specifically, planning is used (1) to compute diversified crisis scenarios corresponding to alternative training paths to foster creative decision-making, (2) to model and maintain trainees' behavioral patterns according to which training can be personalized, (3) to support mixed-initiative interaction between the trainer and the automated learning environment relying on a high level of abstraction for the internal representation.

The remainder of this paper covers: presentation of the main challenge surrounding the training of decision makers at the strategic level; the general building blocks of the learning environment and the issues arising from the creation of an immersive training experience. A software architecture for the PANDORA-BOX is introduced and the role of a timeline-based representation as the core component for creating training sessions is underscored. Finally, a combination of planning and execution functionalities that allows for the maintenance and adaptation of a "lesson plan" to enable trainer-trainee interaction is described.

Training for crisis decision makers

When referring to planning connected to crisis management during emergency situations, we have in mind the intervention plans for those people that go directly to the operational level of response, see (Wilkins et al. 2008). In reality there are distinctly different levels of decision making all of which are relevant in any crisis situation. The success of crisis management often depends not only on the ability to apply well established procedures, but also on the effectiveness of high-level strategic choices. The ability of decision makers to anticipate the possible consequences of their actions (decisions) by means of flexible and forward-looking reasoning is also crucial to an effective response to a crisis. Figure 1 summarizes the three different levels corresponding to different roles of crisis decision makers:

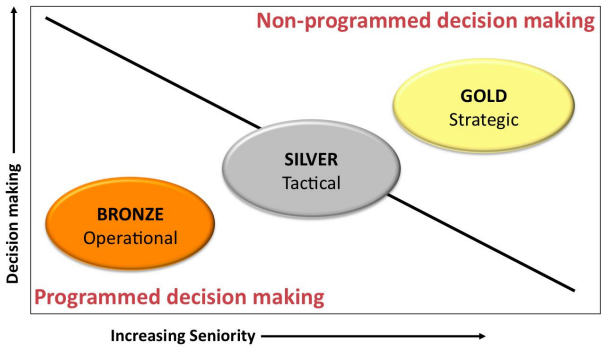


Figure 1: *Different decision makers in crisis management.*

– At the **operational level** we have the operational or **bronze level commanders**, people operating within the detailed area of a crisis situation that perform practical activities and actions, the results of which are monitored and communicated to higher levels;
– At the **tactical or silver level** decision makers that are located close to but not within affected areas of the crisis are responsible for translating high level strategic decisions into actions by allocating tasks and resources down to the bronze level. At this level the anticipated results from the various allocated tasks are monitored and assessed for effectiveness.
– The **strategic or gold level** commanders identify the key issues of a critical situation and prioritize required activity from a detached and sufficiently high level of abstraction. Strategies for resolving the crisis are also decided and are then communicated to the lower levels for their detailed specification and implementation.

The choices at the strategic level are particularly important and critical for the success of the overall crisis response and specifically for devising strategies to contain and correct the developing situation by anticipating future consequences with decisions that try to avoid escalating of the crisis situation.

Also depicted in Figure 1 are the different roles of crisis managers corresponding to the different decision-making levels. Specifically, at the strategic level, decision making is mainly unstructured and not describable in terms of programmed or fixed procedures, being mainly related to the novelty and unpredictability of a catastrophic event. Taken in this view, it is therefore assessed to be non-programmed decision making effort unlike the silver and bronze levels that will respond to higher level tasking and direction in a disciplined and procedural manner.

Most of the state-of-the-art training support systems and simulators are aimed at the operational or tactical levels. PANDORA however is specifically targeted towards strategic level decision makers thus presenting difficult challenges at both modeling and computational levels. Additional challenges arise from the need to foster quick decision making in stressful conditions and the need to encourage creative thinking to devise workable strategies to deal with uncommon situations.

Among the main objectives for gold commanders during a crisis are: protection of human life and, as far as possible, property; alleviation of suffering; support for the continuity of everyday activity; the restoration of disrupted services at the earliest opportunity; upholding the rule of law and the democratic process. The speed with which recovery strategies are identified to contain and resolve the crisis also has a great influence on the scale of loss of whatever nature. For this reason the strategic decision maker has to develop an ability to quickly react and decide to promote the overall goal of obtaining a rapid return to normality.

In this light, training plays a fundamental role. At the strategic level, training aims to teach decision-makers to focus on the possible consequences of their actions. It also teaches the value of integrating and testing the compatibility of plans and the need to work in collaboration with other organizations and between different nations, to promote continuity of efforts and to have a well-defined focus.

One approach to decision making promotes the need for a creative decision making process to identify and construct potential courses of action in response to an identified developing situation. These possibilities are then filtered and reduced to a set of feasible options. The process is gradually refined until alternatives are decided between and a specific (best or least worse) course of action is to be chosen that will be adopted to achieve the identified strategic aim.

As mentioned earlier, two different delivery methods are currently used for training, the table top exercise and the real world simulation exercise. The PANDORA concept is to replicate the benefits of both of these methods by developing a system capable of guaranteeing the realism of the real world simulation and the practicality and affordability of table top exercises.

The PANDORA approach. Goal of the PANDORA project is to build an intelligent training environment able to deploy a spectrum of realistic simulations of crisis scenarios that: (1) reproduce the stressful factors of the real world crisis; (2) personalize the planned stimuli according to the assessed abilities of different trainees and (3) supports the dynamic adaptation of "lesson plans" during the training time-horizon.

The system design has followed a user-centered approach, based on a close cooperation with the training experts who have profoundly influenced the shaping of the system. Specifically, the Cabinet Office Emergency Planning College (EPC) has synthesized their experience, gained from training a wide range of senior decision makers combined with their pre-eminent expertise in emergency planning and crisis management. As the end user representative in the PANDORA consortium, EPC has contributed to identify the main requirements specification of the innovative training environment, and is influencing the design and implementation choices.

A number of general constraints have emerged during a first phase of user requirement analysis:

– *Support cooperative decision making*: it has become clear immediately how important it is to train gold commanders to take key decisions jointly in collaborative working conditions.

– *Training personalization*: the role of personalized teaching has been underscored even within a group decision making context.

– *Mixed-initiative interaction*: The need to have a tool that would empower the trainer to adapt and adjust the training session in real time run became apparent rather than relying upon a video-game type of immersive experience, hence the need to create a mixed-initiative environment in which the trainer is fully integrated in the "lesson loop".

Figure 2 shows the main architectural idea pursued within the project to obtain a system called the PANDORA-BOX whose current complete version was officially demonstrated in March 2011. The system comprises three environments: (a) a Trainer Support Framework allows the trainer to keep control of the training session and dynamically adjust the stimuli based on his/her experience; (b) distributed Trainee

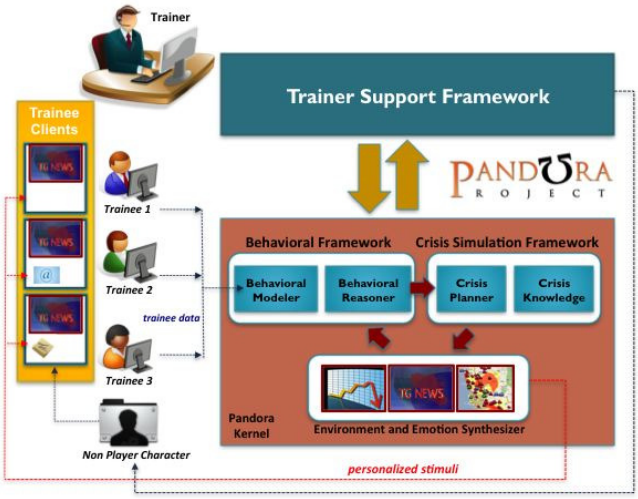


Figure 2: *The* PANDORA-BOX *general architecture*

Clients can access the PANDORA-BOX and receive both collective and individual stimuli during a lesson; (c) a PANDORA kernel which is the main engine that generates the "lesson plan", animates it in an engaging way and adjusts it on a continuous basis to keep pace with both the evolution of the specific group of people under training and their individual performance.

Specifically, a group of trainees, representative of the different agencies that would be involved in the resolution of a crisis (e.g., Civil Protection, Local Authorities, Health, Fire Rescue, Police, Transportation Agencies and so on) have access to the training system through their work station. If some of the representative authorities are not present they would be simulated by the PANDORA system through a Non Player Character (NPC), in which case, features and decisions are synthesized by the trainer via the system.

The various participants in the training session are characterized by different aspects, both in relation to the components closely linked to their role and responsibility, and for the particular "affective states" they may exhibit during the training experience in response to the presented stimuli. Therefore, each trainee, by interacting with the system, feeds personal data to the PANDORA-BOX, which gathers this information to build a *user model* (Behavioral Model shown at Figure 2). Based on this model, the system synthesizes a personalized training path that meets the specific needs and status of each trainee (Behavioral Planner). The output of this process is passed to a second module (the Crisis Planner), which on the basis of the Behavioral Module's indications, as well as the knowledge of the chosen guiding training scenario, synthesizes a sequence of stimuli appropriate for both the group (information shared among all trainees) and the individual trainees (information tailored to induce the "right level of stress" for different individuals).

The plan synthesized by the crisis planner is then given as input to the module called the Environment and Emotion Synthesizer which is responsible for an effective rendering of the training temporal plan. In practice this module adds an additional level of "realism" to the stimuli, by customizing the appropriate presentation mode (e.g., introducing back-

ground noise or other distraction during a phone call report) in order to achieve a high level of realism, stress and pressure. The use of advanced 3D scenario reproduction is also included and assessed in the project.

Overall the PANDORA-BOX supports the loop *trainer* → *training environment* → *trainee*, encouraging the customization and adaptation based on the users feedback as well as the inclusion of training goals and other inputs by the trainer.

We now turn to timeline-based planning technology within the PANDORA-BOX and describe how the planning technology has become the unifying element of the overall system.

The planning problem

The basic goal for the training environment is to create and dynamically adapt content for a four hour continuous training session. The pursued idea is to represent a session's content as a *plan* composed of different "messages" to be sent to trainees which have temporal features and causal relations among them. In PANDORA a lesson master plan is first synthesized starting from an abstract specification given by the Trainer. It is then, animated, expanded and updated during its execution, in response to new information gathered from the trainees and the decisions that they may make. Specifically, the lesson master plan contains time-tagged activities that trigger multimedia events presented presented to the trainees. A key aspect will be the reaction of trainees to lesson stimuli (e.g., the answer to a request to produce a joint decision on a specific critical point). "User reactions" internally represented in the plan trigger different evolutions of the current plan thus supporting dynamic adaptation.

The use of AI planning is quite natural for creating such a master plan. Previous work exists on the use of constraint reasoning for synthesizing multi-media presentations (e.g., (Jourdan, Layaida, and Roisin 1998)), and on the use of planning in story-telling (e.g., (Young 1999)), etc. The main "technological idea" we have pursued in PANDORA is to use timeline-based technology to represent and organize in time heterogeneous information, a choice that naturally matches some of the manipulations that were specifically required by the master plan representation within the project. In particular two aspects offered an interesting challenge for timeline based technology: (a) the idea of doing planning, execution, re-planning in a continuous cycle; (b) the possibility for modeling a completely different type of information with respect to the "usual" applicative domains in which timeline-based planning has been used (e.g., (Muscettola 1994; Jonsson et al. 2000; Cesta et al. 2011)).

A timeline-based problem representation. Figure 3 exemplifies the basic modeling features and introduces some terminology for the PANDORA domain modeling. The main data structure is the timeline which, in generic terms, is a function of time over a finite domain. For the purpose of this description we call "events" the values for a timeline. Events are represented with a predicate holding over a time interval and characterized by start and end time [2].

Events can be linked to each other through *relations* in order to reduce allowed values for their constituting parameters and thus decreasing allowed system behaviors. In general, relations can represent any logical combination of linear constraints among event parameters. According to the number of involved events, relations can be divided into unary, binary, and n-ary. For example, unary relations are used in the PANDORA-BOX to fix initial scenario event parameters by placing them in time. Given an event e, an example of unary relation can be $start\text{-}at\,(e, 15, 20)$ forcing the starting time of the event e to be constrained inside simulation time interval $[15, 20]$. Given two events e_0 and e_1, an example of binary relation can be $after\,(e_0, e_1, 100, 120)$, forcing the starting time of event e_1 constrained to be a minimum 100 and maximum 120 time units after ending the time of event e_0.

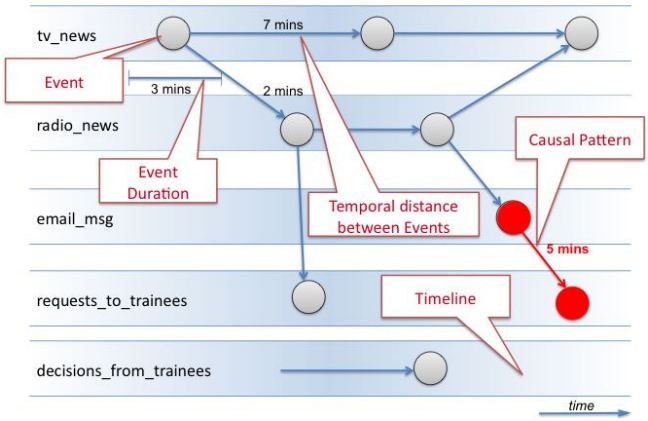


Figure 3: *The timeline-based plan data structure*

An "Event Network" is a hyper-graph having events as nodes and relations as hyper-edges. Through the concept of an Event Network, the whole timeline-based planning procedure can be reduced to the process of reaching a target Event Network, that meets the desired *goal* conditions, starting from an initial Event Network. In our case, goal conditions are characterized by high level scenario events representing the abstract blueprint for the master plan while the initial Event Network is, trivially, an empty Event Network.

In the example of Figure 3 we see an Event Network distributed over 5 timelines (three representing different media for giving "active" information about a situation (`tv_news`, `radio_news`, `email_msg`), and two more special to purpose, seeking trainee input and gathering such input (`request_to_trainees`, `decisions_from_trainees`) [3].

A further basic ingredient in timeline modeling are the

[2] Events here are equivalent to "tokens" in other timeline-based approaches (Muscettola 1994). The reason for re-naming them is to focus attention on the PANDORA main task, namely the generation of timeline values to be "rendered" as a specific multi-media *event* when presented to trainees. Even if the use of timelines in PANDORA is wider with respect to the pure generation of multi-media events, the name survived to facilitate communication internal to the project.

[3] As usually done in timeline-based planning, the sketchy Event Network shown in the figure is defined on top of a Temporal Constraint Network and included between a start-time and end-time of a temporal horizon.

so-called "Causal Patterns" (see an example in Figure 3). These are a way to express planning domain/causal rules in the current internal representation. Any given Event Network should be consistent with respect to the set of such specified causal patterns [4].

Patterns are defined through a logic implication $reference \rightarrow requirement$ where $reference$ is the event value that demands pattern application while $requirement$ is the "consequence" of the presence of the $reference$ value in the Event Network. Making use of a recursive definition, a requirement can be a target event value, representing a new value on the same or another timeline and a relationship between reference value and target values, a conjunction of requirements or a disjunction of requirements. Being relations, in the most general case, linear constraints, causal patterns allow great expressiveness that allows a PANDORA modeler to represent quite complex behaviors.

A planning domain is generically defined by creating a set of timelines and a set of Domain Causal Patterns. From this basic domain representation, receiving a set of goals a planner generates an event network to be executed.

Opening the PANDORA ... BOX

Having introduced the basic modeling features we now describe the use of reasoners that support PANDORA functionalities. Figure 4 shows the different modules that work around the Timeline-based Plan Representation which is the central data structure as always in this type of planning applications.

Trainer abstract plan. The initial driving role is given to the Trainer. Through his Support Framework the trainer can loads a specific "Scenario", an abstract plan sketch that works as a sequence of "lesson goals" and as a skeleton plan for the ground planner. The scenario is contained in a particular timeline that generates sub-goaling by interacting with the set of domain causal patterns. Scenarios have the double role of enabling the Trainer to reason on a high level of abstraction thus avoiding the details of the planning technology and to continuously influence the event network that actually implements the detailed lesson at ground level. Furthermore the Trainer is endowed with commands to introduce single steps in a scenario hence triggering dynamic plan adaptation. It is worth highlighting how the overall system aims to empower the trainer with a more effective means to train people. Indeed the suggested crisis stimuli as well as the behavioral analysis is offered to the trainer who can influence at any moment the training session in perfect line with a *mixed-initiative* style.

The Planner in Figure 4 works on the ground timeline representation to create the training storyboards, e.g., the set of connected "events" that are communicated to the trainees (e.g., a video news from the crisis setting, a phone call or e-mail from a field manager, and a set of temporal distances among events). Once the planner has achieved a fix-point given the abstract scenario goals from the Trainer and the Domain Causal Patterns, the responsibility is left to the Plan Dispatcher that step-by-step executes the plan by sending events to the Rendering Environment according to their progressive start times. Some of the events are requests for trainees to make decisions (see Figure 3), the result of which are fed back to the timeline representation as additional information for plan adaptation. In fact the Planner is able to reacts to trainees' strategic decisions, triggering consequent events to continue the training session.

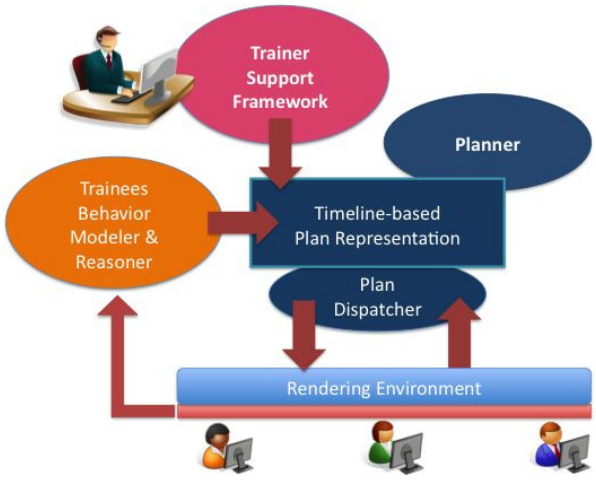


Figure 4: *A blow-up of the* PANDORA-BOX

A further path on the dynamic adaptation of the lesson plan is given by the personalization for each trainee which is fostered by the block named Trainees Behavior Modeler and Reasoner. Through this module psycho-physiological trainee features are modeled and updated during training and internally represented as timelines. Specifically a set of "relevant user variables" has been selected among those that influence human behavior under crisis and used to build a trainee model. The timeline-based approach also supports the dynamic update of the user model during the training session. Based on this model, the Behavioral Reasoner, synthesizes specific result timelines that are used as goals by the general planner, thus introducing a continuous loop of adaptation aimed at tailoring the intensity of stimuli to individual trainees.

Planning a lesson. Starting from scenario goal events and from the set of domain causal patterns, the planning process generates a target Event Network that is consistent with the given goals, ordering events in time through scheduling features and producing proper event consequences. Additionally, as described in previous section, new goals can be added during crisis simulation to represent (a) decisions taken by trainees, (b) inferences made by the behavioral reasoner, (c) new scenario steps added by the Trainer. The PANDORA planner is therefore able to replan in order to make its current Event Network to remain consistent with respect to the new dynamic input and with its consequences, namely, changing the current course of the simulated crisis.

[4]Causal Patters are defined within a domain description language, similar to *compatibilities* (Muscettola 1994) or *synchronizations* (Fratini, Pecora, and Cesta 2008), that allow to specify a pattern of mixed time/causal value relations involving PANDORA events.

In general, target event values are added to current Event Network producing new goals that require pattern application in order for them to be causally justified. It is worth noticing that disjunctions of requirements produce branches on the search tree guaranteeing varieties of presented scenarios. In particular, it may happen that some rule cannot be applied since it imposes too strict constraints resulting in an inconsistent Event Network. In such cases, a backjumping procedure allows to go back to the highest safe decision level. When the planning process succeeds, an Event Network consistent with given goals replaces current planner state. At present we are using a planner which is inspired by (Fratini, Pecora, and Cesta 2008) but is specifically tailored to PANDORA needs.

It is worth noting that because not all courses of action in a crisis can be predicted at scenario design time we have also endowed the Trainer with a service that allows to incrementally modify the ongoing scenario in order to adapt the simulation to unpredicted trainees' decisions. Alternatively, the trainer can manipulate ongoing crisis to bring back execution to a desired behavior having already predicted courses. This kind of scenario modifications are stored in a knowledge base providing capacity to expand and evolve the system training capabilities during its use.

The role of trainees modeling and personalization. The trainee's profile are built by considering relevant variables known to have an influence in decision making under stress (Cortellessa et al. 2011). An initial assessment is made through standardized psychological tests and physiological measurements made off-line, immediately before the training session begins and updated during the training session. We choose to model trainees variables, similarly to the lesson storyboard, hence using timelines, in order to maintain a unique representation system. Therefore, with a little overhead of terminology, we will call *event* any of the values on timelines.

We demonstrate how lessons personalization is planned for through a simple example. Two trainee features that are relevant during training (Cortellessa et al. 2011) are:

– the *background experience*, the crisis leader' past experience in managing crisis situation. A short questionnaire assesses leaders socio-demographic information, their previous experiences with leading public health and safety crises and their level of success in doing it. This variable can be used also for tuning the right level of difficulty during the training exercise. We represent *background experience* through predicates of the form $background\text{-}experience\,(x)$ where x is an integer assuming values 0 for low experience, 1 for medium experience and 2 for high experience;

– the *self efficacy* defined by (Bandura 1986) as the people belief in their capabilities to perform a certain task successfully. It has been shown that this variable has influence on different aspects like the ability to manage stressful situations, performance as well as the probability to receive benefits from training programs. We represent the *self efficacy* through predicates of the form $self\text{-}efficacy\,(x)$ with x being an integer ranging from 0 to 10.

In order to explain how trainees are assigned to profiles during training, let us suppose that a trainee x answers to a self-efficacy question and that, consequently, an event representing its updated level of self-efficacy is added to his (or her) self-efficacy timeline. The causal patterns that is applied by the planner have a structure similar to the following:

$$x.self\text{-}efficacy \rightarrow \begin{cases} pro : x.profile \\ during\,(this, pro, [0, +\infty]\,, [0, +\infty]) \end{cases}$$

This patterns assures that every time we have a self-efficacy update, an event, named pro locally to the rule, is added to $profile$ timeline of trainee x, new self-efficacy value must appear "during" pro (triggering event's starting point is constrained to be $[0, +\infty]$ before pro's starting point while pro's ending point is constrained to be $[0, +\infty]$ before triggering event's ending point). Once the event pro is added to current Event Network the solving procedure is called and requires itself a pattern application.

Let's assume now that the following requirements, representing trainee association to different profiles, are defined inside the Behavioral Modeler:

$$r_0 : (se.value = 0 \wedge be.value = 0 \wedge is.value = 0)$$
$$r_1 : (se.value = 1 \wedge be.value = 0 \wedge is.value = 1)$$
$$\ldots$$

These requirements basically state: if self-efficacy value is equal to 0 and background-experience is equal to 0 than induced-stress' $value$ parameter must be equal to 0; if self-efficacy value is equal to 1 and background-experience is equal to 0 than induced-stress' $value$ parameter must be equal to 1; etc.. Enacting such requirements the association of trainees to profiles can change. Profile information is than passed on to the Crisis Planner that updates values of other timelines associated to trainee x, for example, changing the amount of induced stress for the trainee x, using a pattern like:

$$x.profile \rightarrow \begin{cases} se : (?)\, x.self\text{-}efficacy \\ be : (?)\, x.backgroung\text{-}experience \\ is : x.induced\text{-}stress \\ contains\,(this, se, [0, +\infty]\,, [0, +\infty]) \\ contains\,(this, be, [0, +\infty]\,, [0, +\infty]) \\ equals\,(this, is) \\ r_0 \vee r_1 \vee \ldots \end{cases}$$

where the $(?)$ symbol forces target values se and be to "unify" with an already solved event in order to close the loop and interrupt the pattern application process for the event. Finally, induced stress pattern selects proper events from Crisis Knowledge Base and propose them to the trainee in order to generate an adequate stress level with the aim of maximizing the learning process.

Executing the lesson plan. Another important functionality of the PANDORA system, the more relevant for understanding the use of plans, is represented by the lesson plan execution. Simulation time t is maintained by execution module and increased of execution speed dt at each execution step. Each timeline transition that appears inside

Figure 5: *Screenshots of the current Trainer and Trainee Interfaces*

interval $[t, t + dt]$ is then dispatched to PANDORA rendering modules for creating the best effect for the target trainees. By maintaining information about current simulation time, the executor module is responsible for placing in time events that represent trainees' actions, adding proper relations, thus fostering re-planning features, or plan adaptation, in order to integrate actions' consequences inside current Event Network.

Additionally, the training process requires utilities for temporal navigation through the storyboard allowing execution speed adjustments as well as features for rewind and rerun. When going back in time, two different behaviors are provided by PANDORA-BOX:

– *default roll-back*, intended for debriefing purposes, that simply updates current simulation time t to desired target value keeping untouched actions taken by trainees;

– *heavy roll-back*, intended to revert to a crucial decision point at time t, removing each event representing trainees' choices at time $t' > t$, along with their consequences, in order to allow a different simulation course.

A further feature worth noting is that at the end of a training session the resulting completed plan contains all the information given to the class, as well as well as the trainee decisions to required questions, the simulated consequences of such decision and also the trainee's psycho-physiological state evolution. In general this is an annotated plan that can be used by the trainer during a debriefing phase to explain pros and cons of the trainees behavior during the lesson. The different roll-back functions could contribute to this phase to re-run stretch of the lesson for explanation purposes.

Current status

A first prototype of the complete system has been produced in early December 2010 while a first robust version of the PANDORA-BOX has been officially demoed on March 2011 to the EU project officers during the for mid-term project review. To give the reader an idea of the system at work this section describes first some aspect of the interactive environment that connects Trainer and Trainees and then presents a simple experimental table to show the time needed to the planner to synthesize event networks of different size.

The interactive environment. Figure 5 depicts some of the interaction features that have been implemented in the demonstrator. Specifically, we can distinguish between two types of interaction:

– *trainer-system interaction*, indicated as *Trainer View*, which is related to the functionalities available to the trainer to create a training session, monitor, edit it and interact dynamically with the class;

– *trainee-system interaction*, indicated as *Trainee View*, which is the interface through which the trainee can connect to the PANDORA-BOX, receive stimuli and make decisions about the critical situation.

Trainer View. After creating a *class*, the trainer can load a *Scenario*, and see it in tabular form with a series of important information such as the execution time of each goal event and who is the main recipient of information. It is worth highlighting how this representation reproduces the current way of working of the trainers and has been instrumental in establishing a dialogue with them, before proposing any kind of completely new solutions. Along with the scenario, the interface also contains information about available resources to resolve the crisis and the consequences of trainees' decisions, both represented through resource timelines and dynamically updated during the training. In parallel with the traditional tabular view, the trainer can inspect a more advanced view of the PANDORA module, that is the internal representation of both the Crisis and the Behavioral Framework (*Expert View*). As already said, all type of information within PANDORA is represented as a timeline and continually updated (see different colors for timelines re-

lated to the crisis and the user model in the Expert View). At this point, through the Execute button, the trainer can start the session. A series of additional commands also allows the trainer to dynamically add new stimuli, in perfect line with the mixed initiative interaction style.

Trainee View[5]**.** The Trainee interface contains three main blocks, in addition to a number of features related to communication of each trainee with the rest of the class and the trainer. The main building blocks are the following: *Background Documents*, which represents a set of information delivered off-line to the class in the form of maps, documents, reports, in order to create awareness about the upcoming exercise; *Dynamic information* that represents the information dynamically scheduled and sent to the trainee in the form of videos, maps, decision points etc.; *Main Communication Window*, which is devoted to display stimuli (possibly customized) to individual trainees or to the class.

The interaction environment has been critical in our dialogue with the end users and will be further refined on the one hand to satisfy user requirements on interaction, on the other to make the advanced features more useful for the trainer eventually filling the gap between the internal representation and users' expectation, with the aim of promoting their active involvement in the management of training.

Table 1: Average problem solving times in proportion to initial goal number.

goal #	avg. t (ms)	ev. #	var. #	constr. #
26	36	46	459	42
32	54	64	1459	92
76	296	186	2459	142
101	455	256	3459	192
126	979	326	4459	242
151	1511	396	5459	292
176	1903	466	6459	342
201	2864	536	7459	392
226	3793	606	8459	442
276	6241	746	10459	542

The planning time constants. In order to give an idea of the performance of the timeline-based internal engine we report here an initial scaling test. In particular we have generated a fixed training class of a single trainee plus four NPC players, leading to a total number of 84 timelines, and attempted to load several crisis scenarios of increasing complexity. Table 1 summarizes average scenario loading times showing initial imposed goals, planner solving times expressed in milliseconds, and events number resulting after planning process. Finally, last two columns show the number of involved variables and constraints among them in order to give an idea of underlying problem complexity.

Ongoing work is aimed at finding a smarter way to remove elements from an Event Network, at increasing overall performances through some preprocessing steps and at facilitating scenario editing in order to allow non-technical people to easy modify simulated crisis.

[5]The current Trainee interaction features have been implemented by our colleagues from XLAB.

Conclusions

This paper has described the year one demonstrator of the PANDORA project. Main goal of the paper is to give the reader a comprehensive idea of the use of planning technology in the PANDORA-BOX. We have seen how the representation with timelines is the core component of the crisis simulation, and that a continuous loop of planning, execution, plan adaptation is created to support personalized training with Trainer in the loop.

Acknowledgments. The PANDORA project is supported by EU FP7 under the joint call ICT/Security (GA.225387) and is monitored by REA (Research Executive Agency). Authors are indebted to all the project partners for the stimulating work environment.

References

Bandura, A. 1986. *Social Foundations of Thought and Actions. A Social Cognitive Theory*. Englewood Cliffs, NJ: Prentice Hall.

Beck, U. 1992. *Risk Society: Towards a New Modernity*. London: Sage., translated by m. ritter edition.

Cesta, A.; Cortellessa, G.; Fratini, S.; and Oddi, A. 2011. MrSPOCK: Steps in Developing an End-to-End Space Application. *Computational Intelligence* 27(1):83–102.

Cortellessa, G.; D'Amico, R.; Pagani, M.; Tiberio, L.; De Benedictis, R.; Bernardi, G.; and Cesta, A. 2011. Modeling Users of Crisis Training Environments by Integrating Psychological and Physiological Data. In *IEA/AIE 2011, Part II,*, volume LNAI 6704, 79–88.

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.

Jonsson, A.; Morris, P.; Muscettola, N.; Rajan, K.; and Smith, B. 2000. Planning in Interplanetary Space: Theory and Practice. In *AIPS-00. Proceedings of the Fifth Int. Conf. on Artificial Intelligence Planning and Scheduling*.

Jourdan, M.; Layaida, N.; and Roisin, C. 1998. A Survey on Authoring Techniques for Temporal Scenarios of Multimedia Documents. In Furht, B., ed., *Handbook of Internet and Multimedia Systems and Applications - part 1: Tools and Standards*. CRC Press. 469–490.

Klann, G. 2003. *Crisis Leadership*. Greensboro, NC: Center for Creative Leadership.

Leonard, H. 2004. Leadership in Crisis Situations. In Bums, J.; Goethals, G.; and Sorenson, G., eds., *The encyclopedia of leadership*. Berkshire Publishing Group, Great Barrinton.

Muscettola, N. 1994. HSTS: Integrating Planning and Scheduling. In Zweben, M. and Fox, M.S., ed., *Intelligent Scheduling*. Morgan Kauffmann.

Wilkins, D.; Smith, S. F.; Kramer, L.; Lee, T.; and Rauenbusch, T. 2008. Airlift Mission Monitoring and Dynamic Rescheduling. *Eng. Appl. of AI* 21(2):141–155.

Young, R. M. 1999. Notes on the Use of Plan Structures in the Creation of Interactive Plot. In *Working Notes of the AAAI Fall Symposium on "Narrative Intelligence", Cape Cod, MA*.

Command and Control Training Centers: Computer Generated Forces Meet Classical Planning *

Carmel Domshlak
Industrial Engineering & Management,
Technion, Israel

Ziv Even-Zur and **Yannai Golany**
Elbit Systems, Ltd.
Israel

Erez Karpas and **Yevgeni Nus**
Industrial Engineering & Management,
Technion, Israel

Abstract

We describe SHOGUN, a fully automated system for controlling tactical agents, developed for integration within simulation-based command and control training centers produced by Elbit Systems Ltd. In particular, we focus on describing the action planning module of SHOGUN: while controlling tactical agents in military-style domains involves dealing with uncertainty and partial information in adversarial settings, the planning module of SHOGUN is based on classical, deterministic planning only, and employs a general-purpose classical planner. We describe our embedding of classical planners within the commercial command and control training center, and report on a recent evaluation of SHOGUN in operational scenarios, confronting subject matter experts as trainees.

Introduction

Comprehensive training of forces responsible to react in complex adversarial situations is critical for military high and low intensity conflicts as well as for homeland security scenarios of border control and smart city environments. Such a training should put together teams of trainees at various levels of command, and train them in realistic setups to improve their command and control (C2) capabilities. A vastly dominating portion of C2 training is delegated these days to software simulation systems in which commands of both role-playing trainees and adversary-playing instructors are accomplished by the respective computer generated forces (CGF). Following the paradigm of "train as you fight", the trainees are connected to the virtual battlefield through their operational C2 systems and combat-net radio, coupled by the overall training system to the simulation.

These days, commercial simulations for C2 training already achieve a sufficiently high level of realism in terms of modeling the physical properties of both the environment and forces. The outcome of the training, of course, depends a lot on the effectiveness of the instructors playing the role of the adversary, and this turns out to be an issue. Putting together a team of skilled and coordinated role-players needed for a large-scale simulated exercise requires months of costly preparations, availability of instructors for a long period of time, and a suitable venue. These limitations of relying on human instructors in simulation-based training suggest at least partly replacing them with artificial adversary-players implementing this or another action planning technology. Here we describe SHOGUN, a fully automated system for controlling tactical agents within a commercial military training simulation. SHOGUN has been developed in a joint effort of Elbit Systems Ltd. and the Technion for subsequent integration within the line of large-scale simulation-based training centers produced by Elbit. This system has been recently deployed to Elbit, and successfully passed a detailed performance evaluation.

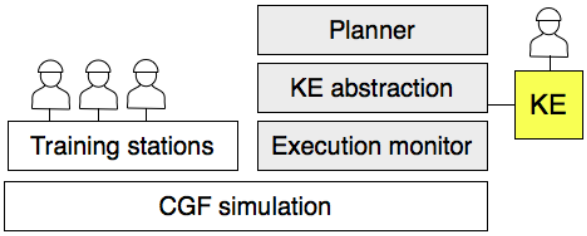
An interesting property of SHOGUN is that the planner it embeds is not just inspired by the artifacts of academic AI research, but actually is such a direct artifact. Moreover, while in general controlling tactical agents in relevant domains involves decision making under uncertainty and partial information in adversarial settings, our experience provides yet more evidence that successful reasoning about real-world systems of active entities does not necessarily have to take explicitly into account all that complexity when choosing between alternative courses of action. While classical planning, capturing single-agent problems with deterministic actions and effectively full knowledge, has been repeatedly criticized for being unrealistic and thus irrelevant to real-world problems, here we demonstrate that this criticism should be taken with lots of caution: The decision making module of SHOGUN is based on classical, PDDL-based planning only, and employs a general-purpose (and thus fully replaceable) classical planner.

In what follows we describe our embedding of classical planners within the commercial C2 training system, as well as the way in which we divide-and-conquer the details of the physical system between the planning and the simulation modules. We then describe the aforementioned evaluation of SHOGUN in operational scenarios, confronting professional military personnel as trainees.

SHOGUN Architecture and Design Decisions

In this section we describe the overall architecture of SHOGUN, focusing on the adopted planning and execution formalism and its support within the system. At high level, SHOGUN comprises a standard architecture of iterative planning, depicted in Figure 1a. It consists of three major mod-

*The work was partly funded by a Magneton Grant. The authors would like to thank Yoav Manor and Gilad Mandel from Elbit Systems for their devoted work on the project.



(a)

input: planning task stub $\Pi = \langle V, A, G \rangle$
local: partial-order plan ρ
$\rho = \langle \cdot \rangle$
forever:
  receive from EM the current state of knowledge σ and plan status $\{\rho_{\text{done}}, \rho_{\text{exe}}, \rho_{\text{next}}\}$
  s = TRANSLATE(σ, Π)
  s' = PROGRESSION(s, ρ_{exe})
  if VERIFY-PLAN(s', ρ_{next}) fails **then**
    ρ = MAKE-PLAN$(\langle V, A, s', G \rangle)$
    send ρ to EM

(b)

Figure 1: High-level (a) structure of the system, and (b) flow of the planner interacting with the KE abstraction mapping.

ules: (i) a *planning module*, (ii) a plan *execution monitor*, and (iii) a real-time high fidelity 3D tactical computer generated forces (CGF) *simulation* in which the actions selected by both trainees and instructors are actually simulated.

Planning, Execution simulation, and Monitoring

The CGF simulation maintains the entire battlefield arena, and supports an arbitrary number of force types (such as tanks, artillery, reconnaissance, etc.), as long as the simulation is provided with their respective physical models. The simulation runs a full 3D virtual environment of the terrain and physical models of sensors (such as line of sight and detection) and actuators (such as ballistics, path planning, and movement). The battlefield comprises two adversarial forces, blue force and red force, each comprising a, possibly heterogeneous, set of acting units. The trainees fully control the blue force troops and interact with the virtual arena via a training station using a high-level language of command. The planning module replacing the instructors fully controls the red force, and communicates with the simulation via effectively the same language of command. The control of the red force is achieved via a planning and execution loop that takes place during the entire training session. The overall loop is described below and the perspective of the planning module on that loop is pseudo-coded in Figure 1b.

- The execution monitor pulls from the CGF simulation all the data σ required to provide the planner with the current state of the red units (their locations, heading, ammunition, etc.), as well as with those parts of the state of the blue units that are considered by the simulation to be observable by the red units. Status of the blue units not detected by any red unit is not provided to the planner. Likewise, the execution monitor pulls from the CGF simulation a status of the currently executed plan ρ of the red force. Since the execution is continuous, at the moment of the query some of the actions of ρ have been already accomplished, some have started and are still executing, and some are yet to be started. Note that "accomplished" can stand here for both "successfully accomplished" and "failed". In any case, both the collected state of knowledge σ and the plan status $\{\rho_{\text{done}}, \rho_{\text{exe}}, \rho_{\text{next}}\}$ are passed to the planning module.
- The CGF simulation is the core of the virtual arena of Elbit's strategic and tactical training centers, designed to communicate with the training stations of human operators. Hence, the information σ about the current state of the (observable) world takes the form of a raw data. This raw data is then translated to a state of the world description s, corresponding to the abstraction of σ in terms of the planning problem operated by the planner. This translation is based on a knowledge engineering layer that is devoted to bridge between the physical view of the simulation and the symbolic view of the planner.
- Given state s and plan status $\{\rho_{\text{done}}, \rho_{\text{exe}}, \rho_{\text{next}}\}$, the planning module estimates whether the current plan ρ of the red force is still valid. In case the goal of reds turns out to be unachievable from s along the still unaccomplished part of ρ, a new plan is generated from the new initial state s, and passed to the execution monitor.

Classical planner: Why and How.

The heart of SHOGUN is its planning system. The first decision we had to make is whether to develop a special-purpose planner, or to adopt a generic, model-oriented planning system. The second, and in a sense, tangential decision we had to make was what details of the problem the planner should take into account and what details it could ignore without sacrificing the quality of the training.

While in principle special-purpose solutions can be more efficient and effective than generic ones, their development requires the enterprise to establish a development team in the respective area of expertise. Along with the fact that the development basically starts "from scratch", that adds numerous risks to the project. Generic planners obviously do not exhibit these risks by the virtue of being generic, having potential to be reused between various verticals. Of course, model-oriented generic planners come with their own risks such as capability of the respective model to capture the desired domain, the computational efficiency of the planner on the domain of interest, etc. However, in contrast to the risks associated with developing a brand new special-purpose system, these risks can be verified in very short time at the beginning of the project using an off-the-shelf planner.

Considering now the choice of the planning formalism, decision making in C2 environments of our interest always involves action non-determinism, partial information, and adversarial settings (Wilkins and Desimone 1992; Tate et al. 2000; Kott et al. 2005). A priori, this sug-

gests that our planning tasks should be specified in terms of much more complicated action models than that of classical planning because the latter assumes deterministic actions, effectively full knowledge, and single-agent setting. Adopting complex planning formalisms, however, comes with a price: the performance of planning for such formalisms currently does not meet the requirements of large-scale C2 training. On the other hand, the performance of classical planners has been dramatically improved over the last two decades, and today these are capable of generating in seconds plans of hundreds of steps in state models of more than 2^{1000} states. In addition, it is of growing understanding that successful reasoning about real-world systems of active entities does not necessarily have to explicitly take into account all the complexity of the reality while choosing between alternative courses of action. This property of many real-world domains has been exploited in the past both in experiments (Yoon, Fern, and Givan 2007; Yoon et al. 2008), as well as in ambitious applications of AI reasoning (Muscettola et al. 1998).

Departing from this matter of business, we have decided to start with a *fully off-the-shelf* satisficing classical planner, adapting it only when really needed and only via external wrappers. Specifically, in the experiments described later on, SHOGUN was using the very popular these days Fast Downward planner (Helmert 2006), using its greedy best first and WA* search engines, and the seminal FF heuristic (Hoffmann and Nebel 2001). The actual planning tasks have been encoded using the PDDL language of the International Planning Competitions (IPC) [1]. PDDL allows representing planning tasks concisely using first-order literals and logical connectives. Fast Downward compiles PDDL input into a ground representation with variables of arbitrary finite range (Helmert 2006). The representation used in Fast Downward is based on the SAS$^+$ action language (Bäckström and Nebel 1995), and extends it with conditional effects and derived predicates. A SAS$^+$ planning task[2] is given by a quadruple $\Pi = \langle V, A, s_0, G\rangle$, where:

- $V = \{v_1, \ldots, v_n\}$ is a set of *state variables*, each associated with a finite domain $dom(v_i)$.
- the *initial state* s_0 is a complete assignment, and the *goal* G is a partial assignment to V.
- $A = \{a_1, \ldots, a_N\}$ is a finite set of *actions*, where each action a is a pair $\langle \mathsf{pre}(a), \mathsf{eff}(a)\rangle$ of partial assignments to V called *preconditions* and *effects*, respectively. Each action $a \in A$ is associated with a non-negative real-valued cost $\mathcal{C}(a)$.

An action a is applicable in a state $s \in dom(V)$ iff $s[v] = \mathsf{pre}(a)[v]$ whenever $\mathsf{pre}(a)[v]$ is specified. Applying a changes the value of v to $\mathsf{eff}(a)[v]$ if $\mathsf{eff}(a)[v]$ is specified. A sequence ρ of actions applicable in the respective states starting from s_0 is a plan for Π if the resulting state s satisfies G.

[1] http://ipc.icaps-conference.org/

[2] For ease of presentation, we present here only the core SAS$^+$ language; for details of the extension we refer the reader to (Helmert 2006).

Formulating planning tasks in SAS$^+$ comes to provide us with a high-level abstraction of the underlying system dynamics: capturing world states at the level of physical simulation would require huge sets of state variables and actions, some state variables are not necessarily observable at any given moment, simulated actions are very much not deterministic, the adversarial blue forces controlled by trainees affect the environment, etc. However, planning at the level of SAS$^+$ has the advantage of fast problem solving, having the potential for compensating for the abstraction coarseness via rapid monitor-and-replan iterations. In what comes next we describe our abstraction mapping of planning tasks from the level of simulation to SAS$^+$.

- *Symbolic abstraction of the physical world.* The function TRANSLATE used by the planning module in Figure 1b to map a physical state σ to a SAS$^+$ state s is implemented via a knowledge engineering sub-module (KE). The latter comes to bridge between the general-purpose planner and the specifics of the simulated domain; as such, it is used twofold. First, KE allows a user to define various layers of information over the map of the training area. These layers describe strategic points, passable areas, ballistically dominating areas, etc., and for most, they can be derived automatically from the digital map used by the simulation. This processing can be performed once per map, and thus completely offline not only to a specific training session, but to the training in general. In addition, the subject matter expert (SME) in charge of the training session can use KE to further enrich this information by specifying, e.g., regions that he prefers not to be used for movements/positions of specific units. Based on the now defined information layers of the map, KE maps status messages received from the execution monitor to proper values of the respective SAS$^+$ variables. The abstraction of the geographic data such as unit locations and headings is archived via a, possibly non-uniform, grid overlaid on the map.
- *Non-determinism of actions.* While basically all actions of the units are simulated to have stochastic effects, the entropy of the underlying probability distributions is usually low, and typically they have single peaks that take most of the probability mass. A natural abstraction of such actions to fully deterministic SAS$^+$ actions simply ignores all but the most likely outcome of each action. SHOGUN uses precisely that simple abstraction, corresponding to a degenerate form of hindsight optimization, an "online anticipatory strategy" for control problems that has previously been successfully applied to problems of online scheduling (Wu, Chong, and Givan 2002) and probabilistic planning (Yoon, Fern, and Givan 2007; Yoon et al. 2008).
- *Partial observability.* Partial observability in the domain of battlefield training stems from the true modeling of reality in which the information that is available to the planner is only what the red force "sees": blue units which are not detected by any red units are not reported to the planner. We use "optimistic sensing" to get rid of this partial observability as follows: when a red unit performs

a sensing action (that is, looks in some direction, trying to find blue units) the expected effects of that action are that no blue forces will be detected. If there are indeed no blue forces - the plan can proceed normally. If there are blue forces there, then the current plan is most likely no longer valid, and therefore re-planning is performed, this time accounting for the "new" blue forces.

- *Optimization objectives.* In most battlefield scenarios, the mission is to achieve some objective, while trying to minimize friendly losses. Since we use single-agent planning, we do not directly account for enemy actions, and specifically, we do not plan for friendly units to be destroyed. Therefore, we do not directly try to minimize friendly losses, but rather try to minimize *risk*. We associate a risk level with each action, by assigning higher costs to riskier actions. For example, maneuvering in a flat area at the base of an enemy-occupied hill is riskier than maneuvering on top of a hill, and is therefore more expensive. Although the planner we use is not an optimal planner, it does try to find a low-cost plan, which directly translates to a low-risk plan.

Domain Formulation

In formulating the domain schema, we made several choices that affect the entire system. First, as stated before, we divide the map into locations, which are arranged on a grid, where each location can hold multiple friendly units, and multiple enemy units. Each grid location is represented by an object in the planning problem, and thus locations are used as parameters for operators and predicates. The translation of world knowledge to a planning state involves mapping units at specific coordinates to the corresponding grid locations.

Second, entities in operational domains often act in line with some standard operating procedures (SOP), and thus, in particular, act in *formations*. In maneuvering, for instance, a formation could be either a single entity moving by itself, 2 entities moving side-by-side, 3 entities moving in a single column, or any other arrangement. Types of formations are defined by the overall set of SOPs, and can be provided by a subject matter expert. We chose to formulate our domain so that all actions are performed by some formations of entities. For example, moving from one location on the grid to a neighboring location is done by using the *Move* action on a formation, which describes the entities to be moved, and their internal arrangement (side-by-side, column, etc.). Two special types of action, *Set-Formation* and *Break-Formation*, allow entities to rearrange themselves in different (possibly larger or smaller) formations. Note that this is similar in spirit to the well-known Logistics planning benchmark from IPC-1998 and IPC-2000, where a formation can be thought of as a truck, and an entity can be thought of as a package loaded into the truck (aka joining formation). This engineering methodology appears to be quite useful in general; for instance a very similar technique has been used by Balla and Fern (2009) in their recent work on action selection for tactical assault, evaluated by the authors on Wargus computer games.

Third, we had to deal with enemy units on the battlefield, and their partial observability. We model enemy presence by creating a state variable for the number of enemy units at each grid location. The possible values for this range are either a number (between 0 and some bound) or *unknown* - a special value indicating that we have no knowledge about enemy presence in that grid location. Thus, the (expected) effect of performing a sensing action on a given grid location is that if the number of enemy units in that location was *unknown*, it becomes 0, and otherwise, there is no effect. This formulation also allows us to ignore the identities of enemy units, which are not part of the knowledge provided to the planner anyway.

Load Balancing and Parallelization

One addition to the standard classical planning setting that we found essential was load balancing between the red units. At high level, the load balancer in SHOGUN pre-assigns each sub-goal to a subset of units, decomposes the overall planning task into several smaller tasks that are planned for independently, and then combines their solutions into a plan for the overall task. This procedure is important for balancing the workload between different role-players, causing forces to act in a more "coordinated matter", and reduces the size of the individual planning tasks solved by the planner.

Similarly to all other system components, the load balancer in SHOGUN is completely domain independent. It starts by assigning to each goal a subset of units that can achieve it as cheaply as possible in the (easy to solve) delete-relaxed version of the planning task. Then, the rest of the units are assigned in proportion to the cost of the relaxed plan for each of the goals, so that goals that are riskier to achieve are assigned more units. Finally, the goals are grouped based on the transitive closure of the forces assigned to them, and each such group of goals is planned for using only the forces assigned to it. In the domain considered here, plan combination is trivial, since no two plans can interfere with each other.

One thing to note is that, although the load balancer might assign several units to a single goal, the planner might still not utilize all of these units (the plan found could involve just a single unit). In order to force SHOGUN to act more realistically, we artificially increase the cost of action repetitions. This puts a heavy bias toward using more than a single unit in each plan, resulting in plans allocating forces to goals in ad hoc proportion to the size of the goal.

Finally, though the quality metric for our plans is risk reduction, and thus we employ a cost-oriented, sequential planning, the actions of different units often can (and if so, should) be applied concurrently. While we do not plan for this second objective directly, we do convert our sequential plans into partial order plans allowing concurrent action execution. If our domain had been formulated in plain STRIPS, then simple Partial Order Causal Link (POCL) backward analysis of the plan would have given us a desired partial order. However, our domain formulation uses conditional effects, and this requires slight extension of the standard POCL analysis. To establish hypothetical causal relations between the actions, we first simulate sequential

execution of the initial plan, determine which conditional effects of which action instances along the plan have been fired, compile the fired conditional effects into now unconditional effects of the respective actions, and then perform the standard POCL backward analysis of the plan. The resulting parallelization is sound and complete, and results in realistic schedule of plans for our multi-unit forces. Overall, the simultaneous acting effect, achieved in SHOGUN via the mixture of load balancing and plan parallelization, achieves the effect of a standard military C2 methodology called "mission-oriented C2", allowing for solving large-scale problems by wisely delegating its sub-parts to different planners.

Usage Practice and Performance Evaluation

One of the critical requirements to "AI driven" systems facing users is that the amount of exposition of the users to the technical details of the system will be as little as possible. In that sense, SHOGUN seems to achieve an extremely high level of transparency. The overall usage of SHOGUN comprises three steps: (i) domain, or action schema, modeling, (ii) training scenario specification, and (iii) the actual training session. The action schema is modeled offline, once for each type of units such as *tank/modelX*, *border-patrol/modelY*, etc. This step does require a subject matter expert to be familiar with the SAS$^+$ action description language, either directly or via a dedicated GUI. However, since it is performed once for all subsequent trainings, it is fully realistic to provide this skill to a small group of SMEs.

Next, possibly at a distant point in time, the properties of a specific training session are specified by an instructor in charge of the training content, e.g., a battalion intelligence officer. This step consists of annotating the map with information relevant to the training (such as markup of areas passable by different types of vehicles, dominating areas, etc.), defining the initial positions of the forces, defining the goal of the training scenario, etc. All these specifications are made through our knowledge engineering tool (already discussed in the previous sections), and requires no knowledge of planning technology whatsoever. Finally, the trainees also need to know neither what SHOGUN is about, nor even the fact that they are trained not by human instructors, but by a fully automated system.

In what follows, we describe the evaluation of SHOGUN that has recently been accomplished by Elbit Systems Ltd. In each training session, an SME controlled the entire blue force, SHOGUN controlled the entire red force, and both operated in a realistic modeling of the situation awareness fog of war. The context of all training sessions was the same 4×4 kilometers area featuring jagged, hilly, terrain, and having several spots dominating large areas and thus having high tactical value. Several regions of the map were marked at the stage of training scenario specification as non-passable, disallowing movement through these regions. Figure 2 depicts a starting position of the forces in CGF simulation. The blue force, controlled by the trainee, comprised an extended armored company of 13 tanks, located initially at a defensive position in the northern part of the map, which

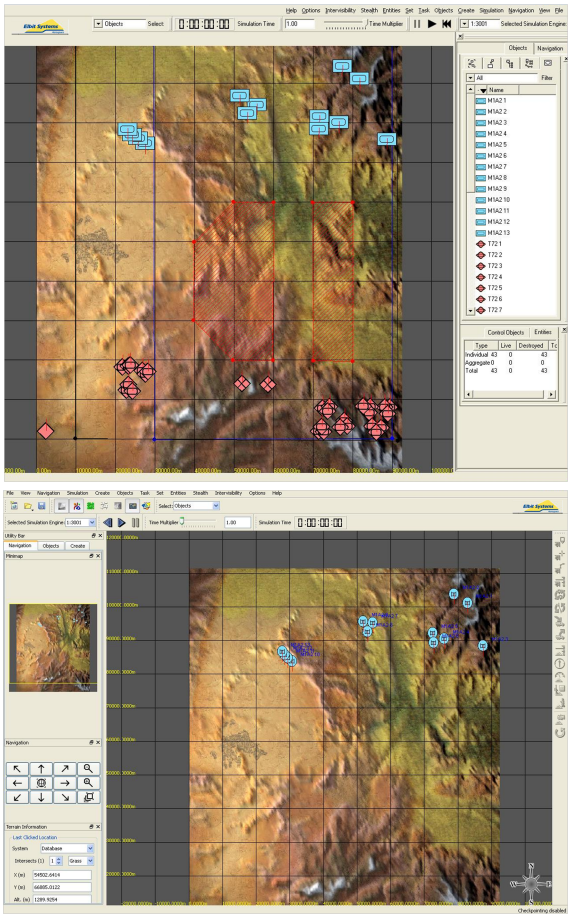


Figure 2: Full (top) and role-player's (bottom) views of a starting position in the Elbit's CGF simulation. Blue/red forces are located in the north/south, respectively. Non-passable areas are highlighted in red.

is the high ground in the training area. The red force, controlled by SHOGUN, comprised an armored battalion of 27 tanks in 9 platoons, supported by 2 reconnaissance units and limited artillery. The mission of the trainee was to defend the blue force's starting region, while SHOGUN's mission was to neutralize all blue units in a predefined region of the map, either by destroying them or causing them to retreat. This magnitude of forces is typical for a mission to neutralize an area of this size, and according to the military best practices, the defending blue force a priori has an advantage in this scenario.

The evaluation consisted of ten training sessions with varying starting positions of the red force, and were done (as detailed in Table 1) by three SMEs: two company commanders in reserve duty (trainees 1 and 2), and one battalion XO in reserve duty (trainee 3). As Table 1 shows, SHOGUN won in 6 out of 10 training sessions, winning all the first training sessions by the individual SMEs, and sometimes losing later on to the same SME, after action review and elaboration of lessons learnt for employing defense battle techniques. From the loss/win pattern followed by the evaluation of the SMEs, SHOGUN provides a realistic adversary

1/R
1. Blue organizes for defense.
2. Red sends first attack with a single company and suffers heavy losses (2 platoons are destroyed, 1 falls back).
3. Red sends full scale attack in 3 different areas, and breaches the blue defense in the east.
4. Red destroys the rest of the blue entities in the area.

1/R
1. Red sends 2 platoons to flank from the east, and one platoon to the attack zone in the west.
2. The red platoon destroys the 4 blue tanks on the hill controling the west zone.
3. The red force takes advantage of the situation and sends a massive attack through the opened west zone (in parallel to units which move through the east zone)
4. Blue tries to organize his defense in order to cover the the west zone, but the red attack progresses too fast and all reinforcement blue forces are destroyed.
5. The remaining blue forces try to avoid further contact but eventually are caught and destroyed by the red force.

1/B
1. Red force sends a preemptive attack. Blue force puts two lines of defense.
2. Red force does not concentrate the attack, and blue force takes advantage of that by destroys the red platoons one by one.
3. The balance of power shifts towards blue force, and the attack fails.

1/B
1. Blue organize in defensive positions which have absolute control on anticipated red attack: moves west passage force to higher ground, and 2 tanks to high control road.
2. Red send 3 platoons to the center and east passages, and 2 platoons to the west passage.
3. Long range sight of the red platoons enables blue forces a safe defense. Red suffer heavy losses in the initial attack, and the remaining force is outmatched in position and fire power.

1/R
1. Red organizes for attack, sends 2 companies with a platoon
2. Red artillery destroys a blue tank controlling the center, yet no reinforcement is sent there by the blue force.
3. Red takes advantage of this: sends a company to the center zone and 2 platoons to the east.
4. Blue is able to hold center attack with eastern defense, shifting its attention on the center.
5. 2 red platoons in the east reach the blue tank and destroy it. This enables the center force to breach in the center area into the blue zone.
6. The blue force defense is breached, eventually loses all troops.

1/B
1. Blue organizes in defense.
2. Red force sends 4 platoons: 2 in the east, 1 in the center, 1 in the west.
3. Blue force moves to higher positions and destroys 2 out of the 4 platoons.
4. 1 of the 2 red platoons left is able to destroy the blue defense in the east zone.
5. Blue tries to defend in the east, but suffers more losses during the defense reinforcement.
6. 1 red tank breaches the blue area in the west, stops and waits for reinforcement.
7. Red organizes for an additional massive attack in the east? but this gives the blue force time to establish there a solid defense.
8. The new positions of the blue force create 2 zones of its full dominance in the west and center, which the red force falls into.

2/R
1. Red sends 4 platoons to the east and center passage.
2. Blue relocates the 4 tanks controlling the west zone.
3. The 4 blue tanks cause damage to the red progress in the east, but are spotted and destroyed.
4. Red artillery weakens the blue defense in the center zone, while the red force destroys the blue defense in the east passage.
5. Blue force tries to reorganize the defense.
6. Red sends a full scale attack.
7. Blue force sends 3 tanks through the eastern passage to counter attack the red forces, but they are neutralized with only minor losses to reds.
8. The single remaining blue tank is surrounded.

2/R
1. Red sends 4 platoons: 3 from the east and 1 through the center passage.
2. Blue relocates the 4 tanks controlling the west zone in a safe passage, trying to occupy safer positions ("learning from experience").
3. The red platoon at the center passage is destroyed by the blue defense.
4. Red starts moving platoons in the west zone.
5. Red company at the east passage destroys the blue defense there, and falls back.
6. One of the red platoons at the west zone encounters the 4 blue tanks, and destroys them with only minor losses to itself. The west area is now totally opened.
7. Red platoon advances in the eastern passage and destroys blue second line of defense.
8. Red starts a full scale attack with artillery support on all 3 passages.
9. Blue destroys a platoon in the east, but the blue area is totally overrun by the red force on all 3 fronts, and eventually the blue force is all destroyed.

2/B
1. Blue organizes differently for defense ("experience from last two loses"): west area tanks move around north hill in a safe route and occupy a high position which controls a huge portion of the terrain. A platoon moves towards a control route in the north part of the terrain.
2. Red sends 2 platoons to the east passage and 1 platoon to the center passage.
3. Blue high position destroys red forces moving in east and center passage in mid way.
4. Red moves to full scale attack including a company in the west.
5. Blue dominant positions disallow the red forces to progress into the blue territory.
6. More than half of the red battalion is destroyed with no losses to the blue force.
7. The entire red company in the west area is destroyed.
8. Red starts artillery fire, and dstroyes the first blue tank.
9. The remaining 5 red tanks fall back and wander around the southeast part of the terrain.
10. Blue moves into offense and starts to progress towards the red forces.
11. Red force artillery destroys one more blue tank.
12. The blue force intercepts, destroys the remaining red tanks and wins.

3/R
1. Blue sends 2 tanks for patrol over high area road which controls most of the terrain.
2. Red sends 3 platoons: 1 in the center, 2 in the east.
3. Red center platoon is destroyed by the 2 tanks patrolling the higher route, but east platoons destroy blue force defense.
4. 2 blue tanks in the high route destroy another red platoon.
5. Red sends a full scale attack accompanied with artillery (2 platoons in the center and 3 platoons in the east).
6. Blue centers his forces in the north west corner of the terrain.
7. The blue force succeeds to hold off 3 red platoons from the east zone while suffering minor losses.
8. Red breaches the blue area in the center zone and destroys all blue troops not located in the northwest corner.
9. Red organizes his troops for final assault on the remaining blue forces and eventually destroys the last blue tank in the area.

Table 1: Summary of system evaluation training sessions. Each table entry marked with X/Y corresponds to a training session by trainee X, ended with the win of Y force. Per trainee, the sessions are listed chronologically. Double line separates indicates a change of either the initial conditions or the trainee.

in a military high intensity conflict. The gradual improvement of the trainees is inline with the objectives of training, and this improvement was indicated not only by the binary outcome of the training sessions, but also by the quality of actions the trainees learned to select from session to session.

References

Bäckström, C., and Nebel, B. 1995. Complexity results for SAS$^+$ planning. *Comp. Intell.* 11(4):625–655.

Balla, R., and Fern, A. 2009. UCT for tactical assault planning in real-time strategy games. In *IJCAI*.

Helmert, M. 2006. The Fast Downward planning system. *JAIR* 26:191–246.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *JAIR* 14:253–302.

Kott, A.; Budd, R.; Ground, L.; Rebbapragada, L.; and Langston, J. 2005. Building a tool for battle planning: Challenges, tradeoffs, and experimental findings. *Applied Intelligence* 23(3):165–189.

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

Tate, A.; Levine, J.; Jarvis, P.; and Dalton, J. 2000. Using AI planning technology for army small unit operations. In *AIPS*.

Wilkins, D., and Desimone, R. V. 1992. Applying an AI planner to military operations planning. In *Intelligent Scheduling*, 685–709. Morgan Kaufmann.

Wu, G.; Chong, E. K. P.; and Givan, R. 2002. Burst-level congestion control using hindsight optimization. *IEEE Transactions on Automatic Control* 47(6):979–991.

Yoon, S. W.; Fern, A.; Givan, R.; and Kambhampati, S. 2008. Probabilistic planning via determinization in hindsight. In *AAAI*, 1010–1016.

Yoon, S. W.; Fern, A.; and Givan, R. 2007. FF-Replan: A baseline for probabilistic planning. In *ICAPS*, 352–359.

Planning and replanning for a constellation of agile Earth observation satellites

Romain Grasset-Bourdel[⋆†] and **Gérard Verfaillie**[⋆]
[⋆] Onera - The French Aerospace Lab, Toulouse, France
{Romain.Grasset, Gerard.Verfaillie}@onera.fr

Antoine Flipo[†]
[†] CNES, Toulouse, France
Antoine.Flipo@cnes.fr

Abstract

In this paper, we present the problem of planning offline on the ground all the activities of a constellation of next-generation agile Earth-observing satellites and the specific algorithm that was developed to solve it. Then, we present the replanning problem that arises when urgent observation requests are received during plan execution. We show how the planning algorithm can be used in this replanning setting, with some modifications that limit computing time and favour plan stability and optimality.

Introduction

The context of the work we present in this paper is the European defence MUSIS project (Multinational Space-based Imaging System for Surveillance, Reconnaissance, and Observation) and more precisely the management of the MUSIS agile satellites that are equipped with high-resolution optical observation instruments.

As usual, such satellites are managed from the ground by a mission planning system which receives user observation requests, builds regularly satellite activity plans over a limited horizon ahead (typically one day), and receives plan execution reports. These plans must meet all the physical constraints and satisfy as well as possible the user requests.

However, such a management system is not very reactive. Any observation request, arriving at any time during the day, must wait for the next day to be taken into account. This led project managers to consider a more reactive management system that would take full advantage of the presence of several ground control stations and of the numerous associated satellite visibility windows that allow updated activity plans to be uploaded.

In such a setting, replanning may be called before any satellite visibility window. Replanning problem data is, on the one hand, a current activity plan involving hundreds of observations and, on the other hand, some urgent observation requests (at most some tens). The goal is to build quickly (efficiency) a new plan over the rest of the day that is of an as high as possible quality (optimality) and is as close as possible to the previous one (stability).

In this paper, we present the physical system we have to manage, the physical constraints we must meet, the user requests we must satisfy as well as possible, and the organization of the management system we assume. Then, we describe the chronological forward search algorithm we developed to solve the planning problem. After that, we describe the replanning problem and how the planning algorithm can be adapted to a replanning setting. Experimental results on real-size scenarios show the right behavior of the chosen approach.

Satellite constellation

The constellation we consider is made up of two identical satellites[1] moving on the same orbit (circular, low altitude, quasi-polar, and heliosynchronous) with a phase shift of 180 degrees between the two satellites (see Figure 1).

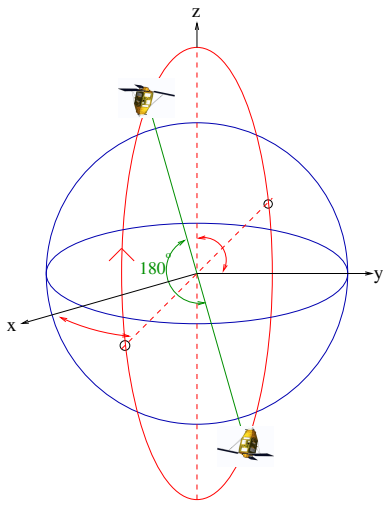


Figure 1: Two-satellite constellation.

Each satellite (see Figure 2) is equipped with thrusters which allow orbital manoeuvres to be performed in case of a too important drift with regard to the reference orbit and with gyroscopic actuators which allow very quick attitude movements (agility) useful to perform observations and transitions between observations.

[1] The planning algorithm we propose is able to manage any number of satellites, possibly not identical: not the same parameter values.

Figure 2: Artist view of a satellite.

A telescope, with two focal planes, allows observations to be performed in the visible and infra-red spectra, with two images (visible and infra-red) within day periods (on the ground) and only one image (infra-red) within night periods. A mass memory allows observation data to be recorded and a high-rate large-aperture antenna allows it to be downloaded towards ground reception stations. Solar panels allow batteries to be recharged when the satellite is not in eclipse. For the sake of agility, all these equipments are body-mounted on the satellite.

Physical constraints

The physical constraints that must be met can be classified into six classes : attitude trajectory, observation, download, memory, instruments, and energy.

Attitude trajectory Thanks to gyroscopic actuators, the satellite is permanently moving around its gravity centre along the three axes (roll, pitch, and yaw). These attitude movements allow observations of areas on the ground to be performed by scanning them. They also allow transitions from the end of an observation to the beginning of the following to be performed relatively quickly. These movements are limited in terms of angular speed and acceleration, resulting in minimum times for moving from an attitude to another. However, the attitude that is required to observe a given area on the ground depends on the orbital position of the satellite and thus on the time at which the observation is performed. The result is a minimum time between the end of an observation and the beginning of the following that depends on the time at which the first ends (see Figure 3 for a schematic 2D illustration). Moreover, computing this minimum time requires solving a complex continuous optimization problem (see (Beaumet, Verfaillie, and Charmeau 2007)). For solving it efficiently inside planning algorithms, dedicated approximate algorithms were developed at ONERA, assuming three-phase movements (constant acceleration, constant speed, and constant deceleration) performed concurrently along each axis (roll, pitch, and yaw).

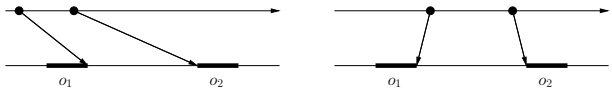


Figure 3: How the angular distance and thus the minimum transition time between observations depends on the time at which the first ends.

Observation Due to maximum observation angles, the observation of a given area on the ground must be performed within one of its visibility windows. Its duration is fixed, because it only depends on the required scanning speed on the ground. The satellite attitude trajectory to be followed during observation depends on the time at which it starts.

Download The same way, due to maximum communication angles, a data download must be performed within one of the visibility windows of one of the ground reception stations. However, this does not suffice because the satellite attitude must be compatible with download (the target station must remain within the satellite antenna communication cone). The result is a set of effective communication windows that depends on the satellite attitude trajectory. Observation and download can be performed concurrently. Two images (visible and infra-red) resulting from the same observation (within a day period) must be downloaded towards the same station and during the same station overflight.

Memory The amount of memory available on board for observation data recording must be never exceeded.

Instruments Concerning the three instruments (high-rate antenna, visible and infra-red focal planes) a minimum preheating time must be met before use, as well as a maximum total ON time and a maximum number of ON/OFF cycles over the planning horizon, for the sake of reliability. Temperature of the infra-red focal plane is automatically regulated by a cryothermic system, but temperatures of both the visible focal plane and the antenna must remain below a given level. Moreover, it must be checked that, at every point on the satellite attitude trajectory, the focal planes are not dazzled and thus not damaged by the sunlight (minimum angle between the satellite axis and the Sun direction).

Energy On-board energy cannot exceed a maximum level due to battery limitations. For the sake of safety, it must remain above a given level, particularly when the satellite is in eclipse and solar panels produce nothing. When the satellite is not in eclipse, the production of energy via the solar panels depends on the satellite attitude trajectory. On the other hand, energy consumption depends on instrument activations.

User requests

With each user request, are associated a polygon which is split into strips, a priority level, a weight, and a deadline.

Typically, three priority levels are available, from 3 (the highest) to 1 (the lowest). It is assumed that any request of priority p is preferred to any set of requests of priority strictly less than p. Weights allow to express preferences between requests of the same priority level and are assumed to be additive. In general, user requests exceed the constellation capacity and choices must be made using request priorities and weights.

It is assumed that any strip can be observed using only one strip overflight. With each strip, are associated a geographical definition, observation durations (day or night), image sizes (visible, day or night infra-red), a maximum observation angle, and a set of triples ⟨satellite, visibility window, weather forecast⟩.

Management system

User requests may arrive at any time and, each day, at a given time, a plan is built for the next day from all the requests that are not out of date and not fully satisfied yet. This plan is built on the ground and then uploaded to the satellites for execution. Typically, up to ten minutes of computing are available for planning. After plan execution, observation data that has been downloaded to the ground is analyzed, taking into account the actual cloud cover, and satisfied requests are removed.

In addition to these normal user requests, urgent ones may arrive at any time too. The latter must be taken into account as soon as possible. To do that, before any visibility window between a ground control station and a constellation satellite, an updated plan is built for the rest of the day from all the requests, either normal or urgent. Replanning is guided by two objectives: on the one hand, to produce a new plan of highest quality, as in planning, and, on the other hand, to maintain in the new plan the highest number of observations present in the previous one, because a plan is a kind of commitment facing users. In order to be able, to take into account urgent requests until the last minutes, we consider that half of the computing time available for planning is available for replanning, that is up to five minutes.

Differently from other studies that considered on-board planning and replanning (Chien et al. 2004; Beaumet, Verfaillie, and Charmeau 2011), planning and replanning are here performed on the ground. This choice is justified by the fact that the information used by planning and replanning (normal and urgent user requests) comes from the ground and not from board. In such a setting, there would be no advantage to plan on board. Limited computing resources on board would even make it disadvantageous.

Planning problem modeling

The planning problem can be modeled using for each satellite the following state variables:

- the current time and thus the orbital position;
- the attitude position and speed along the three axes;
- the available memory and energy;
- for each instrument, its status (ON or OFF), the remaining ON time, and the remaining number of ON/OFF cycles;
- for the antenna and the visible focal plane, its temperature.

Six types of action are available for each satellite:

1. orbital manoeuvres which are mandatory and characterized by starting and ending times and attitudes and by an energy production (function of the attitude trajectory during the manoeuvre);
2. observations which are characterized by a strip, a visibility window, and a starting time;
3. data downloads which are characterized by an image, a reception station, a communication window, and a starting time;
4. heliocentric pointings (solar panels directed towards the Sun in order to recharge batteries as fast as possible) which are characterized by starting and ending times;
5. geocentric pointings (satellite axis directed towards the Earth centre; default action when there is nothing else to do) which are characterized by starting and ending times too;
6. instrument switchings which are characterized by an instrument and a time.

It must be observed that actions of all the types, but the third and sixth (data downloads and instrument switchings), constrain the satellite attitude and are thus mutually exclusive. They must be performed in sequence. Only data downloads and instrument switchings can be performed in parallel, at any time for instrument switchings, but only within effective communication windows for data downloads. As a consequence, a plan has the form of a sequence of actions of any type, except the third and sixth, with attitude movements between consecutive actions and with data downloads and instrument switchings in parallel.

Any plan must satisfy all the constraints described above in Section *Physical constraints*.

We define the criterion to be optimized as a vector of utilities v_p, one for each priority level p. Two vectors resulting from two plans are lexicographically compared. For each priority level p, let R_p be the set of requests of priority p. For each request r, let w_r be the utility associated with r defined as the weight of r weighted by four factors whose value is between 0 and 1 and which represent (1) the percentage of realization (observation and data download), (2) the mean percentage of cloud cover, (3) the mean observation angle, and (4) the mean data delivering delay, over all the strips of the polygon associated with r. At each priority level p, we assume that utility is additive: $v_p = \sum_{r \in R_p} w_r$. The result is a global hierarchical (lexicographic) criterion and a local additive criterion at each priority level.

Planning algorithm

To solve this planning problem, we developed a specific chronological forward search algorithm with dedicated decision heuristics, constraint checking, limited lookahead, and backtrack in case of constraint violation, which guarantees the production of a plan that may be not optimal, but is really executable by the satellites.

Decreasing priorities First, the algorithm we developed works by decreasing priority levels from 3 (the highest) to 1 (the lowest). At each priority level p, the starting point is the plan Pl produced at the previous level $p+1$, which includes orbital manoeuvres, observations (of priority $p+1$ or more), pointings (geo or heliocentric), data downloads, and instrument switchings. However, what is kept from Pl is only the sequence Seq of orbital manoeuvres and observations present in Pl, without their starting times. Other actions present in Pl, such as pointings, data downloads, or instrument switchings are disregarded. At level p, observations of priority p will be inserted into Seq by moving starting times when necessary. Other actions, such as pointings, data downloads, or instrument switchings will be added to build a consistent plan. At priority level 3, the starting point is the set of orbital manoeuvres which are imposed on the mission planning system by the satellite control system and can be classed as observations of priority 4.

Such an approach is justified by the fact that any request of priority strictly greater than p is preferred to any set of requests of priority p. This leads us to consider the sequence of observations present in the plan produced at level $p+1$ as being mandatory when building a plan at level p.

A forward chronological algorithm At each priority level p, the algorithm builds a plan in a forward chronological way, from the beginning Ts of the planning horizon to the end Te. With any step of the algorithm, are associated the current time t, the next observation o of priority $p+1$ or more to be included in the plan because it belongs to Seq, and the set Os of observations of priority p that can be scheduled after t and before o. At the first step, $t = Ts$ and o is the first observation in Seq. The algorithm chooses an observation o' in Os as the next observation to be included in the plan and a starting time t' for o'. If $Os = \emptyset$, then $o' = o$ (observation o is chosen and then a starting time for it). At the next step of the algorithm, t is replaced by the ending time t'' of o' and, if $o' = o$, then o is replaced by the observation that follows it in Seq (empty when o is the last observation in Seq). Figure 4 illustrates two successive steps of the algorithm. The algorithm stops when o and Os are both empty (no other observation to be included in the plan).

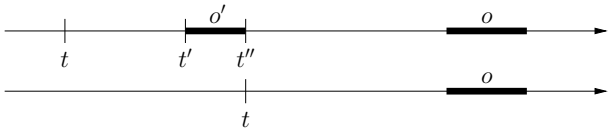


Figure 4: Two successive steps of the forward chronological algorithm.

Decision levels This is the first decision level (1) of the algorithm (choice of the next observation to be included). Once this choice is made, the algorithm makes other choices over the temporal horizon from t to t'' at other decision levels: (2) possible insertion of geo or heliocentric pointings, (3) possible data downloads, and (4) instrument activations.

At the second decision level, geo or heliocentric pointings are inserted between t and t' when possible. Once insertions are decided, the satellite attitude trajectory is completely fixed from t to t''. Hence, the production of energy and the effective communication windows can be computed and the absence of focal plane dazzle can be checked by simulating trajectories.

At the third decision level, data downloads are inserted within the effective communication windows from t to t'' and memory constraints can be checked. This means that observations (first decision level) have priority over downloads (third level). This choice is justified by mission and algorithm considerations: on the one hand, observation is the main system bottleneck and, on the other hand, it is necessary to know the effective communication windows and thus observations and pointings before planning downloads.

At the fourth decision level, instrument activations are inserted in order to satisfy the requirements in terms of observation (visible and infra-red focal planes) and download (high-rate antenna). Energy and instrument constraints can be checked.

Figure 5 shows an example of decisions at the four levels: at level 1, observation o', starting at t', is chosen; at level 2, a geocentric pointing followed by a heliocentric one are inserted before t'; at level 3, data downloads d_1 and d_2, followed by d_3 and d_4, are inserted between t and t''; at level 4, the following decisions are made concerning instrument activations: at time t, the visible focal plane was OFF and it is decided to switch it ON only before o'; on the contrary, the infra-red focal plane was ON and it is decided to maintain it ON between t and t''; at time t, the antenna was OFF, and it is decided to switch it ON before downloading d_1 and to maintain it ON until the end of d_4's download.

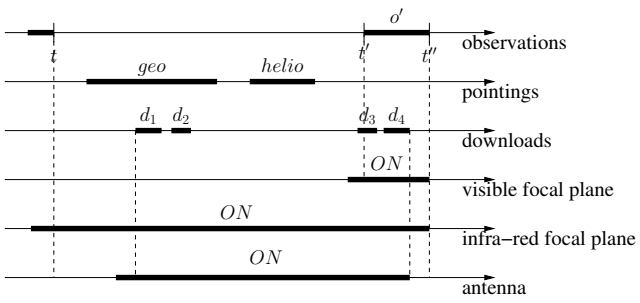


Figure 5: Example of decisions at the four levels: (1) observations, (2) pointings, (3) downloads, and (4) instruments.

Once decisions are made at the four levels, a consistent plan is available from t to t'', extending the plan that already exists from Ts to t, and the planning process can continue from t'', starting from a completely known satellite state.

This incremental process, which built incrementally a complex system trajectory, is the main justification for using a forward chronological search.

For the sake of simplicity, we present the algorithm by assuming only one satellite. However the planning process is in fact interleaved on the two satellites and the next planning

step is the earliest one over the two satellites.

Backtracks At any decision level, in case of constraint violation, other choices are made. If no other choice is available, a hierarchical backtrack at the relevant decision level is triggered (see Figure 6).

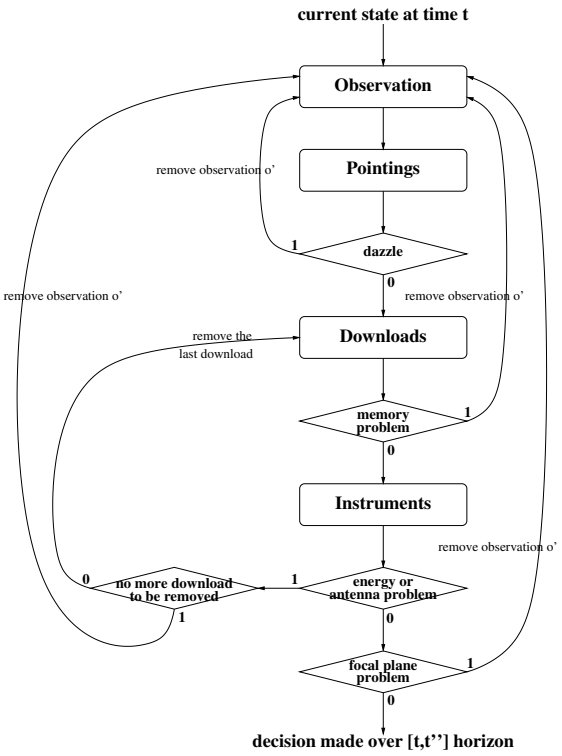


Figure 6: Hierarchical backtracks between decision levels.

At the first level, if $Os = \emptyset$ and thus o is chosen, but insertion of o is impossible, a chronological backtrack is triggered to the previous insertion of an observation of priority p. However, in order to avoid as much as possible such situations, the latest observation ending times are propagated from the end to the beginning of Seq before planning.

Heuristics At all the decision levels, heuristics are necessary to make choices. These heuristics are crucial to the production of good quality plans because, for the sake of efficiency, the algorithm backtracks only in case of constraint violation and never to try and improve on the current plan. It may be important to stress the difference between the global optimization criterion defined in Section *Problem modeling* and the local heuristics described below which only aim at guiding the search towards good quality solutions.

The following heuristics were implemented at the various decision levels:

1. at the first level, as in the knapsack problem, one chooses an observation o' that maximizes the ratio between the increase in the criterion resulting from the insertion of o' (gain) and the time consumed by this insertion ($t'' - t$, considering the earliest starting time for o'; cost); once an observation o' is chosen, one chooses for it a starting time t' that maximizes the increase in the criterion resulting from the insertion of o' at time t' (function of the observation angle; gain) minus the sum of the decreases in the criterion resulting from this insertion (other observations that would become impossible and whose quality would degrade because of too large observation angles; cost);
2. at the second level, an expert rule aims at making easier energy production and data download; it systematically chooses a geocentric pointing when the satellite is in eclipse; when it is not in eclipse, it gives priority to a heliocentric pointing in order to recharge batteries, except in case of visibility of a ground reception station, because a geocentric pointing always allows data download, whereas a heliocentric one may prevent it;
3. at the third level, as in the knapsack problem, one chooses an image of maximum priority that maximizes the ratio between the increase in the criterion resulting from its download (gain) and the duration of this download (cost);
4. at the fourth level, the choice is, for each instrument, at the end of each mandatory activity period, between switching it OFF and maintaining it ON; these choices have an impact on four "resources": energy, temperature, total ON time, and number of ON/OFF cycles; the result is a kind of multi-criteria decision problem; for each resource and for each alternative a, it is possible to compute a ratio between remaining and maximum quantity, if a is chosen; finally, as usual in multi-criteria decision making, one chooses the alternative that maximizes the minimum ratio over the four resources.

The main difference between this algorithm and the one presented in (Beaumet, Verfaillie, and Charmeau 2011) is the use of backtrack mechanisms in case of constraint violation and of more sophisticated choice heuristics. In (Beaumet, Verfaillie, and Charmeau 2011), no backtrack was allowed and heuristics were limited to randomized decision rules.

Replanning problem modeling

When replanning, the main question in terms of modeling is how to manage the possibly contradictory two objectives: (1) the intrinsic quality of the new plan which can be measured using the same criterion as the one used when planning and (2) the plan stability which can be measured by the difference between the new and the previous plan.

The resulting two questions are: (1) how to define the difference between two plans? and (2) how to combine quality and stability objectives? These questions were discussed in planning (Fox et al. 2006; Cushing, Benton, and Kambhampati 2008), in scheduling (Sakkout, Richards, and Wallace 1998), and in constraint satisfaction (Verfaillie and Jussien 2005).

Our experience led us to consider that there is no generic answer to these questions. Answers depend on the problem at hand. In our problem, the quality of a plan is measured by a vector of utilities v_p, one for each priority level p. We maintain this global hierarchical view when replanning. For

each priority level p, let R_p be the set of requests of priority p. For each request r, let w_r be the utility associated with r. We have: $v_p = \sum_{r \in R_p} w_r$. Let $I_p \subseteq R_p$ be the set of requests r of priority p that are negatively impacted by replanning (at least one strip of the polygon associated with r was present in the previous plan, but does not appear in the new one). We define the stability as the sum over the impacted requests of the loss in utility: $s_p = \sum_{r \in I_p} (w'_r - w_r)$ with w'_r (resp. w_r) the previous (resp. new) utility associated with r. s_p is positive or null. The lower s_p, the more stable the plan. Then, we define the criterion to be optimized when replanning as a weighted combination of quality and stability: $vs_p = v_p - \alpha \cdot s_p$, with α a positive parameter to be set by system users according to the importance they attach to stability with regard to intrinsic quality.

To take an example, let us consider two requests A and B of the same priority and of weights w_A and w_B, both reduced to one strip and thus to one observation. Let us assume that A was previously planned, that B is a new urgent request, but that A and B are in conflict (it is impossible to satisfy both). The value of a new plan involving A (no change) is w_A, but the value of a new plan involving B (change) is $w_B - \alpha \cdot w_A$. The second one is preferred only if $w_B - \alpha \cdot w_A > w_A$, that is $w_B > w_A \cdot (1 + \alpha)$.

The data of a replanning problem is very similar to the one of a planning one: same requests, state variables, actions, and constraints. The main difference is in the definition of the criterion to be optimized. Specific data is however:

- the previous plan;
- a set of urgent requests to be taken into account;
- for each constellation satellite s, a replanning horizon from the first time t at which a new plan can be received for execution by s to the end of the current day: before t, the previous plan cannot be modified by replanning.

Replanning algorithm

When replanning, temporal pressure is generally higher than when planning. This pressure takes generally the form of a deadline for plan production. In our problem, this deadline is the beginning of the next visibility window between a ground control station and a constellation satellite.

Local search methods (Aarts and Lenstra 1997) are known to be able to produce quickly good quality solutions on hard combinatorial optimization problems. One of their strengths is that they can be used the same way, with the same local change operations, in a static setting (to solve a problem) and in a dynamic one (to solve a slightly modified problem, using a previously computed solution). This is why they are intensively used in a context of planning and replanning (Zweden et al. 1994; Chien, Knight, and Rabiddeau 2000).

To solve our problem, we did not choose to use local search methods, mainly because of the high potential cost of a local change: adding or removing an action in the middle of a plan requires the complex system trajectory to be computed and checked again from the adding/removing point to the end of the planning horizon.

We chose to develop a chronological forward search algorithm. On this basis, the idea is to use the same algorithm for replanning with slightly different data.

Let P be the set of observations that were considered when planning, let $S \subseteq P$ be the set of observations that were selected by planning (present in the previous plan), and let U be the set of observations associated with urgent requests.

We consider four possible modes of replanning.

1. in the first mode, the set of candidate observations is $S \cup U$; however, we favour stability and consider that all the observations in S are mandatory; for that, it suffices to consider them as observations of priority 4; in this mode, we try and insert the urgent observations in the previous plan without removing anything; however, starting times of observations in S can be moved; the same way, pointing, download, and instrument activation plans can be modified;
2. in the second mode, the set of candidate observations is the same: $S \cup U$; however, at each priority level, we consider that observations in S have priority over observations in U; for that, it suffices to add 0.5 to the priority level of each observation in S; as a result, the number of priority levels is multiplied by 2; in this mode, an observation in U of priority level 3 cannot remove an observation in S of the same priority level, but can remove an observation in S of lower priority level (2 or 1);
3. in the third mode, the set of candidate observations remains the same: $S \cup U$; at each priority level, there is no priority between observations in S and U; all of them are equally considered in terms of priority level;
4. finally, in the fourth mode, the set of candidate observations is $P \cup U$ (all observations); as in the previous mode, at each priority level, there is no priority between observations in P and U; all of them are equally considered in terms of priority level.

Roughly speaking, the search is less and less restrictive from the first to the fourth mode: less and less constraints imposing previously planned observations, more and more observations taken into account. It would be possible to run these modes sequentially or concurrently and to get the best result obtained by the deadline.

Modes 1 and 2 naturally favour stability. Modes 3 and 4 do not so. To favour stability in the latter modes, it is sensible to modify the heuristics used at the first level (choice of the next observation to perform and of its starting date) by multiplying by $(1 + \alpha)$ the weight of a request r if one of its observations o was present in the previous plan ($o \in S$). The idea is to give these requests more importance when making observation choices (see the example in the previous section for an intuitive justification).

Scenarios and experimental results

Planning and replanning algorithms were implemented in a tool, called PLANET for *PLanner for Agile observatioN satElliTes*, which was developed for this mission, on the basis of a previous tool (Beaumet, Verfaillie, and Charmeau

first (easy) instance					
		mode 1	mode 2	mode 3	mode 4
CPU time (s)		130	275	225	232
# obs. removed	prio 3	0	0	1	0
	prio 2	0	0	0	0
	prio 1	0	1	5	4
# urgent obs. added		10	10	8	8
criterion	prio 3	103.7	102.4	101.5	101.7
	prio 2	115.2	114.8	114.8	114.8
	prio 1	96.9	95.4	93.7	94.3
second (medium) instance					
		mode 1	mode 2	mode 3	mode 4
CPU time (s)		133	270	221	231
# obs. removed	prio 3	0	0	1	0
	prio 2	0	4	1	1
	prio 1	0	4	4	7
# urgent obs. added		2	10	9	9
criterion	prio 3	101.9	104.6	103.1	103.3
	prio 2	115.3	111.6	113.6	115.2
	prio 1	96.9	93.4	93.0	92.7
third (hard) instance					
		mode 1	mode 2	mode 3	mode 4
CPU time (s)		129	263	217	225
# obs. removed	prio 3	0	0	3	2
	prio 2	0	2	2	2
	prio 1	0	5	6	13
# urgent obs. added		0	7	8	8
criterion	prio 3	100.8	103.0	102.7	103.2
	prio 2	115.3	112.3	113.2	114.0
	prio 1	97.0	93.4	93.1	93.2

Table 1: Results on the three instances using the four replanning modes

2011). Algorithms were experimented on a real-size realistic instance, built by CNES (French Space Agency) and whose characteristics are the following ones:

- a one-day planning horizon;
- 8 ground reception stations;
- 3 priority levels;
- 1166 observation requests, all of them with polygons limited to one strip and all of them of the same weight (1); among them, 377 of priority 3 (the highest), 419 of priority 2, and 370 of priority 1 (the smallest);
- meteorological forecast built from climatological data.

On this instance, planning takes 236 seconds (about 4 minutes), using a 3Ghz Intel processor with 2.5Go of RAM, running under Linux. In the resulting plan, 906 (78%) observations are performed and downloaded, 16 (1%) are performed, but not downloaded, and 244 (21%) not performed at all. Among the observations of priority 3, 280 (74%) are performed. Results are 367 (88%) for priority 2 and 275 (74%) for priority 1. The fact that relatively more observations of priority 2 are performed than observations of priority 3 can be explained by the fact that, in this instance, observations of priority 3 are more geographically conflicting with each other.

In order to evaluate the four replanning modes, we considered a scenario where 10 urgent requests of priority 3 (the highest) arrive some minutes before uploading the daily plan. Such a scenario is one of the most stressing for replanning because planning must be performed again over the whole one-day planning horizon. For the combination of the quality and stability objectives, we set $\alpha = 0.5$. Following such a scenario, we built three replanning instances of increasing difficulty:

1. in the first one, strips associated with urgent requests are randomly generated on continents; the probability that these strips be in overloaded areas is low;
2. in the second one, strips associated with urgent requests are manually generated on areas where many requests of priority 1 or 2 are present, but few of priority 3;
3. in the third one, strips associated with urgent requests are manually generated on areas where many requests of priority 1, 2, or 3 are present.

Table 1 and Figure 7 show the results obtained by replanning in its four modes on these three instances: CPU time, number of observations removed from the previous plan at the three priority levels, number or urgent observations added in the new plan, value of the new plan at the three priority levels, taking into account quality and stability.

On the first (easy) instance, Mode 1 is clearly the most efficient: all the urgent requests can be added without removing anything; moreover this mode is the fastest in terms of CPU time.

On the second (medium) instance, Mode 2 produces the best results in terms of criterion value: all the urgent requests are added; no request of priority 3 is removed (it is anyway forbidden in Mode 2); only requests of priority 1 and 2 are removed (fewer of priority 2 than of priority 1); however, this mode is the most costly in terms of CPU time.

On the third (hard) instance, things are more complex. No urgent request can be added using Mode 1. 7 urgent requests can be added using Mode 2. One more (8) can be added using Modes 3 or 4. However, fewer requests of priority 3 are removed using Mode 4. Moreover, fewer requests of priority 3 and 2 are removed than of priority 1 using this mode. Mode 4 produces the best results in terms of criterion value, closely followed by Mode 2.

In terms of CPU time, replanning modes 2, 3, and 4 require nearly the same time as planning does. However, this time remains less than the maximum time specified in the mission requirements (5 minutes). Replanning mode 1 requires only half the time used for planning. Moreover, most of the time, replanning will be faster because it will be not performed over a one-day horizon, as in our scenario, but only on the remaining part of the day.

Conclusion

In this paper, we showed that is it possible to use the same chronological forward search algorithm for planning and replanning, only by modifying request priorities and weights,

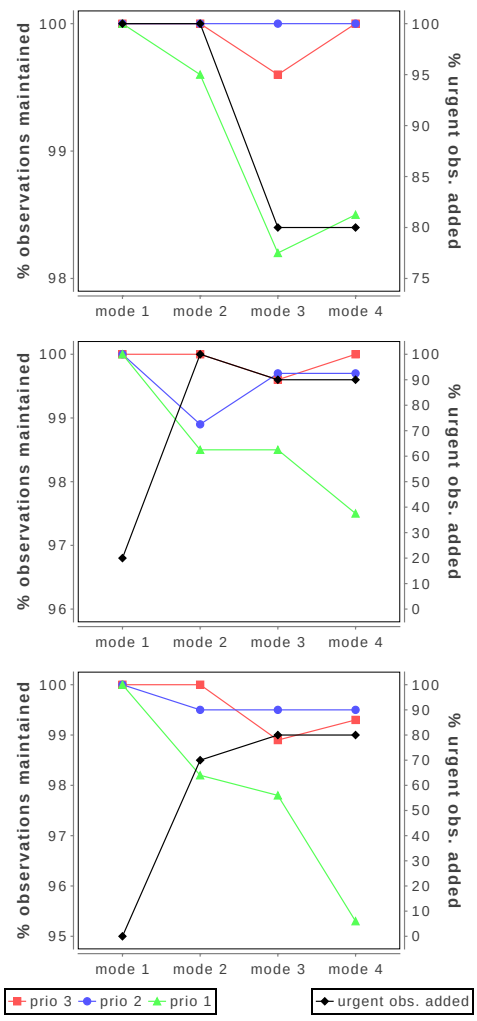


Figure 7: Graphical view of the replanning results; top: first (easy) instance; middle: second (medium) instance; bottom: third (hard) instance

as well as the set of candidate observations. We considered four more or less restrictive replanning modes. First experiments show that their efficiency in terms of quality, stability, and computing time depends on the instance type.

Running these four replanning modes in parallel would be an option. Another option would be to run them in sequence. For that, the order according to which modes are called could be determined for each replanning instance by performing a quick analysis of the setting: urgent requests either geographically spread, or concentrated on already overloaded areas.

References

[Aarts and Lenstra 1997] Aarts, E., and Lenstra, J., eds. 1997. *Local Search in Combinatorial Optimization*. John Wiley & Sons.

[Beaumet, Verfaillie, and Charmeau 2007] Beaumet, G.; Verfaillie, G.; and Charmeau, M. 2007. Estimation of the Minimal Duration of an Attitude Change for an Autonomous Agile Earth-observing Satellite. In *Proc. of the 13th International Conference on Principles and Practice of Constraint Programming (CP-07)*, 3–17.

[Beaumet, Verfaillie, and Charmeau 2011] 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.

[Chien et al. 2004] Chien, S.; Sherwood, R.; Tran, D.; Cichy, B.; Rabideau, G.; Castano, R.; Davies, A.; Lee, R.; Mandl, D.; Frye, S.; Trout, B.; Hengemihle, J.; D'Agostino, J.; Shulman, S.; Ungar, S.; Brakke, T.; Boyer, D.; VanGaasbeck, J.; Greeley, R.; Doggett, T.; Baker, V.; Dohm, J.; and Ip, F. 2004. The EO-1 Autonomous Science Agent. In *Proc. of the 3rd Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-04)*, 420–427.

[Chien, Knight, and Rabiddeau 2000] Chien, S.; Knight, R.; and Rabiddeau, G. 2000. An Empirical Evaluation of the Effectiveness of Local Search for Replanning. In *Proc. of the ECAI-00 Workshop on "Local Search for Planning and Scheduling"*.

[Cushing, Benton, and Kambhampati 2008] Cushing, W.; Benton, J.; and Kambhampati, S. 2008. Replanning as Deliberative Re-selection of Objectives. Technical report, Arizona State University.

[Fox et al. 2006] Fox, M.; Gereveni, A.; Long, D.; and Serina, I. 2006. Plan Stability: Replanning versus Plan Repair. In *Proc. of the 16th International Conference on Automated Planning and Scheduling (ICAPS-06)*.

[Sakkout, Richards, and Wallace 1998] Sakkout, H. E.; Richards, T.; and Wallace, M. 1998. Minimal Perturbation in Dynamic Scheduling. In *Proc. of the 13th European Conference on Artificial Intelligence (ECAI-98)*, 504–508.

[Verfaillie and Jussien 2005] Verfaillie, G., and Jussien, N. 2005. Constraint Solving in Uncertain and Dynamic Environments: A Survey. *Constraints* 10(3):253–281.

[Zweden et al. 1994] Zweden, M.; Daun, B.; Davis, E.; and Deale, M. 1994. Scheduling and Rescheduling with Iterative Repair. In Zweden, M., and Fox, M., eds., *Intelligent Scheduling*. Morgan Kaufmann. 241–256.

Diagnosis As Planning: Two Case Studies

P@trik Haslum and **Alban Grastien**
Australian National University & NICTA
firstname.lastname@anu.edu.au or @nicta.com.au

Abstract

Diagnosis of discrete event systems amounts to finding good explanations, in the form of system trajectories consistent with a given set of partially ordered observations. This problem is closely related to planning, and in fact can be recast as a classical planning problem. We formulate a PDDL encoding of this diagnosis problem, and use it to evaluate planners representing a variety of planning paradigms on two realistic case studies. Results demonstrate that certain planning techniques have the potential to be very useful in diagnosis, but on the whole, current planners are far from a practical means of solving diagnosis problems.

Introduction

In automating the operation of complex technical systems, automated monitoring and diagnosis are as important as automated planning and control.

Traditionally, the diagnosis task is to answer the question: what is wrong? In model-based diagnosis, this means inferring, from a system description and a set of observations, what *mode* the system may be operating in: nominal, one of a number of known fault modes, or an unknown fault mode (e.g., de Kleer and Williams 1989). A typical example is the diagnosis of an electronic circuit, where observations are test inputs paired with measured outputs, and a fault mode is a subset of faulty components. In this setting, the system is static: observations are assumed to all be generated by the same mode, and the system imposes no order on them.

In contrast, diagnosis of dynamical systems may be posed as the question: what has happened? That is, given a system model and a (partially) ordered set of observations, the diagnosis task is to identify possible evolutions of the system over time which would generate the given observations, in an order consistent with that given (Cordier and Thiébaux 1994; McIlraith 1994). Crucially for our case studies, this permits system models that are inherently non-deterministic, even when operating correctly. In most cases, there are many system histories consistent with observations. Thus, there is a notion of preference over histories, and the diagnoser is required to find one, some or all preferred explanations.

There is a close connection between diagnosis of dynamical systems and planning (noted by Cordier & Thiébaux, 1994, and McIlraith, 1994), as the task of generating a (most preferred) system event history to match given observations can be viewed as a plan generation problem. In spite of this, the two fields have developed quite different methods, with much work in diagnosis exploiting off-line analysis of the system model to enhance on-line diagnosis, e.g., by building a system-specific diagnoser (Sampath et al. 1996), identifying if sufficient conditions for using faster methods hold (e.g., Basile et al. 2003), or decentralising the work of diagnosis (e.g., Pencolé and Cordier 2005). The SAT-based diagnoser by Grastien et al. (2007), inspired by the use of SAT for planning and model-checking, is one of a few examples of transfer of techniques between the two fields. To determine if methods that have been successful at solving planning problems will be so also for diagnosis of discrete event systems, we formulate a reduction of the diagnosis problem to planning, i.e., an encoding of the problem in PDDL. It is of course unlikely that such a direct problem translation is the most effective way of applying planning techniques to diagnosis problems, but it is a very useful tool to evaluate a spectrum of planning methods, due to the availability of diverse domain-independent planning systems.

Comparing the effectiveness of planning techniques with existing diagnosis methods raises further challenges. First, there is not one single diagnosis problem. The problem we consider can be characterised as *passive*, *non-exhaustive* diagnosis (of discrete event systems). In contrast, much work concerns exhaustive diagnosis (i.e., finding all preferred explanations consistent with the facts), or active diagnosis (where the diagnoser can take actions to control which observations are made; cf., e.g., Kuhn et al. 2008). These are different problems, and require different solutions. Second, the diagnosis research community does not have the same focus on empirical evaluation over common sets of benchmarks as in planning. Although diagnosis researchers develop domain-independent methods, very few "off the shelf" implementations of domain-independent diagnosis systems are available, and also very few benchmark problem sets.

As a step towards remedying this situation, we formalise two realistic diagnosis problems[1], which we use to evaluate the effectiveness of planners representing several different paradigms. The first case study problem comes from a UAV research project, and is a fault detection problem. The sec-

[1]To the extent that we are permitted, we will make these formalisations available.

ond comes from the domain of electric power transmission, where we aim to do "intelligent alarm processing", by minimising the number of "unexplained" observations. In both cases, the purpose of diagnostic reasoning is to support the situational awareness of a human decision maker. Thus, the diagnoser must reliably provide information that is timely and correct. The meaning of "timely" is not precisely defined, but as a rule of thumb, solutions should be available within a few tens of seconds. Results are mixed: For the first case, a combination of planning techniques present a viable solution. In the second case, the SAT-based diagnosis engine performs better than the best planners, but no approach quite meets the demands of the application.

We are aware only of one similar study. Sohrabi, Baier & McIllraith (2010) proposed an encoding of discrete event diagnosis as a planning problem with temporally extended goals (via the situation calculus), and tested the ability of a heuristic search planner supporting such goals to solve a diagnosis problem. (They also used our encoding to apply the FF planner, which performed better.) The problem they used, however, is an artificial example (introduced by Grastien et al. 2007), and the results they observed do not generalise to our case study problems.

Diagnosis of Discrete Event Systems

The dynamical systems we consider are modelled as discrete state, discrete event systems. This fits naturally with classical planning models. In principle, there is no obstacle to applying the same reduction to diagnosis of timed or hybrid systems – indeed, we construct both a timed and a classical model for our second case study – but the range of planners capable of dealing effectively with such problems is much narrower. In this section, we present a brief review of the discrete event system diagnosis problem. Examples, and details of the reduction to planning, will be described along with our two case studies in the following sections.

A finite discrete event system consists of a finite collection of state variables, each with a finite domain of values, and a finite set of transitions. Each transition has a precondition and a deterministic effect on some subset of the state variables. (Other variables keep their values.) Although each transition is deterministic, the system model as a whole is generally non-deterministic, in the sense that in any state, several transitions may be applicable and there is no deterministic rule that dictates which is taken. For modelling convenience, it is common to divide the system up into components, each an instance of a component type which defines variables and transitions in a schematic way. To model non-trivial behaviour the components must be able to interact, using some communications mechanism such as shared variables, message queues or synchronised transitions.

Some transitions emit one or more *observable events*. There can be several transitions emitting the same event. Given a system model, a set of possible initial system states, a set of observations, O, each labelled with an event $\mathrm{event}(o_i)$, and a partial order $\prec$ on O, a *discrete-event diagnosis* is a sequence of transitions, applicable from some initial state, such that the events emitted by the sequence correpond one-to-one with the set of observations, and the order of these events induced by the sequence of transitions is consistent with the given partial order on the observations. In both our case studies, observations are time stamped, but time stamps are not precise or accurate enough to totally order the observations. Therefore, we assume only a partial order. In any case, this does not complicate the formulation of the diagnosis task as a planning problem.

In the models we consider, transitions are divided into "good" and "bad", and the objective is to minimise the number of bad transitions occurring in the explanation sequence. (The bad transitions are often referred to as *faults*, but this terminology is a bit misleading in the case of alarm processing.) In planning terms this corresponds to the objective of minimising total cost, where only fault transitions have a non-zero cost. In the models we construct, the bad transitions are all equally bad, and therefore all have unit cost, but obviously varied costs could be used to express degrees of badness. For example, if we know the probability of each fault transition occurring, we can assign a cost proportional to its negative logarithm to obtain a most likely explanation. Other preference criteria are also conceivable.

We can also consider more stringent notions of diagnosis: In planning, a *disjunctive action landmark* is defined as a set of actions at least one of which must appear in any valid plan (Helmert and Domshlak 2009). Analogously, we may consider a set of fault transitions at least one of which must occur in any system history consistent with the observations. This is more informative, since it identifies a – albeit disjunctive – set of faults that must have happened, rather than a (minimum cost) set of faults that may have happened. It also has some similarity with the notion of *conflicts* in static system diagnosis (Reiter 1987). A conflict is a set of statements about the system mode that is inconsistent with the system model and the set of observations. Thus, every diagnosis must include the negation of at least one statement from each conflict set. For static systems, there is a one-to-one correpondence between minimal diagnoses and minimal hitting sets over the set of all minimal conflict sets. For discrete event systems, however, such a correspondence does not hold in general.

When modelling a system for diagnosis we have the freedom to trade off the fidelity of the model (and thus its complexity) and its diagnostic power. Abstracting away some aspects of the real system means only that the model may allow more explanations of a given set of observations (some of which may not correspond to possible system behaviours), and thus fewer certain conclusions. We will see this repeatedly in modelling our case study problems.

Intelligent Alarm Processing

The availability of remote sensing and control facilities means that today very large industrial systems, such as power or telecommunications networks, can be overseen and managed by a few operators in a central location. Fault conditions in such large systems frequently give rise to "alarm cascades", where the original fault causes a range of secondary abnormalities, all of which generate multiple alarms, thus quickly overwhelming operators' attention. This problem has been recognised for some time (e.g., Prince, Wol-

lenberg, and Bertagnolli 1989), and there has been a lot of work on the use of AI techniques to aid operators by filtering, prioritising and synthesising alarms. This is known as "intelligent alarm processing".

Two approaches to the problem can be distinguished: One is to view it as pattern recognition, i.e., identifying "situations" or "episodes", that are meaningful to operators, in the alarm stream. This has been developed using methods such as neural networks and chronicles (e.g., Dousson 1996). The other is to view it as root cause analysis, i.e., finding a smaller set of "root cause events" that together would cause the observed alarms. From this perspective, there is no technical difference between alarm processing and diagnosis: the difference lies in the model, and in how it is interpreted. The root cause events, which we seek to minimise, are "unexplained" but not necessarily faults. Model validity means only that for those alarms which are explained, the explanations are correct. It is acceptable, if undesirable, to leave any alarm unexplained. Most work on this approach uses a static system model (e.g., Dijk 1992; Larsson 2009), relying on *ad hoc* methods to divide the flow of alarms into sets for analysis. (An exception is the method described by Guo et al., 2010, which uses a system model based on a temporal constraint network and computes windows for analysis on the basis of the time constraints.) We formulate it instead as discrete event diagnosis, which allows for potentially more powerful explanatory models.

First Case Study: Detecting Missing Events

Input data for our first case study is a set of event logs recorded on the ground during test flights of an autonomous unmanned helicopter (UAV) in 2004 as part of the WITAS project.[2] The logs contain events issued by the UAV control system. The high-level control system consists of dynamically created instances of concurrent software components, called "task procedures" (TPs), which perform mission tasks (e.g., navigating the UAV to a position) by a combination of issuing commands to the low-level (real-time) control system, calling on-board "services" (e.g., the path planner), and invoking other TPs to perform subtasks. Events are sent by TPs, e.g., to signal when a task has been completed, and are tagged with a unique identifier of the sending TP instance. Some events also include additional data. The low-level control system sends events to signal changes in state and in response to commands (e.g., completed or failed).

Due to occasional data overload and loss of telemetry, logs are not complete: some events are missing. The diagnosis task is to detect when such gaps exist. Ideally, we would also like to infer which events are missing, but it is in most cases not possible to do this uniquely.

Formulation as a Planning Problem

TPs are finite state machines augmented with data variables and bits of code executed at transitions. TPs are deterministic, but because we abstract away many details, our models of them are non-deterministic finite automata. For example, the sequence of events generated by the Fly3D TP, which flies the UAV along a given path, depends on the number of waypoints in the path. By leaving the path out of the model, the number of iterations of the TPs main loop becomes non-deterministic. This is an example of the trade-off between model complexity and diagnostic power: if more details were included we could, in some cases, detect missing events that are not detectable with the current model.

Encoding automata with predicates and actions in PDDL is straightforward. Some transitions are synchronised, modelling communication between TPs, but since synchronised transitions involve a fixed number of automata, this can be simply modelled by an action that conjoins the preconditions and effects of the participating transitions. Synchronising an unbounded number of simultaneous transitions requires a more elaborate encoding. (An example of a PDDL encoding of general inter-process communication using message queues is described by Hoffmann et al., 2006, for the Promela domain.) The set of TP identifiers is unbounded, but when creating a planning problem instance for a particular event log, we include only those identifiers mentioned in the log. (In one instance, it was necessary to include an identifier that does not appear in the log for the problem to be solvable. While there are a number of ways that such missing identifiers could be inferred, we have no general solution to this issue.) Finally, each log begins at the start of a mission, with the system in an idle state. Thus, we can assume a single, fully known initial state.

Encoding Observations Some transitions emit an observable event. Recall that in the discrete event diagnosis problem, we have a set of observations, $O = \{o_1, \ldots, o_n\}$, each labelled with an event $\mathrm{event}(o_i)$, and a partial order $\prec$ on O, and we seek a transition sequence that reproduces the observations. This can be formulated as a planning goal as follows: Each observation o can be in one of three states: (future o), meaning that some observation ordered before o is yet to be made; (pending o), meaning that o can be the next observation made; and (observed o), meaning that o has been generated. The initial state is (pending o) if o is minimal in the order on observations and (future o) otherwise. The goal is (observed o), for all $o \in O$. Each action corresponding to a transition that emits an observable event e is given an additional parameter ?o; its precondition is extended with (pending ?o) and $\mathrm{event}(?o) = e$, and its effect with (not (pending ?o)) and (observed ?o). An action corresponding to several synchronised transitions that emit events will have one such observation parameter for each event. These parameters are required to be distinct. To ensure that observations are made consistently with the given order, an observation o can change state from (future o) to (pending o) only when all observations preceding o have been made. We encode this with an action, (advance-to o), whose precondition is (future o) and (observed o') for all $o' \prec o$, and whose effect is (not (future o)) and (pending o).

Proposition 1 *This encoding is correct, in the sense that any valid plan will generate each observation in O exactly once, in an order that is consistent with the given order $\prec$.*

[2] A detailed presentation of the WITAS project and the architecture of the control system is provided by Doherty *et al.* (2004). See also `http://www.ida.liu.se/ext/witas/`.

Proof: For each observation $o \in O$, a valid plan must contain at least one action that exchanges (pending o) for (observed o), and no more than one since it is not possible to reverse the exchange. By construction, this action corresponds to a transition that emits $\mathrm{event}(o)$. Suppose two observations $o' \prec o$ are generated in an inconsistent order, i.e., o before o': Since o is not minimal in $\prec$, (future o) holds in the initial state, and the only action that may exchange (future o) for (pending o), which is a precondition of any action generating o, is (advance-to o). But the precondition of this action includes (observed o'), so it cannot be applied before o' has been generated. □

A slightly simpler encoding would distinguish only between observations made and not made, with the ordering condition part of the precondition of each action emitting an observable event. However, distinguishing future and pending observations allows using observations as "time stamps", which will be useful in modelling our second case study.

Events in the flight logs are stamped with the time that the event was generated. The order on observations is still partial, because events separated by too small a margin (less than $1/100$th of a second) cannot be reliably ordered. It is, however, a special kind of partial order, namely, a sequence of sets of mutually unordered elements.

Encoding Faults The faults that we wish to detect are lost events. Thus, for every transition that emits an observable event, the model has an identical fault transition without any observation. In the PDDL formulation, actions corresponding to fault transitions have a cost of 1 while all non-fault actions have a cost of zero, and the objective is to minimise the cost of the plan. In particular, a zero-cost plan exists iff the observations can be explained without missing events.

Experiments and Results

The data set comprises 8 logs, ranging in length from 41 to 273 observations. Five are complete, while three have between 5 and 17 missing events. The number of model components (i.e., TP instances) ranges from 3 to 26, and the number of states per component between 20 and 129. To obtain a larger set of problems for experimentation, we take prefixes of these logs, of increasing length, and remove randomly chosen events up to a desired total. This way, we obtain 196 instances, 36 of which are complete.

Since the main task is to decide if a zero-cost plan exists, it is natural to use planner that guarantees minimal-cost plans. For this we use the Fast Downward implementation of A* search with the admissible landmark-cut heuristic (Helmert and Domshlak 2009).[3] This planner finds zero-cost plans for all 36 instances without missing events: 30 problems are solved in less than 10 seconds, but the longest runtime for a problem is over 150 seconds. It does not solve all 160 instances with missing events. However, it does prove a lower bound on cost greater than zero – thus *detecting* that some event must be missing – for all but one of these problems, never taking more than 3 seconds to do so. (The one problem were it fails to detect missing events actually admits a zero cost solution, so in this case the blame lies with the model.) In fact, only applying the landmark-cut heuristic to the initial state is sufficient to detect some event is missing in 150 instances (including the three full-length logs).

We also use a cost-ignorant planner: greedy best-first search using the FF heuristic (also implemented in Fast Downward). It solves all but two problems, but generates false positives, i.e., plans of non-zero cost, for 35 of the 36 logs without missing events. However, when run on a model without fault transitions, and thus forced to find only zero-fault plans, the planner solves all instances that admit such solutions, even within a 30 second time limit. In summary, using the combination of two one-sided tests – an admissible heuristic for fault detection and a fast planner for "no-fault detection" – seems to be a viable approach, though it fails to reach a decision for a few problems.

The SAT-based diagnoser (Grastien et al. 2007) suffers from the fact that explanation trajectories are very long, and fails to solve any problem with more than 100 observations. The SAT-based planner that we tried (Mp, by Rintanen 2010) exhibits the same behaviour, though it scales a little further, solving one problem with 170 observations.

Second Case Study: Alarm Processing

Input data for our second case study is an alarm log from the operations center of TransGrid, the company that owns and operates the electricity transmission network in NSW and the ACT, Australia. The log contains alarms generated by automatic equipment – switch gear, voltage and power sensors and regulators, etc. – located throughout the transmission network, as well as commands issued by the operators. It covers roughly fifteen hours: the first two thirds are routine operation, then a major fault situation arises and the rest of the log chronicles the operators' efforts to reconfigure the network to restore service. Figure 2 gives an indication of how alarms are distributed over time.

Our aim is to do "intelligent" alarm processing, which begins with finding a consistent system history with the fewest "unexplained" events. What is an unexplained event can depend on context. For example, an alarm indicating that a circuit breaker has opened may be explained by the observation that it was commanded to open a short time earlier. Four breakers isolating a line (cf. figure 1) opening almost simultaneously may be explained by an electrical fault on the line triggering the line protection relays, if the line was energised. In this case, the occurrence of the line fault is itself an unexplained (and unobservable) event. If no reason for the breaker opening is discernable within the model, the alarm remains unexplained.

The log contains 2246 entries (alarms and commands) in total. However, our model considers only a subset of alarm types and restricted to these the total number of observations is only 731. The model is also overly simplified in some other respects (discussed below); to achieve a level of alarm processing that would truly benefit end users will require a much more detailed and comprehensive model. This, however, is not a limitation of the approach of formulating alarm

[3] We replaced the Fast Downward translator component with a different translator. Except where otherwise noted, experiments were run with 30 minutes CPU time and 2Gb memory limits.

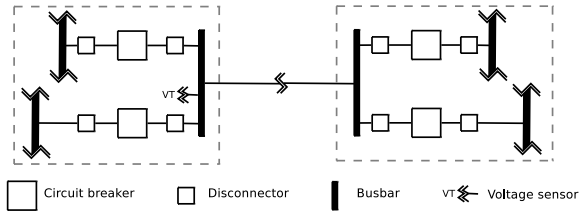


Figure 1: Schematic of a part of the transmission network, showing a (long-distance) line and its isolators.

processing as a diagnosis problem, or of reducing this to a planning problem: the limitation is in our current knowledge of the system, and in the ability of existing planners to reason with a more detailed model.

Formulation as a Planning Problem

We model components of the system as non-deterministic finite state machines, and use a combination of synchronised transitions and shared variables to model their interactions. The dynamics of the system depend a great deal on the propagation of electricity through the network. Most of this we abstract away from the model, since to capture it would require a complex hybrid continuous/discrete model. Moreover, the electrical state of any component can depend on network-wide topological properties, such as the existence of a conducting path to an active generator. Such conditions could be modelled using PDDLs derived predicates, as done by Hoffmann et al. (2006) for the PSR domain. However, the range of planners that support this feature is very limited, and the size of the network – especially coupled with the incomplete initial state – mean that such a formulation would most likely be intractable.

Therefore, the model includes only local topological information, and reasoning about electrical properties is limited to what can be inferred from this. For example, the circuit breakers in figure 1 are all marked as being isolators of the line. Any transition that changes the state of one of these breakers to "open" flags that the isolation state of the line may have changed. This enables the line to transition to being isolated (and having changed), if all isolators are now open. When the line has changed to the isolated state, we know it is not energized, and this can explain an alarm signalling voltage on the line dropping to zero shortly thereafter. Again, this is an example of trading reduced explanatory power for a a simpler, but still valid, model. For example, the line may also become de-energized as a result of all generators currently feeding it switching off, but this explanation is not discernible in our model, so in that case the voltage drop alarm would go unexplained.

Because the model does not use global relations, such as connectivity, the set of components that can influence any given component is bounded. Thus, when creating a planning problem instance for a particular event log, we include only components mentioned in the log, or directly related to those mentioned in the log. This reduces the size of problems significantly: the whole network has over 10,000 components in the primary electrical system alone, while the largest problems we consider contain a few hundred.

Encoding an Incomplete Initial State In this study, we have no knowledge of the initial state.[4] However, a diagnosis is a consistent system history starting from *some* possible initial state, so we can let the planner choose the values of initially unknown state variables (as noted by Sohrabi et al. 2010). This is done by initialising unknown variables to an "unknown" value and including actions that allow them to be set, once, to any value. Variables may remain unknown as long as no action that depends on them is taken.

Encoding Time-Dependent Conditions As illustrated by many of the examples above, time plays an important role in the dynamics of the electricity network, and it is therefore important to include some time constraints into the system model. For example, if a breaker opens, causing a line to become isolated, a voltage drop alarm may be emitted. But if so, we expect that alarm to follow within a few seconds of the breaker opening: a change in isolation state cannot explain a voltage drop that takes place hours later. In other words, the line component should remain in the "changed to isolated" state only for a limited time, and then transition to a different state (although still isolated). Similarly, a fault causing protection to trip should cause all isolation breakers to open within a second of each other – if they don't, protection tripping the breakers is not a plausible explanation.

We can model such time constraints using the durative actions of PDDL2.1. However, we can also make use of the observations as "time stamps", that way obtaining a classical planning model. This is advantageous because the classical model is accessible to many more planners. We found only one planner capable of solving the timed PDDL2.1 model, and even that planner performs better on the classical model!

As in our first case study, observations in the event log are time stamped (though in this case only to one seconds precision). Thus, for any pair of observations we know the delay, $\delta(o_i, o_j) = \tau(o_j) - \tau(o_i)$, between them, and order the observations when that delay exceeds a fixed threshold: $o_i \prec o_j$ iff $\delta(o_i, o_j) > T$. (We use $T = 0$, i.e., we order observations whenever there is a noticable difference in their time stamps. A greater threshold is appropriate when time stamps are not accurate.)

The encoding of time constraints in the classical model is most easily explained by an example: Suppose (open-isolator ?b ?l ?o) is the action of opening breaker ?b, which is an isolator of line ?l (?o is the observation to be matched by the event, signalling the breaker opening, emitted by the transition). As described above, besides changing the state of ?b to open, this action adds a proposition (iso-maybe-chgd ?l), representing that the isolation state of ?l may have changed. Now, we want to model that this proposition will only persist for a fixed time (say, 10 seconds). To achieve this, we add to the predicate iso-maybe-chgd an extra pa-

[4] In actual application, this would likely not be the case. The state of network devices is polled on a regular basis, and even if that information is not used, the log of past events would indicate the state of many components. The case of an unknown initial state may be thought of as a "cold start" of the alarm processor.

rameter ?o, to be filled by an observation that is "current" at the time when (iso-maybe-chgd ?l ?o) is made true. This way, the observation acts as a time stamp for the proposition. An observation is current if (pending ?o) is true. (Action (open-isolator ?b ?l ?o) already has an observation argument, which must be pending when the action takes place. To any action that adds a transient condition and which does not emit an observation, we must add such an observation argument.) Every occurrence of iso-maybe-chgd in the precondition of an action must now existentially quantify the extra observation parameter. (For simple conjunctive action preconditions, existential quantification is equivalent to adding an extra action parameter.) To ensure that (iso-maybe-chgd ?l ?o) does not remain true beyond its time limit, we add to each (advance-to o) action the effect (not (iso-maybe-chgd ?l o')) for each o' such that $\delta(o', o)$ exceeds the limit.

Proposition 2 *This encoding is correct, in the sense that any valid plan can be scheduled in a way that respects both observation time stamps and the maximum time lag constraint, to within the threshold T.*

Proof: Transitions are instantaneous. Schedule each transition t_i that generates an observation o exactly in the middle between the earliest and latest time of all observations that are pending when t_i takes place. They cannot be more than T apart, and hence t_i's scheduled time is no more than $T/2$ from $\tau(o)$. Suppose t_i adds a transient proposition, $p(o)$, with time window w (o is the observation that serves as time stamp for p). If t_i is not the transition that generates o, schedule it at the time of the next transition t' that generates an observation o'. Since o and o' are both pending at this point, t_i's scheduled time is no more than $T/2$ from $\tau(o)$. Suppose t_j requires $p(o)$: the scheduled time of any transition t' between t_i and t_j in the sequence which generates an observation o' is no more than $T/2 + w$ later than $\tau(o)$ (as otherwise (advance-to o') would negate $p(o)$). Thus it is possible to schedule t_j no later than $T + w$ after t_i. □

This encoding can be over-constraining, in the sense that in some system models there could be trajectories consistent with time constraints that do not correspond to valid plans. However, this does not happen in our model.

The encoding has some similarity to the use of "envelope" actions in temporal modelling, as suggested by Fox et al. (2004). Indeed, our PDDL2.1 model uses precisely such envelope actions to achieve the same effect.

Experiments and Results

No system that we tried was capable of finding a solution to the problem corresponding to the complete event log. However, it may be argued that an alarm processor should only have to consider windows of time spanning sets of causally related observations. To obtain a set of problem instances for evaluation, we divide the event log up in two ways: (1) by splitting it into "chunks" separated by intervals of at least 1 minute during which no alarm is observed, and (2) by taking a 1 minute time "window" starting from each distinct alarm time stamp. Figure 2 shows the distribution of observations over the "chunk" problems: while most have only a few observations, the alarm cascade that results from the fault incident creates a few large spikes. Those are precisely the times when intelligent alarm processing is most needed.

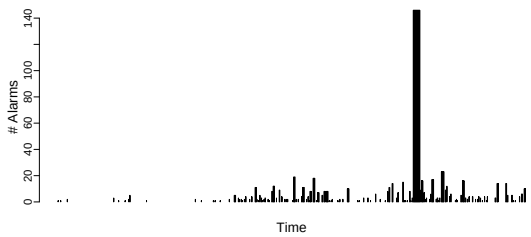


Figure 2: Number of observations in each "chunk" of the log (bar width and spacing is proportional to time). Counts include only the subset of alarm types included in the model.

For each subsection of the log, we estimate the number of unexplained alarms that would result without processing by counting all except commands and command acknowledgements. Removing problems where the highest known lower bound matches this estimate leaves 129 problem instances in which there is scope for alarm processing to make a non-trivial improvement. Table 3(a) summarises the number of instances solved within the time limit, grouped by problem size as measured by the number of observations. The systems we compare are:

- The cost optimal A*/LM-Cut planner.
- Gamer (cost optimal; Edelkamp and Kissmann 2008).
- LAMA (Richter and Westphal 2010).[5]
- Greedy best-first search using the cost-ignorant FF heuristic (Fast Downward implementation). To find better plans, we also use two variants: one continues search past the first solution and one runs many repeated searches, randomising decisions that are otherwise made arbitrarily.
- Mp (a heuristically-enhanced SAT-planner, also cost-ignorant; cf. Rintanen, 2010).
- Crikey, a heuristic-search based temporal planner (Coles et al. 2008), run on both the timed (PDDL2.1) and the classical model.
- The SAT-based diagnosis engine (Grastien et al. 2007).

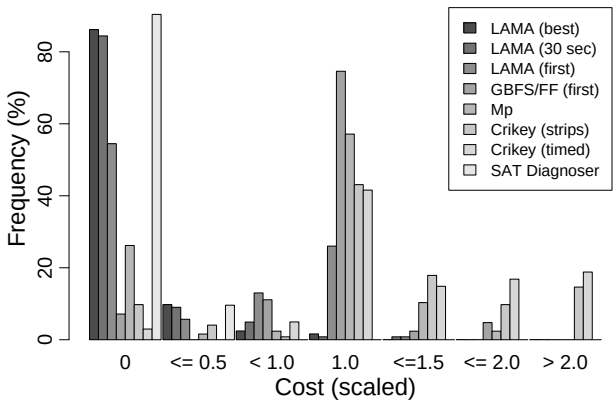
Figure 3(b) shows the distribution of the quality of non-optimal solutions. LAMA, and our two GBFS/FF variants, may find better solutions given more time. In figure 3(b), we separate the quality of the first solution, the best found within 30 seconds, and the best solution found at all.

Results are quite predictable: The cost-optimal planners cannot solve large problems, while the cost-ignorant plan-

[5] LAMA avoids issues related to zero-cost actions in search by uniformly adding 1 to all action costs. To better preserve the distinction between "good" (explained) and "bad" (unexplained) transitions through this transformation, we scale up the cost of bad transitions to 10 in the input to this planner. (The plans it finds are of course evaluated using same costs costs as for all other planners, i.e., zero for good and one for bad transitions.) If we do not apply scaling, LAMA is slightly more efficient, but finds worse (first and best) solutions.

	# Observations						
	1-5	6-10	11-20	21-30	31-50	51-100	>100
# Problems	36	43	35	6	4	4	1
# Solved by							
A*/LM-Cut	36	43	30	0	0	0	0
Gamer	36	43	31	0	0	0	0
LAMA	36	43	35	6	4	2	0
GBFS/FF	36	43	35	6	4	4	1
Mp	36	43	35	6	4	4	1
Crikey (timed)	36	40	19	6	2	1	0
Crikey (strips)	36	43	35	6	4	2	0
Diagnoser	36	43	35	6	4	4	0
# Optimal	36	43	35	1	1	0	0

(a)



(b)

Figure 3: (a) Number of alarm processing problems solved. Problems are grouped by the number of observations, as a measure of size. The last line shows the number of problems for which optimal solution cost is known. (The two optimal planners solve slightly different sets of problems. There are also some cases where a solution found by one of the other systems matches the highest proven lower bound.) (b) Distribution of the cost of solutions found by non-optimal systems, scaled to the interval between the lower bound and the cost of the trivial solution, with no alarm processing (i.e., 0 means equal to the lower bound, 1 means equal to the trivial solution).

ners, though fast, find solutions that are often as bad as using no alarm processing at all, and sometimes even worse than that. LAMA, and continued GBFS, strike a balance between these extremes, finding good solutions quickly most of the time. However, both fail to find improving solutions for the largest instances. Repeated GBFS with randomisation is generally worse, but is the only method to find a solution better than the trivial one for the largest instance. The diagnoser solves all problems but one, though somewhat slowly. It produces minimal-cost solutions, but because it is based on a SAT encoding, these are only optimal w.r.t. a bound (on the number of "steps" between observations), which was fixed in this experiment. However, no other solver finds a better solution for any instance.

We also measure the impact of incomplete knowledge of the initial state, by running the optimal planners on instances with complete initial states (taken from the solutions found by the diagnoser). In this setting, the A*/LM-Cut planner solves an additional 13 problems and Gamer an additional 6, but still neither planner solves any problem with more than 50 observations.

Discussion & Conclusions

We summarise our findings by discussing three questions:

Are the case study problems solved? For our first case study, the combination of two one-sided tests, viz. using an admissible heuristic such as the landmark-cut procedure to detect event loss and running a fast planner on a fault-free model to identify loss-free scenarios, seems to be a viable approach (though it does not decide every problem).

For intelligent alarm processing, existing approaches use either very simple system models (most not even including any notion of time) or lack capability to choose the best among competing explanations. In contrast, formulating it as a discrete event diagnosis problem allows for minimisation of unexplained events over very expressive system models. On the other hand, it is clear that this problem cannot yet be solved efficiently: Planners that deliver solutions quickly find solutions that are often as bad as using no alarm processing at all. The diagnoser produces high quality solutions, but is still too slow to be practical. LAMA makes a valiant attempt to balance fast plan generation and the quality of plans found, and does produce fairly good solutions, even within a 30 second time limit. However, it, and the other iterated search methods we tried, fail to find a solution better than the trivial for the problems with the highest number of alarms. Since those represent precisely the situations when effective alarm reduction is most sorely needed, we cannot call this an acceptable performance overall. It must also be remembered that our model is highly simplified, and covers only a subset of alarm types. To achieve the level of alarm filtering and synthesis that would truly benefit end users we would need a much more sophisticated model, placing a higher computational burden on the system used to solve it.

What can planning technology contribute to discrete event diagnosis? The question we seek to address is if and how the techniques that have been successful in solving planning problems can be effectively brought to bear on the problem of discrete event system diagnosis. To this end, we devised a general reduction of this diagnosis problem to planning, and used it to test a variety of planning approaches.

It is not so easy to identify the "state of the art" in discrete

event diagnosis, for several reasons: what are reasonable assumptions and what is a useful diagnosis differs between applications, and quantitative data on the relative performance of different approaches is scarce. Among approaches proposed in the literature for the kind of problem we consider, many rely on expensive off-line preprocesing of the model, making them unusable for our first case study, where the set of components is dynamic, and severely impractical for the second, where the complete system has some 10,000 components. Decentralised diagnosis methods (e.g., Pencolé and Cordier 2005) may be usable for the second case study.

Nevertheless, we have shown that there are planning techniques, such as the use of an admissible heuristic to identify necessary fault events, which have not previously been applied to diagnosis problems and which show enough promise to merit further development. On the other hand, it is also clear that direct reduction to planning is not a universal solution to discrete event diagnosis.

What are the implications for planning research? We believe that our case study problems are interesting and challenging also for planning, since they highlight some issues not commonly encountered in other benchmark domains.

One such issue is the essential requirement of taking into account action costs. It has been noted (e.g., by Richter and Westphal 2010) that in many benchmark domains, planners that ignore action costs frequently find solutions as good as – sometimes even better than – those found by planners that pay attention to cost and try to minimise it, and that planners of the former kind have an advantage in efficiency. Likewise, Sohrabi et al. (2010) report that cost-ignorant planners always find near-optimal solutions for the artificial example problem they consider. The problems we study clearly demonstrate that this phenomenon is an artefact of those particular problem domains, not a universal rule.

Acknowledgements

We thank TransGrid and the LiU UAVTech group for their permission to use the data, and Sylvie Thiébaux for many helpful discussions. This work was supported by ARC project DP0985532. NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

References

Basile, F.; Chiacchio, P.; and De Tommasi, G. 2003. An efficient approach for online diagnosis of discrete event systems. *IEEE Trans. on Automatic Control* 54(4):748–759.

Coles, A.; Fox, M.; Long, D.; and Smith, A. 2008. Planning with problems requiring temporal coordination. In *AAAI'08*.

Cordier, M., and Thiébaux, S. 1994. Event-based diagnosis for evolutive systems. In *DX'94*, 64–69.

de Kleer, J., and Williams, B. 1989. Diagnosis with behavioral modes. In *IJCAI'89*, 1324–1330.

Dijk, H. 1992. AI-based techniques for alarm handling. *Int'l Journal of Electrical Power & Energy* 14(2–3):131–137.

Doherty, P.; Haslum, P.; Heintz, F.; Merz, T.; Persson, T.; and Wingman, B. 2004. A distributed architecture for intelligent unmanned aerial vehicle experimentation. In *Proc. 7th Int'l Symp. on Distributed Autonomous Robotic Systems*.

Dousson, C. 1996. Alarm driven supervision for telecommunication network: II – on-line chronicle recognition. *Annals of Telecommunications* 51(9-10):501–508.

Edelkamp, S., and Kissmann, P. 2008. GAMER: Bridging planning and general game playing with symbolic search. In *IPC 2008*.

Fox, M.; Long, D.; and Halsey, K. 2004. An investigation into the expressive power of PDDL2.1. In *ECAI'04*, 338–342.

Grastien, A.; Anbulagan; Rintanen, J.; and Kelareva, E. 2007. Diagnosis of discrete-event systems using satisfiability algorithms. In *AAAI'07*.

Guo, W.; Wen, F.; Liao, Z.; Wei, L.; and Xin, J. 2010. An analytic model-based approach for power system alarm processing employing temporal constraint network. *IEEE Trans. on Power Delivery* 25(4):2435–2447.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In *ICAPS'09*.

Hoffmann, J.; Edelkamp, S.; Thiébaux, S.; Englert, R.; Liporace, F.; and Trüg, S. 2006. Engineering benchmarks for planning: the domains used in the deterministic part of IPC-4. *Journal of AI Research* 26:453–541.

Kuhn, L.; Price, B.; de Kleer, J.; Do, M.; and Zhou, R. 2008. Pervasive diagnosis: The integration of diagnostic goals into production plans. In *AAAI'08*, 1306–1312.

Larsson, J. 2009. Real-time root cause analysis with multilevel flow models. In *DX'09*.

McIlraith, S. 1994. Toward a theory of diagnosis, testing and repair. In *DX'94*, 185–192.

Pencolé, Y., and Cordier, M. 2005. A formal framework for the decentralised diagnosis of large scale discrete event systems and its application to telecommunications networks. *Artificial Intelligence*.

Prince, W.; Wollenberg, B.; and Bertagnolli, D. 1989. Survey on excessive alarms. *IEEE Trans. on Power Systems* 4(3):950–956.

Reiter, R. 1987. A theory of diagnosis from first principles. *Artificial Intelligence* 32:57–95.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of AI Research* 39:127–177.

Rintanen, J. 2010. Heuristics for planning with SAT. In *CP'10*, 414–428.

Sampath, M.; Sengupta, R.; Lafortune, S.; Sinnamohideen, K.; and Teneketzis, D. 1996. Failure diagnosis using discrete-event models. *IEEE Trans. on Control Systems Technology* 4(2):105–124.

Sohrabi, S.; Baier, J.; and McIlraith, S. 2010. Diagnosis as planning revisited. In *KR'10*.

Computing Genome Edit Distances using Domain-Independent Planning

P@trik Haslum
Australian National University
firstname.*lastname*@anu.edu.au

Abstract

The use of planning for computing genome edit distances was suggested by Erdem and Tillier in 2005, but to date there has been no study of how well domain-independent planners are able to solve this problem. This paper reports on experiments with several PDDL formulations of the problem, using several state-of-the-art planners. The main observations are, first, that the problem formulation that is easiest for planners to deal with is not the obvious one, and, second, that plan quality – in particular consistent and assured plan quality – remains the biggest challenge.

Introduction

For several decades now, the comparative study of biological sequence data (i.e., DNA or protein amino acid sequences) has played an increaing role in determining the evolutionary history of life on Earth (e.g., Page and Holmes 1998). While early studies examined differences in the nucleotide sequences of individual genes, more recent work has also examined differences in the arrangement of genes across a whole genome (Sankoff et al. 1992; Boore and Brown 1998; Snel, Huynen, and Dutilh 2005).

A computational problem that arises in this context is the calculation of edit distances between sequences: that is, the number of "mutation events" required to transform one sequence into another. The edit operations considered depends on the type of sequence. For DNA sequences, they are typically insertion, deletion or substitution of individual nucleotides. Minimising this edit distance is the famous sequence alignment problem. For genome rearrangement, the edit operations usually assumed are inversions and transpositions (and combined transposition and inversion, called transversions) of subsequences within the genome, plus insertion/duplication and deletion if needed to account for differences in gene content.

Erdem and Tillier (2005) suggested that the calculation of genome edit distance under these operations can be considered as a planning problem: The arrangement of genes is the state, and each edit operation is an action that modifies it. These actions may furthermore be assigned different costs, to account for different relative frequencies with which they are assumed to occur in the organisms studied. They developed a solution to the problem based on TLPlan, which was later refined by Uras and Erdem (2010).

However, the system developed by Erdem and colleagues is a plain depth-first state-space search guided by entirely domain-specific heuristics. Although the problem is formulated in a planning language, the solver makes essentially no use of this. Consequently, there has been no comprehensive study of how well state-of-the-art domain-independent planners are able to solve the genome edit distance problem.[1]

In this paper, I discuss alternative encodings of the genome edit distance problem in PDDL, and evaluate the ability of some current domain-independent planners to solve the resulting formulations. My aim is not to show that domain-independent planners are better than Erdem et al.'s system (quite unlikely, given the extent to which it is tailored to the domain), but to find out exactly how well – or badly – such planners perform on this problem, and what shortcommings of these planners may need to be addressed to improve their usefulness.

Phylogenetic Reconstruction and Genome Edit Distance

A phylogeny (evolutionary tree) is a tree where the leaf nodes represent living taxa (species or species groups) and interior nodes the ancestral organisms from which they have evolved. Typically, these ancestors are long extinct. The problem of phylogenetic reconstruction is to find the best, i.e., most plausible, such tree, based on observed characters of the living taxa. In a genome-based phylogenetic analysis, these characters are the content and arrangement of genes in the organisms genomes. What constitutes a "most plausible" phylogeny depends on what criteria are assumed. A common criterion is parismony, i.e., preferring trees with the smallest total amount of change. Criteria such as maximum likelihood can also be used, but depend on additional assumptions about the evolutionary process.

Finding a maximum parsimony phylogeny based on genome rearrangement events is computationally hard. Distance-based methods simplify the problem by divding it into two steps: The first is to compute a matrix of pair-wise distances between the taxa, and the second to construct a tree with minimum total distance. Optimal tree construction

[1] Erdem and Tillier report that they formulated a highly simplified version of the problem in PDDL, and had little success solving it using HSP and SATPLAN.

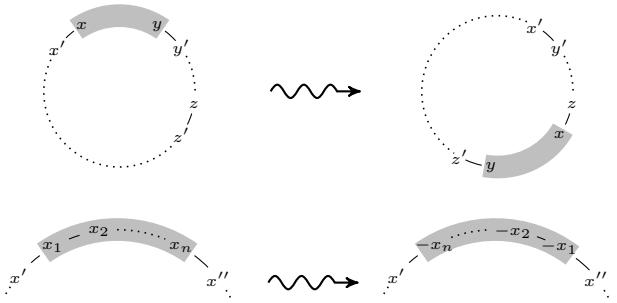


Figure 1: Edit operations on a circular genome. Above: Transposition of the x–y segment to after z. Below: Inversion of the x_1–x_n segment.

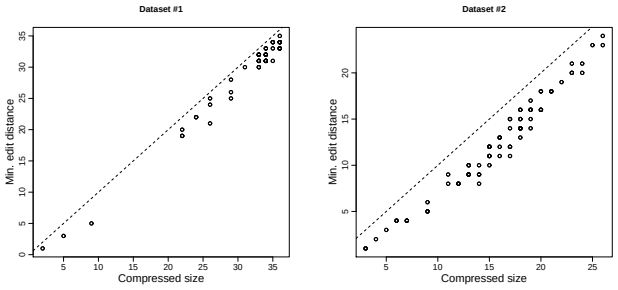


Figure 2: Comparison between compressed genome size and (weighted) edit distance. Note that this is the smallest known edit distance, and does not necessarily reflect the true minimum.

is still hard, but there are widely used and seemingly good approximate methods (e.g., Saitou and Nei 1987). Computing the entires in the distance matrix in this setting gives rise to the genome edit distance problem. Note that the distance-based tree construction is an approximation: even if distances correspond to optimal rearrangement sequences, a minimum distance tree is not necessarily a most parsimonious tree.

Genome Edit Distance

A genome is a linear or circular sequence of genes. In addition to their order, each gene has an orientation ("normal" or "inverted", relative to the, arbitrarily chosen, direction of the sequence). The definition of an edit distance measure requires defining the edit operations, and their relative weights. For comparison of genomes with equal gene content, i.e., which differ only in the order and orientation of their genes, the usual edit operations are inversion and transposition, and the combination of both, called transversion. Transposition moves a segment of the genome (which may consist of a single gene) to a different location, while inversion reverses a segment and reverses the orientation of each gene within it. The two operations are schematically illustrated in figure 1. Transversion simultaneusly inverts a segment and moves it to a new location. For comparing genomes with unequal content, operations such as insertion, duplication and deletion of genes must also be used. The inversion-only distance can be computed in polynomial time (Hannenhalli and Pevzner 1995) but no polynomial time algorithm is known for the larger set of edit operations. The problem is conjectured to be NP-hard.

That edit operations are not equally frequent is reflected by giving them different weight in the distance calculation. The relative frequency of apparent occurrence of transposition and inversion is not known precisely, and varies between different species groups. Blanchette et al. (1996) suggest a relative weight of 2–2.5 for transpositions and transversions to 1 for inversions. This weight range is where the number of operations in an approximately minimum weight transformation between one pair of mitochondrial genomes (human and a *Drosophila*) diverges from the number between random sequences.

Although differences that can be ascribed to inversions and transpositions can be observed in the genomes of related organisms, the mechanisms that cause these changes are not well understood. Some studies have suggested that the apparent transpositions and inversions are in fact the result of duplications followed by gene loss (Lavrov, Boore, and Brown 2002). Thus, there are reasons to experiment with different sets of edit operations, and different weighting schemes. The generality of domain-independent planning can offer an advantage in that.

Problem Simplification

Uras and Erdem (2010) describe two simplifications that they apply to problem instances before attempting to solve them. The first is only relevant when comparing genomes with unequal content, and is to remove from both genomes being compared any gene that appears only in one of them, and adding the number of such deletions to the edit cost. (Each such gene must obviously be inserted/deleted, which can be done at the beginning/end of the plan.)

The second is to "compress" both genomes by replacing common substrings with (new) atomic symbols, since there is no reason to apply any edit operation that breaks up such a substring. Note that common substrings may appear inverted in one of the genomes. Compression is done pairwise, per problem, so the substrings need not be common to all genomes in the data set. This simplification turns out to be very important for performance, since it can drastically reduce the size of the problem. Genomes in the second data set (cloroplast genomes of 13 plants from the *Campanulaceae* family) contain 105 genes each, but after compression no pair has more than 26 elements (genes or substrings). Size after compression also turns out to be a fairly good predictor of the edit distance: figure 2 shows a comparison. (This was also noted by Nadeau & Taylor 1984.) In data set #1 (mitochondrial genomes of 11 animal species), which generally has a much higher degree of rearrangement between genome pairs, the effect of compression is much less dramatic.

Formulation in PDDL

In Erdem and Tillier's formulation, the arrangement of genes is described by a binary predicate, (cw ?x ?y), with the meaning that ?y is the next gene from ?x in clockwise order. I will call this the "relational" encoding. The alternative is a "positional" encoding, i.e., to specify the position of each gene w.r.t. a fixed frame of reference, for example by a predicate (at ?x ?p) meaning gene ?x is at position ?p. The orientation of each individual gene is additionally specified by one of the predicates (normal ?x) and (inverted ?x) being true. Both encodings have their advantages and disadvantages.

As noted above, correct and precise relative weights for transpositions and inversions are not known. What is important when formulating edit distance computation as a planning problem is allowing for the possibility of specifying different weights, and generating plans that minimise the weighted distance.

Formulations Based on the Relational Encoding

It is easy to think of each edit operation as an action, but these actions are not so easy to formulate in PDDL, because they have complex preconditions and/or effects. Consider, for example, inversion: the segment to be inverted can be defined by the two genes at its ends, but the operation will have an effect also on every gene between them. The precondition of transposition must enforce that the new location (gene z in figure 1) does not lie within the segment that is moved (i.e., not between x and y). Since the relational encoding specifies only the "neighbours" in the genome, this "betweeness" is a transitive closure property. There are (at least) three ways to formulate the operations:

1. Use PDDL2.2's derived predicates and axioms to define the between and not-between properties, and use quantified conditional effects to encode the effect of inversion. This is, essentially, the formulation used in Erdem et al.'s TLPlan-based system. Note that the recursive derived predicates used in this formulation cannot be expressed in ADL, which allows only first-order pre- and effect conditions.[2]
2. Break each operation up into a sequence of actions, each of which affects only a fixed-size genome part (e.g., only two neighbouring genes). This requires additional predicates to control the sequencing of these actions so that they actually correspond to complete and correct edit operations.
3. Use a separate action for each size of segment operated on, with a matching number of arguments (for example, an action (invert-3 ?x-pre ?x ?y ?z ?z-post) for inverting the segment ?x–?y–?z of length 3).

The third option is used in Erdem & Tillier's simplified PDDL formulation, which permits only operations on segments of limited size. For the general problem, it becomes infeasible because of the very large number of action parameters, which make grounding impossible. (Grounding, which is used by nearly all modern domain-independent planners, is a significant obstacle for some other formulations as well, as discussed below.)

Single-Step Formulation In the single-step formulation, each edit operation is performed by one action, using derived predicates and quantified conditional effects to specify actions' preconditions and effects. Formulating axioms for between is straightforward. For example,

```
(:derived (between ?x ?y ?z) (= ?z ?x))
(:derived (between ?x ?y ?z) (= ?z ?y))
(:derived (between ?x ?y ?z)
          (exists (?w) (and (cw ?x ?w) (not (= ?y ?w))
                            (between ?w ?y ?z)))))
```

i.e., ?z is between ?x and ?y, inclusive, iff ?z equals either ?x or ?y, or the next gene ?w clockwise from ?x is not equal to ?y and ?z is between ?w and ?y. (A similar definition can be written for not-between, so it is not necessary to use negation over the between predicate.) The effect of reversing the cw relation in the segment between ?x and ?y (an effect of the inversion operation) can then be written as:

```
(forall (?v ?w)
  (when (and (between ?x ?y ?v) (between ?x ?y ?w)
             (cw ?v ?w))
        (and (not (cw ?v ?w)) (cw ?w ?v))))
```

Most modern domain-independent planners work (internally) on a grounded representation, and this is a major obstacle to using this formulation. The transposition operation involves six distinct genes whose neighbour relations will change, and thus the transpose action has six arguments. Apart from a few disequalities (e.g., $z \neq x'$; cf. figure 1), all possible instantiations lead to potentially applicable actions, which makes the number of ground actions huge. No problem with genomes containing more than 7 elements could be grounded. Note that the issue here is not the (in-)efficiency of the grounding process, but the size of the grounded problem. Thus, advanced grounding techniques, such as combined grounding and relaxed reachability analysis (Helmert 2009), will not help.

However, since there are at most binary predicates, the effect of an operation can be divided into a series of steps each of which can be performed by an action with only two parameters (e.g., for transposition: break x'–x; break y–y'; connect x'–y'; etc). Part of the inversion operation must still be written using quantified conditional effects, but there are only a quadratic number of instances, each with an at most quadratic number of conditional effects after grounding the quantifiers. In this formulation, all problems (up to size 26) can be grounded effectively.

Multi-Step Formulation The use of derived predicates and conditional effects can be avoided by "simulating" their effects through sequences of actions.[3] This can be done in several ways: the following is just one alternative.

The edit operations can all be viewed as first cutting out a segment of the genome and then re-inserting it somewhere else (transposition), or inserting it reversed in the same place

[2] Because TLPlan, which uses no domain-independent reasoning, uses action descriptions only to generate successor states in a "black box" maner, it has a very expressive input language which includes, for example, procedurally defined (recursive) functions.

[3] This idea is also used in compilations that remove these features (cf. Nebel 2000, and Thiebaux, Hoffmann, and Nebel 2003). Note that these compilations do not preserve plan length, which is why zero-cost actions become necessary to model the relative weight of edit operations in this formulation.

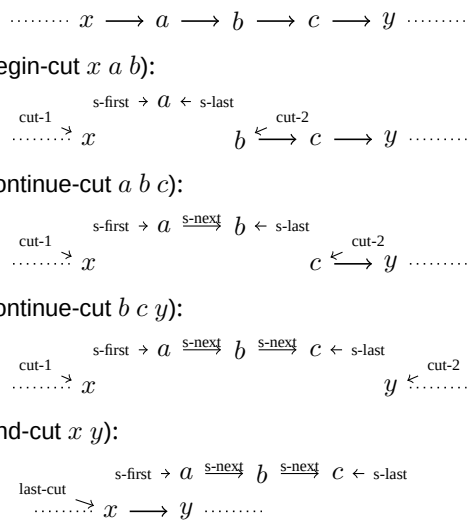


Figure 3: Cutting segment a–b–c. The unlabelled arrows represent the cw relation.

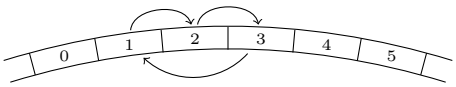
(inversion) or elsewhere (transversion). Cutting, inserting and inserting-in-reverse a segment can all be done in a gene-by-gene fashion. Figure 3 illustrates how a segment of length 3 is cut out. Sequencing of the step actions is controlled by auxiliary predicates: for example, the precondition of action (continue-cut ?x ?y ?z) requires that (s-last ?x), (cut-2 ?y) and (cw ?y ?z) hold. At the end of the operation, the end points and sequence of the cut segment are identified by these auxiliary predicates, so that this information can be used to control insertion. Also marked is the point in the genome where the cut was made: this permits to distinguish inversion, which is a reversed insertion at that point, from transversion, and thus to assign them different weights.

Expressed in a "natural" way, this formulation has actions with three parameters, which is still enough to make grounding large problems difficult. Through further splitting of the steps it can be brought down to two parameters, allowing all problems (up to size 26) to be grounded effectively.

In this formulation, the number of actions in a single edit operation varies with the length of the transposed or inverted segment (this is in contrast to the split version of the single-step formulation described above). Thus, to accurately encode relative weights (not dependent on segment length) it is necessary to assign some actions zero cost.

Formulations Based on the Positional Encoding

In the positional encoding, all complex relations are over positions. For example, transposing the segment at positions 1–2 to after the gene at position 3 (assuming the segment inbetween slides counter-clockwise) results in a permutation moving the gene at position 1 to position 2, the gene at 2 to 3, and the gene at 3 to position 1:



This permutation is the same regardless of which genes occupy the affected positions. Thus, for a given genome size, the permutations caused by edit operations can be computed in advance and provided to the planner through static predicates. For example, the effects of transposing the segment ?x–?y to ?z can be written

```
(forall (?g - gene ?v ?w - pos)
  (when (and (transpose-shift ?x ?y ?z ?v ?w) (at ?g ?v))
        (and (not (at ?g ?v)) (at ?g ?w))))
```

where (transpose-shift ?x ?y ?z ?v ?w) is the static predicate that specifies the permutation, i.e., that the transposition moves the gene at position ?v to position ?w. In the above example, we would have (transpose-shift p1 p2 p3 p1 p2), (transpose-shift p1 p2 p3 p2 p3), and (transpose-shift p1 p2 p3 p3 p1). (The predicate is false for all positions not involved in the move.) This allows a single-action-per-edit-operation domain to be written without derived predicates, and using actions with no more than three parameters. Grounding this formulation is still challenging, but here the size of the grounded problem is moderate (although the number of ground actions is roughly cubic, most do not affect a large part of the genome and so have a relatively modest number of conditional effects). Thus more efficient grounding techniques may make it practical.

It is also worth noting that the formulation could be made more compact by making use of the recent addition of "object fluents" to PDDL (Helmert, Do, and Refanidis 2008), which model multi-valued state variables. For example, the effect of the transposition above could be written

```
(forall (?v ?w - pos)
  (when (transpose-shift ?x ?y ?z ?v ?w)
        (assign (gene-at ?w) (gene-at ?v)))
```

i.e., without quantifying over the possible content of each reassigned position. This would reduce the number of conditional effects to linear. However, there is, to my knowledge, no planner that natively supports effects of this kind. (Fast Downward, and planners derived from it, such as LAMA, internally use a format based on grounded multi-valued state variables, but allow only constants on the left-hand side of assignements.)

The positional encoding has another significant drawback when applied to circular genomes, in that it introduces an arbitrary fixed reference point. Thus, the fact that two circular arrangements may be equal but placed differently w.r.t. this reference point must be taken into account. This can be done by introducing a "rotate" action, which shifts the whole genome relative to the reference point without changing the arrangement. Applications of this action do not count towards the edit distance, i.e., it must have zero cost.

Genome Data Sets

Experiments were done on two data sets.[4]. The same data sets were used by Erdem and Tillier (2005).

Data set #1 comprises the mitochondrial genomes of 11 species of the animal kingdom. They are a diverse collection, with one to three exemplars from each of six major

[4]Obtained from `http://grimm.ucsd.edu/MGR/examples.html`

Formulation		Max Arity	# Grnd	# Sol
			LAMA	
Relational				
Single-step	(R1)	6	18 (7)	18 (7)
Single-step, partly split	(R1′)	4	34 (12)	28 (11)
Single-step, fully split	(R1″)	2	156 (26)	38 (15)
Multi-step	(Rm)	3	156 (26)	156 (26)
Multi-step, split	(Rm')	2	156 (26)	156 (26)
Positional				
Single-step	(P1)	3	122 (19)	40 (15)
			Mp	
Relational				
Multi-step	(Rm)	3	156 (26)	36 (14)
Multi-step, split	(Rm')	2	156 (26)	45 (14)
Positional				
Single-step	(P1)	3	34 (12)	25 (11)

Table 1: Comparison of problem formulations on data set #2. "Max Arity" is the largest number of action parameters. "# Grnd" is the number of instances (out of 156) that could be grounded (and translated to SAS+) within 2Gb memory. In parenthesis, the largest problem size that could be grounded. "# Sol" is the number of instances solved, within a 30 minute time and 2Gb memory limit. In parenthesis, the largest problem size that was solved. Mp was run with a higher memory limit, 3Gb, but a 30 minute time limit for grounding and solving combined. However, the time for grounding and preprocessing is negligeable; when grounding fails it is by exhausting memory.

groups: chordates, echinoderms, arthropods, mollusks, annelids and nematodes. (Each of these groups is believed to be monophyletic.) Blanchette et al. (1999) used this data to investigate gene order evidence for different theories about the evolutionary relationship between these groups. Most mitochondiral genomes have 37 genes, without duplicates. However, one gene is missing from all nematodes, so only the arrangement of the remaining 36 genes is compared. Genomes of species from different groups generally show a high degree of rearrangement, and thus compress poorly, while within some groups the genomes are very similar, resulting in small problems after compression. Thus, in this set there is an uneven spread of problem sizes (cf. figure 2).

Data set #2 comprises the chloroplast genomes of 13 plants from the *Campanulaceae* family. It was used by Cosner et al. (2000) to compare different phylogenetic analysis methods. The genomes contain 105 different genes, some duplicated. Cosner et al. removed duplicate genes, so that each genome has a length of 105. There are large segments common to all genomes in the set, and therefore compression of common substrings reduces the size of problems significantly, to an even spread between 3 and 26.

Experiments

The experiments aim to determine which of the different formulations planners find easiest to deal with, and generally evaluate the ability of some domain-independent planners to compute genome edit distances. Note that this means ability to generate solutions of consistent quality. The purpose of computing edit distances is to *compare* them (i.e, organism A is assumed to be more closely related to organism B than to C if the edit distance between the genomes of A and B is smaller than the distance between A and C.) This does not mean it is necessary to find optimal plans: as long as the relative differences between the distances computed for different pairs is the same as between the minimal edit distances between the same pairs, any information contained in the true edit distances is not lost. If, however, the cost of the plans produced by a planner can differ significantly and unpredictably from the optimal cost, to the extent that this "random" difference overshadows any relation between distances between different pairs, the results are of no use.

Comparison of Formulations Table 1 summarises the comparison of problem formulations. This was done on data set #2, because it shows a fairly even spread of problem sizes. To the extent of my knowledge, the only planner capable of handling all formulations, and trying to minimise plan cost, is LAMA (Richter and Westphal 2010). Thus, the difficulty of solving each formulation is estimated by how many, and how large, problems this planner solves. As mentioned, grounding is major obstactle for problem formulations that use actions with a large number of parameters. It is, however, clearly not the only source of difficulty: in all non-STRIPS formulations, except relational single-step, some problems that could be grounded were not solved. To get some idea of how much these results are specific to LAMA, I also ran the Mp planner (a heuristically enhanced SAT-based planner; cf. Rintanen 2011). Mp ignores plan quality, and thus is not really suited to calculating the weighted edit distance, but its ability to find any solution can still be used as a measure of problem difficulty. As it does not support derived predicates, Mp could only be tested on three formulations, but on these it mostly agrees with LAMA.

Comparison of Edit Distances Figure 4 shows a comparison of the edit distances computed by different planners, as well the inversion-only distance computed by a domain-specific system, GRIMM.[5] The planners are LAMA, Mp and several variants of greedy best-first search with the FF heuristic (Fast Downward implementation). The first two variants use a cost-sensitive version of the heuristic, i.e., one that estimates the true cost of the relaxed plan, and the standard unit-cost version, which estimates the size of the relaxed plan, in a standard, single GBFS search. The third ("cGBFS/FF") uses the unit-cost heuristic, but continues to search after the first solution has been found, for a plan of lower real cost. In the final variant ("GBFS/FF+RR"), some decisions that are normally made arbitrarily (order of successors and the choice of minimum-cost supporting action in the heuristic) are randomised, and the search repeated as many times as possible within the time limit, keeping the best plan found.

The smallest known distance is found by combining the results of all planners, exploiting the symmetry of the edit

[5] `http://grimm.ucsd.edu/GRIMM/`

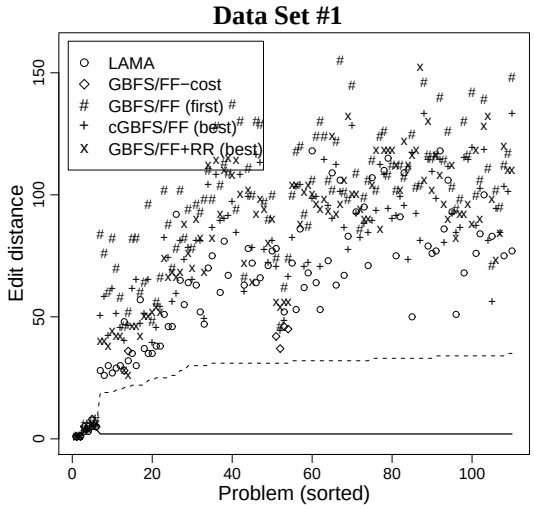


Figure 4: Comparison of edit distances found by different planners on the relational multi-step split formulation (Rm'). Problems are sorted by the minimum (known) distance (shown as a dashed line); this includes the inversion-only distance obtained from GRIMM. The solid line shows the highest lower bound proven by A*/LM-Cut. For data set #2, the lower graph shows a close up of the region below distance 50, for better vertical resolution. This graph also includes the distances computed by Erdem & Tillier's (2005) system ($\cdots$). Note, however, that those were computed with transversion weighted at 1, instead of 2; the corresponding distance under the weighting used here may be anywhere in $[1, 2]$ times that shown in the graph.

distance (the minimum cost of transforming genome g into g' is the same as of transforming g' into g; both problems are solved), and the inversion-only distance computed by GRIMM. (Since a weight of 1 for inversions in used in the planning formulation, the unweighted inversion-only distance is a valid upper bound on optimal plan cost.) For most problems, the latter is clearly smaller, but there are a few instances in which a planner finds a lower-cost plan. For data set #2 the comparison also includes the distances computed by Erdem & Tillier's TLPlan-based system. These distances, however, were computed with transversion weighted at 1, and thus are not directly comparable. (Note that this distance is sometimes below the lower bound!) The corresponding distance under the weighting used here may be anywhere in $[1, 2]$ times that shown in the graph. For data set #1, the weights used by Erdem & Tillier differ too much from those used here to allow for any meaningful comparison. Finally, the highest f-value proven by A* search using the LM-Cut heuristic is provided as a lower bound.

Three important observations can be made: First, the cost-ignorant planners exhibit a very large spread in plan costs, relative to the smallest known. This makes them unusable for computing edit distances, since the noise that is created by the planners is clearly enough to drown out any signal present in the true minimum distances. Second, using the

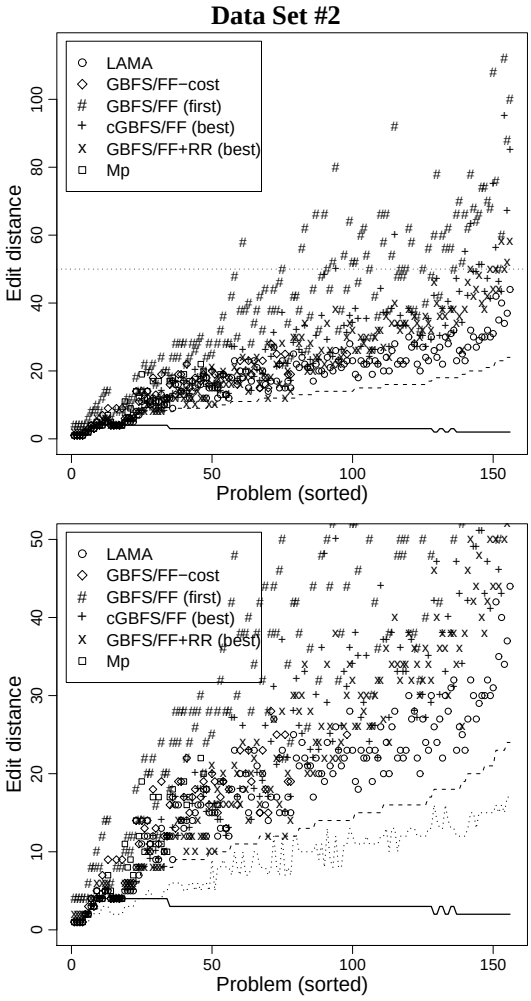


Figure 4 (continued).

cost-sensitive FF heuristic results in plans of much better quality, but is much less effective; it solves only 30% of instances overall (whereas with the unit-cost FF heuristic the planner solves all but one problem). LAMA, and the two iterated variants of GBFS with the unit-cost FF heuristic, achieve more consistent quality, while still solving most instances (90% for LAMA). Even so, the difference between the lowest cost plan found by any of these planners and the smallest known edit distance is of the same magnitude as the edit distance itself, which is clearly too much. Finally, there is an enourmous gap between the smallest known edit distance and the lower bound. This is a significant problem, since it means there is no way to tell how close to the true minimum these distances are. In other words, we cannot know for sure if the smallest known edit distances equal the true minimum, or if they are just noise.

Conclusions

Computing minimum, or at least consistent, genome edit edistances is clearly a challenging problem for current

domain-independent planners. The "obvious" formulation (relational single-step) of the problem requires advanced features of PDDL (e.g., conditional effects with derived predicates in effect conditions), which are not dealt with effectively even by the very few planners that support them. Furthermore, all but very small instances this formulation cannot be grounded, which rules out all modern domain-independent planners. However, these problems can be overcome by formulating the problem in a way that, while perhaps appearing "unnatural", is better suited to the planners. This formulation needs only STRIPS with action costs.

Finding plans of consistent (high) quality, while scaling up to large problem instances, remains a challenge. It is interesting to note that the time taken by GBFS with the unit-cost FF heuristic to solve even the largest problems is no more than around 15 minutes. How to make use of that efficiency to find high-quality plans is an important question for future planning research, not only for its application to the genome edit distance problem. (The simple "randomise and repeat" scheme tried here is not a good enough answer.) Most important, however, is the lack of a sufficiently strong lower bound. As long as the gap between the lower bound and the smallest known distance is as large as the distance itself, we cannot be confident that edit distances computed by planners reflect the truth.

Finally, with these results in hand, we may ask: would biologists, interested in computing genome edit distances, want to use domain-independent planners? The promise of domain-independent planning in this application is its generality, providing a means to explore different sets of edit operations without the need to develop new, customised heuristics or special-purpose algorithms for each set. However, there are clearly hurdles to be overcome before this promise can be realised. Formulating the problem such that current planners can effectively solve it requires insight into the workings of the planners, and may even be more difficult for a non-specialist than developing a problem-specific search heuristic. Better methods of finding high-quality plans, and of finding lower bounds to provide assurance of the quality of those plans, are also needed.

Acknowledgements I thank Esra Erdem for kindly providing details about her formalisation of the genome rearrangement problem, and data on experiments with it. This work was supported by ARC project DP0985532.

References

Blanchette, M.; Kunisawa, T.; and Sankoff, D. 1996. Parametric genome rearrangement. *Gene* 172:11–17.

Blanchette, M.; Kunisawa, T.; and Sankoff, D. 1999. Gene order breakpoint evidence in animal mitochondrial phylogeny. *Journal of Molecular Evolution* 49:193–203.

Boore, J., and Brown, W. 1998. Big trees from little genomes: mitochondrial gene order as a phylogenetic tool. *Current Opinion in Genetics & Development* 8:668–674.

Cosner, M.; Jansen, R.; Moret, B.; Raubeson, L.; Wang, L.-S.; Warnow, T.; and Wyman, S. 2000. An empirical comparison of phylogenetic methods on chloroplast gene order data in campanulaceae. In *Comparative Genomics*, 99–121. Kluwer Acad. Publ., Montreal, Canada.

Erdem, E., and Tillier, E. 2005. Genome rearrangement and planning. In *Proc. 20th National Conference on AI (AAAI'05)*.

Hannenhalli, S., and Pevzner, P. 1995. Transforming cabbage into turnip (polynomial algorithm for sorting signed permutations by reversals). In *Proc. 27th ACM-SIAM Symposium on the Theory of Computing (STOC'95)*, 178–189.

Helmert, M.; Do, M.; and Refanidis, I. 2008. Changes in PDDL 3.1. `http://ipc.informatik.uni-freiburg.de/PddlExtension`.

Helmert, M. 2009. Concise finite-domain representation for PDDL planning tasks. *Artificial Intelligence* 173:503–535.

Lavrov, D.; Boore, J.; and Brown, W. 2002. Complete mtDNA sequences of two millipedes suggest a new model for mitochondrial gene rearrangements: Duplication and nonrandom loss. *Molecular Biology and Evolution* 19(2):163–169.

Nadeau, J., and Taylor, B. 1984. Lengths of chromosomal segments conserved since divergence of man and mouse. *Proc. National Academy of Sciences USA* 81:814–818.

Nebel, B. 2000. On the compilability and expressive power of propositional planning formalisms. *Journal of AI Research* 12:271–315.

Page, R., and Holmes, E. 1998. *Molecular Evolution: A Phylogenetic Approach*. Blackwell Science, Oxford.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of AI Research* 39:127–177.

Rintanen, J. 2011. Heuristics for planning with sat and expressive action definitions. In *Proc. 21st International Conference on Automated Planning and Scheduling (ICAPS'11)*.

Saitou, N., and Nei, N. 1987. The neighbour-joining method: A new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution* 4:406–425.

Sankoff, D.; Leduc, G.; Antoine, N.; Paquin, B.; Lang, B.; and Cedergren, R. 1992. Gene order comparisons for phylogenetic inference: Evolution of the mitochondrial genome. *Proc. National Academy of Sciences USA* 89:6575–6579.

Snel, B.; Huynen, M.; and Dutilh, B. 2005. Genome trees and the nature of genome evolution. *Annual Review of Microbiology* 59:191–209.

Thiebaux, S.; Hoffmann, J.; and Nebel, B. 2003. In defense of PDDL axioms. In *Proc. 18th International Conference on Artificial Intelligence (IJCAI'03)*, 961–968.

Uras, T., and Erdem, E. 2010. Genome rearrangement and planning: Revisited. In *Proc. 20th International Conference on Automated Planning and Scheduling (ICAPS'10)*, 250–253.

Temporal Planning for Co-Design of Host Scheduling and Workflow Allocation in Mobile Environments*

Qiang Lu[1], Yixin Chen[2], Mart Haitjema[2], Catalin Roman[2], Christopher Gill[2], Guoliang Chen[1]
[1]School of Computer Sci. & Tech., Univ. of Sci. & Tech. of China, Hefei, Anhui, 230026, China
[2]Department of Computer Sci. & Eng., Washington Univ. in St. Louis, St. Louis, MO, 63130, USA
rczx@mail.ustc.edu.cn, {chen, mah5, roman, cdgill}@cse.wustl.edu, glchen@ustc.edu.cn

Abstract

Workflows have been successfully applied to model collaborations with a well-defined structure, which has a common restriction that the network settings are stable. Recently, a new challenging domain, collaborations among groups of hosts in an ad hoc mobile network, attracts many interests. Some key features of this application domain, such as ad hoc interactions among hosts and high levels of mobility, introduce many challenges for designing a workflow management system. The main difficulty is to design an efficient planning algorithm that automatically schedules not only workflow allocation but also the actions (movement and communication) of hosts. Existing works only consider workflow allocation but do not specify how the hosts should act to achieve the plan in a mobile environment.

In this paper, we propose a framework that co-designs the host schedule and workflow allocation in a unified way. We transform the collaboration problem into a temporal planning model and then solve it using automated planners that can also minimize the total makespan of the plan in an anytime fashion. We integrate this framework into a workflow management system CiAN. The experimental results show that our approach significantly broadens the scope of previous works by removing the requirements of knowing host schedule. Our approach is also very efficient in finding high-quality solution plans with short makespans.

Introduction

Workflows have been successfully applied to model collaborations that have a well-defined structure, which has a common restriction that the network settings are stable. (Sen et al. 2007) presents an initial investigation into the possibility of using workflows in a challenging new domain - that of supporting arbitrary collaborations among groups of hosts in a mobile ad hoc network (MANET). This application domain shares several key features: ad hoc interactions among people, high levels of mobility, the need to respond to unexpected developments, the use of locally available resources, prescribed rules of operation, and specialized knowhow. Efforts toward using workflow in ad hoc wireless environments are relatively new. Workow allocation in MANETs has wide applications such as geological monitoring, emergence coordination, and robot communities.

Designing a workflow management system (WfMS) for this domain faces many challenges, as hosts may move, and service availability may depend upon which hosts are within communication range. These challenges make most existing workflow management algorithms inadequate since they do not consider the mobility and communication constraints in ad hoc settings.

A few research efforts toward this new domain have been carried out. (Sen et al. 2007) introduces a simple heuristic allocation algorithm which gives the tasks that are harder to satisfy higher priorities to be allocated and divides the allocation of the workflows into sub-problems recursively. A workflow management system for MANETs, CiAN, is introduced in (Sen, Roman, and Gill 2008) based on this allocation algorithm. (Haitjema et al. 2010) solves this workflow allocation problem by transforming it to a numeric temporal planning problem and calling SGPlan (Chen, Wah, and Hsu 2006) to find a feasible allocation.

All the above works have a major limitation. They all assume knowing the activity schedule of each host before using an allocation algorithm to assign each task to a suitable host. However, such an assumption is very restrictive since they fix the host schedule during allocation and hence limit the decision space of the workflow allocation algorithms. The scheduling of hosts has significant impacts to the feasibility and quality of workflow allocation, since it is a main source of freedom for coping with the communication, dependency, and geometrical constraints. In an ad hoc mobile environment, two hosts usually have a communication range (e.g. the bluetooth range), and they cannot exchange data unless they are within the communication range. Since the tasks have dependencies that require notification of completion and exchange of results, such communication limits greatly complicate the workflow allocation and make the problem much harder. Existing works separate the host scheduling problem from workflow allocation. They only solve the allocation problem and do not address how to precompute a feasible host schedule. In fact, we observe that it is necessary to consider the host schedule and workflow allocation simultaneously in a **co-design** approach, in order to design a complete algorithm that can always find a feasible solution if there exists one.

*This work is supported by China Scholarship Council, NSF grants IIS-0713109, CNS 1017701, and Microsoft Research New Faculty Fellowship.

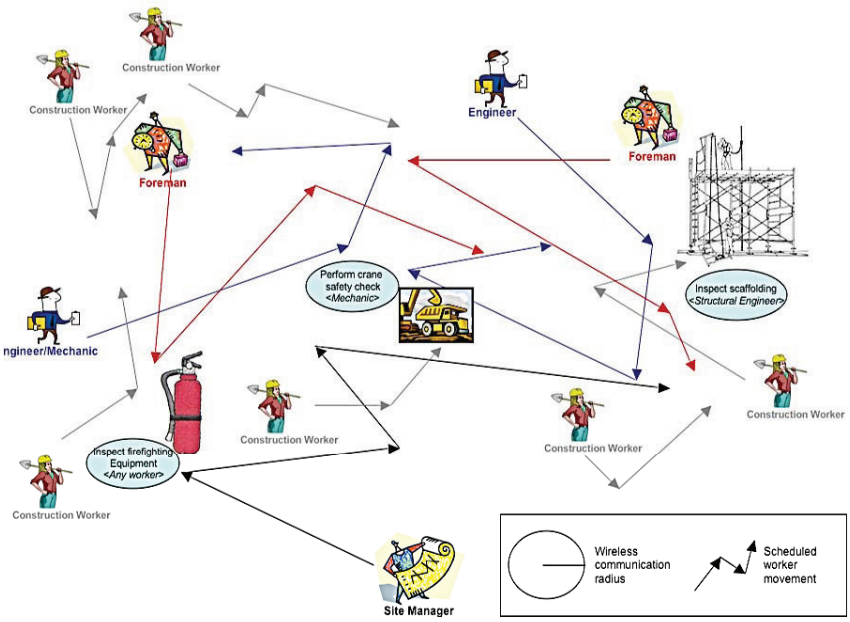


Figure 1: The construction site safety report problem.

In order to address these challenges for collaboration in MANETs, we propose an approach to co-design host scheduling and workflow allocation in a unified framework. Our approach automatically translates the collaboration-in-MANETs problem to a temporal planning problem, and calls a temporal planner to find suitable host schedule and workflow allocation for WfMS. Our model can handle dynamic initial and goal states to support online insertion of new tasks. Moreover, it provides the ability to optimize the total makespan in an anytime fashion by leveraging on the advance of AI planning research.

System Model: CiAN on MANETs

We develop our planning approach on the CiAN architecture (Sen, Roman, and Gill 2008), although in principle our approach can be generalized to other workflow systems for ad hoc mobile environments. A workflow engine provides the build time environment for process modeling (definition, design, and evolution) and the runtime environment for activating, managing, and executing workflow processes (Myers and Berry 1999). Process modeling will build a library of process templates which usually integrate information flow requirements, activity decomposition, and communication constraints. In the runtime environment, it usually contains three phases:

- Process Selection: The engine will respond to some new requests by selecting and instantiating suitable process templates from the library.
- Task Allocation: Once the processes are instantiated, the engine will assign tasks to suitable processing entities according to some predefined rules or algorithms. This task allocation can be viewed as a scheduling problem, which can also be solved by automated planners.
- Enactment Control, Execution Monitoring, and Failure Recovery: The engine will maintain all the knowledge and internal control data to identify the state of each activity, transition conditions, connections among processes, and performance metrics.

In most of the existing workflow engines, they take a centralized role in coordinating the operation of processing entities, as the hosts report to the central host during execution and wait for instructions. However, designing a workflow management system targeted to MANETs faces a new challenge: coordination among the various participants becomes more complex due to the dynamic topology of the MANET which often allows only transient connectivity among hosts. To bring workflows to this dynamic type of mobile networks, CiAN is designed as a lightweight and choregraphed engine that facilitates the workflow execution in MANETs (Sen, Roman, and Gill 2008). Specifically, it uses a publish-subscribe-like protocol that takes results from a task and delivers them to the host responsible for executing the immediately succeeding tasks without going through a central coordinating entity. Note that CiAN allows decentralized information exchange during execution, but still requires centralized workflow allocation before execution.

Task Allocation Problems in CiAN

We use an example on construction site coordination to help describe the task allocation problem in CiAN. Our planning-based approach is general for workflow allocation in MANETs and not limited to this example.

Example 1 *Consider a group of construction workers moving around a large construction site like the one shown in Figure 1. All the workers are equipped with mobile devices (e.g. PDAs and smart phones) and are moving around the*

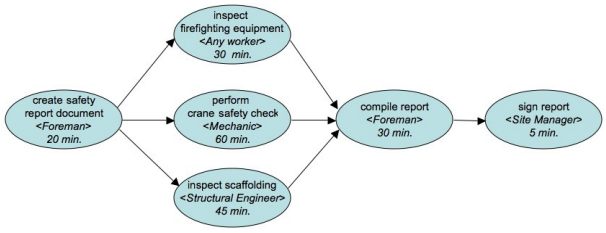


Figure 2: The workflow of the construction site safety report problem.

site working according to their individual schedules, which may be known to the management.

Now imagine that the management wants to perform a safety inspection check ad hoc and demands that a safety inspection report to be compiled. The various steps in creating the report are shown as a workflow in Figure 2.

In an ad hoc mobile environment, tasks that make up the workflow can only be done in specific locations and may require certain qualifications in order to be performed. For example, only a structural engineer working near the scaffolding can perform the inspect scaffolding task. A further complication is that after a task is executed, the results will need to be communicated to the person(s) executing the subsequent task(s) in the workflow. This transfer of results is normally done over a network. But, since the construction site is large, there may be no network infrastructure and we assume two people can only communicate when their mobile devices are within the communication range of one another. This means that even if a worker can execute a task, we must consider whether or not the worker will be able to pass the result onto the person we have chosen to perform the subsequent task. All of these constraints can make the process of choosing a worker for a given task a non-trivial problem. In fact, deciding whether or not there is an allocation of tasks to workers such that the workflow could possibly complete in this manner is NP-hard (Haitjema et al. 2010).

In the CiAN model, each task is associated with the following features:

- Locations: the locations where the task can be performed. Specified as a coordinate pair (x,y). Note that a task can be performed at one or more locations.
- Duration: the time required to execute the task.
- Hosts: the hosts who can perform the task. Note that a task can be performed by one or more hosts.
- Qualifications: a list of the qualifications necessary to perform the task. It also includes the task dependencies, a list of the tasks in the workflow that must be completed before the task to be executed. A host must know that all dependent tasks have been finished before executing the task.
- Status: whether the task has been performed.

We also have a set of workers which we will call hosts. CiAN requires to know the exact schedule of each hosts before performing task allocation, specified by the following features:

- Location: the location where the host is at for a certain time.
- Move speed: the speed of the host moving from one location to another.

The goal of the problem is to determine if there is a feasible allocation. A feasible allocation is a mapping of tasks to hosts such that each host can execute all its assigned tasks. In order for a host to execute a task t, the host must have the qualifications required by t, be in the location for the duration specified by t, and must be able to receive the results from the host(s) executing the dependencies of t. Note that the requirement that a host receives results for all the dependencies before executing its task ensures that the workflow is completed in the order specified by the workflows ordering constraints.

Limitations

CiAN requires to know the schedule of each host before performing task allocation (Sen, Roman, and Gill 2008; Haitjema et al. 2010). Each schedule entry contains a start time, location at the start time, end time, and location at the end time, which indicates when a host will be at certain location. In ad hoc mobile networks, locations of hosts may dynamically change and two hosts must meet up within a range before exchanging data when one host requires results from the other. Fixing the schedules of all hosts, CiAN can pre-compute the exact locations of moving hosts and the availability of communication between hosts for any given time. Such a strong assumption greatly simplifies the planning problem.

However, in real-world applications, a host may arrange its schedule according to the tasks it will execute. Thus, the schedules of hosts are usually unknown when performing task allocation in WfMS. Moreover, considering the host schedule and workflow allocation together gives a larger decision space which may leads to more preferable (e.g. shorter) plans. The co-design of both host schedule and workflow allocation, while considering the dependency, communication, and temporal constraints, is beyond the capability of the allocation algorithm in CiAN, or any other existing workflow algorithms we know of.

Temporal Planning for Co-Design

To overcome the limitations of the existing allocation algorithms, we formulate the host/workflow co-design problem into a temporal planning problem in PDDL and use state-of-the-art temporal planners to solve it. In our approach, we discard the assumption of knowing the host schedule *a priori* and aim at finding task allocation and host schedule at the same time.

Using the Planning Domain Definition Language (PDDL) 2.2 (Edelkamp and Hoffmann 2003), we define the temporal problem for the co-design problem as follows.

Objects. We define four objects: **host, task, location**, and **token**. Each host can execute certain tasks at certain locations. Each token is associated with a task to indicate that

the task is finished. A token is consequently also an edge between two tasks representing the dependency. A host must have all the relevant tokens for a task (one for each dependency) before executing it.

Predicates. Based on the definition of objects, we define four kinds of predicates representing the status of hosts and tasks.

- **at ?h - host ?l - location**: a host h is at location l.
- **free ?h - host**: a host h is free, which means it can execute a task, move to other locations, or communicate with another host.
- **done ?t - task**: a task t is finished.
- **has-token ?h - host ?t - token**: a host h has token t, which means the host t knows that task t has been executed. Note that we use the same name for task t and the token associated with t.

Durative Actions. We consider three kinds of durative actions: an **execute** action of a host to execute a task at a given location by a given host, a **move** action of a host between two locations, and a **communicate** action between two hosts. These three kinds of durative actions are defined as follows.

- **execute ?h - host ?l - location ?t - task**: 1) Duration: a positive rational number indicating the executing time of task t. 2) Preconditions: (at start (at h l)), (at start (free h)), (over all (at h l)), and (at start (has-token h t')) for all dependent tasks t' of the task t. Since PDDL2.2 is able to describe numeric resources, we also can describe some other qualifications, such as resource requirements. 3) Effects: (at start (not (free h))), (at end (free h)), (at end (done t)), and (at end (has-token h t)). The last two effects represent that task t is finished and host h has token t.
- **move ?h - host ?l_1 ?l_2 - location**: 1) Duration: a positive rational number indicating the time for host h to move from l_1 to l_2. It equals to the distance between l_1 and l_2 divided by the speed of the host. Note that all the move actions together encode the map information of the environment. 2) Precondition: (at start (at h l_1)). 3) Effects: (at start (not at h l_1)) and (at end (at h l_2)).
- **communicate ?h_1 h_2 - host ?l_1 ?l_2 - location ?t - task**: 1) Duration: a positive rational number indicating the time for transferring a message (token t). 2) Preconditions: (at start (at h_1 l_1)), (at start (at h_2 l_2)), (at start (free h_1)), (at start (free h_2)), (at start (has-token h_1 t)), (over all (at h_1 l_1)), and (over all (at h_2 l_2)). 3) Effects: (at start (not (free h_1))), (at start (not (free h_2))), (at end (free h_1)), (at end (free h_2)), and (at end (has-token h_2 t)). The last predicate represents that the host h_2 has the token t after communicating with h_1.

The precondition (over all (at h l)) indicates that the the host h cannot move to other locations when executing a task or communicating with another host. The precondition (at start (free h)) and effect (at start (not (free h))) guarantee that the host h cannot perform two actions at the same time.

Note that since dependent tasks are different in different *execute* actions, we cannot define all *execute* actions in an ungrounded way. Therefore, all actions are grounded in the domain definition of PDDL.

Initial State. The initial state of the problem includes predicates: 1) (free ?h - host) and (at ?h -host ?l - location) which indicate that host h is free now and at location l. 2) (at n (free ?h - host)) and (at ?h - host ?l - location) which indicate that host h will be free at time n (a positive rational number) and at location l then. This timed initial literals, (at n (free ?h - host)), is a feature of PDDL2.2 (Edelkamp and Hoffmann 2003) which represents facts that become true or false at certain time points.

Note that we use this feature to support dynamic planning. During the execution of a plan, when a set of new task requirements come in, a host may be executing some tasks currently and will be free in the future time n. If the host is required for a task, we need to add the host as an object in the planning problem. Thus we use the timed initial predicates to represent the initial status of such hosts. By exploiting the timed initial literals in PDDL2.2 planning, our approach can support dynamic planning in response to new tasks during execution.

Goal State. The goal state includes predicates (done ?t - task) for all required tasks t. Again, our approach can support dynamic planning. When new tasks are added during execution, we generate a new planning problem with the dynamic initial state discussed above and goal state using these new tasks, and then call a planner to solve it.

Based on the above PDDL model, any solution plan found by a temporal planner will give a co-design solution, which specifies the workflow allocation as well as a schedule of movement and communication for each host.

The resulting temporal problem is temporally simple without required concurrency (Cushing et al. 2007). A temporal problem has required concurrency (called temporally expressive) when, in any solution, there exist two actions a_1 and a_2 such that: 1) a_1 has two effects (at start f) and (at end (not f)) (which means f is an interval add-effect), and 2) a_2 has a precondition (at start f). These two conditions require a_1 and a_2 to be executed concurrently. Clearly, in our PDDL2.2 model, no two actions satisfy these two conditions. Note that temporally simple problems are typically more tractable than temporally expressive ones. Hence, our formulation can be efficiently solved using existing temporally simple planners.

Note that although the problem is temporally simple, it is essential to exploit the durative natures and concurrency of actions in our model in order to generate efficient plans with short makespans. In real-world applications, the temporal feature is very important because users usually want to finish all tasks as soon as possible. In our implementation, we use an anytime temporal planner, TFD (Eyerich, Mattmüller, and Röger 2009), to optimize the makespan.

We show three grounded actions of Example 1 in Figure 3.

The optimal solution found by Temporal Fast-Downward (TFD) (Eyerich, Mattmüller, and Röger 2009) is shown as follows. The first column is the scheduled time of each action and the last column is the action duration. The

```
(execute Foreman-0 crane-zone compile-report
  :duration (= ?duration 30)
  :condition (and
    (at start (at Foreman-0 crane-zone))
    (at start (free Foreman-0))
    (at start (has-token Foreman-0 inspect-firefighting-equipment))
    (at start (has-token Foreman-0 perform-crane-safety-check))
    (at start (has-token Foreman-0 inspect-scaffolding))
    (over all (at Foreman-0 crane-zone))
  )
  :effect (and
    (at start (not (free Foreman-0)))
    (at end (has-token Foreman-0 compile-report))
    (at end (done compile-report))
    (at end (free Foreman-0))
  )
 )


(communicate Foreman-0 crane-zone Site-Manager crane-zone compile-report
  :duration (= ?duration 1)
  :condition (and
    (at start (at Foreman-0 crane-zone))
    (at start (at Site-Manager crane-zone))
    (at start (free Foreman-0))
    (at start (free Site-Manager))
    (at start (has-token Foreman-0 compile-report))
    (over all (at Foreman-0 crane-zone))
    (over all (at Site-Manager crane-zone))
  )
  :effect (and
    (at start (not (free Foreman-0)))
    (at start (not (free Site-Manager)))
    (at end (free Foreman-0))
    (at end (free Site-Manager))
    (at end (has-token Site-Manager compile-report))
  )
 )

(move Foreman-0 crane-zone firefighting-zone
  :duration (= ?duration 2)
  :condition (at start (at Foreman-0 crane-zone))
  :effect (and
    (at start (not (at Foreman-0 crane-zone)))
    (at end (at Foreman-0 firefighting-zone))
  )
 )
```

Figure 3: Three grounded actions of Example 1.

total time (makespan) of the solution is 120.18 seconds.

– 0.01 (move site-manager init-loc crane-zone) [2]

– 0.02 (move foreman-1 init-loc crane-zone) [2]

– 0.03 (move construction-worker-4 init-loc firefighting-zone) [3]

– 0.04 (move engineer init-loc scaffloding-zone) [3]

– 0.05 (move mechanic init-loc crane-zone) [2]

– 2.06 (execute foreman-1 crane-zone create-safety-report-document) [20]

– 22.07 (communicate foreman-1 crane-zone mechanic crane-zone create-safety-report-document) [1]

– 23.08 (execute mechanic crane-zone perform-crane-safety-check) [60]

– 23.09 (communicate foreman-1 crane-zone construction-worker-4 firefighting-zone create-safety-report-document) [1]

– 24.10 (execute construction-worker-4 firefighting-zone inspect-firefighting-equipment) [30]

– 24.11 (communicate foreman-1 crane-zone engineer scaffloding-zone create-safety-report-document) [1]

– 25.12 (execute engineer scaffloding-zone inspect-scaffolding) [45]

– 54.13 (communicate construction-worker-4 firefighting-zone foreman-1 crane-zone inspect-firefighting-equipment) [1]

– 70.14 (communicate engineer scaffloding-zone foreman-1 crane-zone inspect-scaffolding) [1]

– 83.15 (communicate mechanic crane-zone foreman-1 crane-zone perform-crane-safety-check) [1]

– 84.16 (execute foreman-1 crane-zone compile-report) [30]

– 114.17 (communicate foreman-1 crane-zone site-manager crane-zone compile-report) [1]

– 115.18 (execute site-manager crane-zone sign-report) [5]

Task execution

Once the planning process is completed, the execution of the workflow begins. When an agent is assigned an action, it adds an entry to its schedule that contains all the necessary information to execute the appropriate service as directed by the manager. The agent is free to roam, but is responsible for completing assigned tasks, including moving to certain locations, executing tasks, and exchanging task execution information. The execution phase of this workflow is in a decentralized and distributed manner.

The flexibility of such a dynamic domain is much higher than normal workflow problems, due to the high variability of agent speeds and task completion probability. The change of speed may cause an agent not able to arrive at a required location in time for executing tasks or communicating with another agent. A failed execution of a task may break the dependencies of other tasks and cause the whole plan to crash. Thus, the ability to handle these execution exceptions is very important for the workflow management system.

In our system model, we can handle these exceptions since our planning algorithm, as we discussed in this paper before, can support dynamic planning by exploiting the timed initial literals. During the execution of a plan, when an agent fails to execute an action, it will first try to re-do it if there is enough time before the start time of the next action in its schedule. The replanning process will be triggered if the re-do fails. We first collect the current state of free agents and timed future state of working agents if necessary to generate the new initial state, and the unfinished goal tasks to generate the new goal state. Then we generate a new planning problem with the dynamic timed initial state and goal state, and then call a planner to solve it. As we will show in our experimental results, we can solve the planning problem efficiently (under 10 seconds for the largest problem with 30 hosts). Hence, it is feasible to plan dynamically during execution.

Related Work

AI Techniques for Workflow Management

An overview of early uses of AI techniques in workflow engines is presented in (Myers and Berry 1999). The paper overviews three major areas: 1) reactive control systems providing adaptive process management, 2) AI scheduling providing adaptive resource allocation, and 3) AI planning providing process synthesis and repair (with a focus on replanning).

There are several other work on applying AI planning techniques to WfMS. (Moreno and Kearney 2002) describes an integration of AI planning techniques with an existing workflow management system. It uses AI planning to automatically generate a sequence of instantiated activities. (Schuschel and Weske 2003) outlines a framework for an integrated planning and coordination system, which uses AI planning to support business processes. (Shi, Yang, and Sun 2011) presents a workflow management system with dynamic goal tasks, using AI planning to solve these goal driven workflow planning problems. However, none of these works considers ad hoc mobile environments and the planning of host activities as this paper does.

WfMS with Temporal Constraints

Modeling temporal constraints in WfMS is first proposed in (Marjanovic and Orlowska 1999; Eder, Panagos, and Rabinovich 1999). (Marjanovic and Orlowska 1999) proposes a framework for time modeling in production workflows. Relevant temporal constraints are presented, and rules for their verification are defined. Furthermore, to enable visualization of some temporal constraints, a concept of "duration space" is introduced. (Eder, Panagos, and Rabinovich 1999) proposes modeling primitives for expressing temporal constraints between activities and binding activity executions to certain fixed dates. It presents techniques for checking satisfiability of temporal constraints, and enforcing these constraints at run-time. The techniques for temporal constraint management are based on the *timed activity graph*.

(Son and Kim 2001) proposes a scheme to maximize the number of workflow instances satisfying given deadlines. It develops a method to determine the minimum number of servers (MNS) for any critical activities, an activity that should be finished without delay for a given input arrival rate. (De Maria, Montanari, and Zantoni 2005) proposes to use the finite timed automata as a tool to specify timed workflow schemas and to check their consistency. Temporal constraints are often set when complex e-science processes are modeled as scientific workflow specifications. (Chen and Yang 2010) systematically investigates how to localize a group of fine-grained temporal constraints so that temporal violations can be identified locally for better cost effectiveness. Most of these works are designed to derive or check certain properties of a temporally constrained system. However, they cannot be directly used to solve workflow allocation problems, nor the more complex co-design problem. An interesting future work is to incorporate these analysis as heuristic guidance and pruning conditions to further improve our planning approach.

Experimental Results

We evaluate our method by designing a simulator. In our implementation, we use two state-of-the-art temporal planners, SGPlan (Chen, Wah, and Hsu 2006) and TFD (Eyerich, Mattmüller, and Röger 2009), to solve the compiled temporal planning problem in PDDL2.2. We generate a series of random problems and measure the time taken to find a solution plan and the temporal makespan. The main parameters of generated problems include:

- n_h: the number of hosts.
- n_t: the number of tasks.
- n_c: the number of dependencies for each task.
- max_x, max_y: location range ($0 \leq x \leq max\_x$, $0 \leq y \leq max\_y$) that hosts may work at. We set $max\_x = max\_y = 400m$.
- s: the moving speed of hosts. We set it to 1.7 m/s which is close to human walking speed.
- d: the duration of actions. The duration of *execute* is randomly chosen from [60, 300] seconds, the duration of *communicate* is set to 1 second, and the duration of *move* is set to the distance of two locations divided by speed ($dis(l1, l2)/s$).

n_h, n_t, and n_c are the main parameters deciding the complexity of the generated problems. In (Sen et al. 2007), the largest problem's parameters are $n\_h = 12$, $n\_t = 2{*}n\_h$, and the probability for each task to be assigned a task dependency equals to 0.3. Since the host schedule in (Sen et al. 2007)'s testing problem is randomly generated and a large number of task dependencies may lead to a low chance of finding a feasible task allocation, it sets a low probability of having task dependencies. In our evaluation, we generate a series of problem with $n\_h \in [1, 30]$, $n\_t = 2 * n\_h$, and n_c is randomly chosen from [0,3]. The dependencies between tasks are randomly generated with the guarantee that no cycle exists. We set this higher dependency probability than (Sen et al. 2007) because real-world applications usually require a large number of task dependencies as shown in Example 1.

All experiments are conducted on a workstation with 2.8GHz CPU and 2GB memory. The running time limit for each problem instance is set to 30 seconds. We set this relatively low time limit in order to ensure the practicability of our approach in real-world applications, where users prefer short planning time.

From experiments, we see that our approach can solve the co-design problems efficiently. Specifically, it requires no more than 2 seconds to find a solution on any problems with up to 30 hosts and 60 tasks, a problem size much larger than the largest problem previous work considered (12 hosts and 24 tasks) (Sen et al. 2007). The running time and makespans are shown in Figure 4.

Since the allocation algorithms in (Sen et al. 2007; Haitjema et al. 2010) and our approach are based on different assumptions, we cannot compare them directly using our testing problems. However, some quantitative comparisons can be made here. Comparing our approach against the allocation algorithm in (Sen et al. 2007), we found that our approach is still very efficient even though it is solving a much harder problem (which considers host scheduling under a higher degree of task dependencies). For example, for a problem with $n\_h = 12$ and $n\_t = 24$, the mean time to perform CiAN's allocation algorithm is about 1.2 seconds (Sen et al. 2007) while the solution time of our method is 0.25 second.

(Haitjema et al. 2010) usually cannot find a feasible allocation on our problems where $n\_h \geq 8$ as the randomly generated host schedule. This clearly shows the limitation of that approach and the need for co-design. Also, it spends more time to solve allocatable problems than our approach. For a problem with $n\_h = 4$ and $n\_t = 8$, using the same planner SGPlan, it takes 3.74 seconds to find an allocation while our approach takes only 0.03 second.

Another advantage of our method is that we can minimize the temporal makespan (the total time of finishing all tasks) by using TFD, an anytime planner which can optimize the makespan. Figure 4 shows the solution time and makespan of a feasible solution found by SGPlan and of the best solution found by TFD under 30 seconds. Obviously, SGPlan is faster and solves more problems than TFD under 30 seconds. On the other side, TFD usually finds better plans (with hundreds of seconds shorter makespans) at the cost of more planning time. For example, on the problem with $n\_h = 10$, TFD finds a plan with a makespan $2313s$ in 1.96 seconds while SGPlan found a solution with a makespan $4384s$ in 0.08 second. Considering the large saving of execution time, users are likely willing to spend a little more time on planning.

Conclusions and Future Work

Workflows have been extensively studied and applied to model collaboration in well-defined and stable networks. In this paper, we consider a new challenging environment of ad hoc mobile networks for workflow allocation. Existing

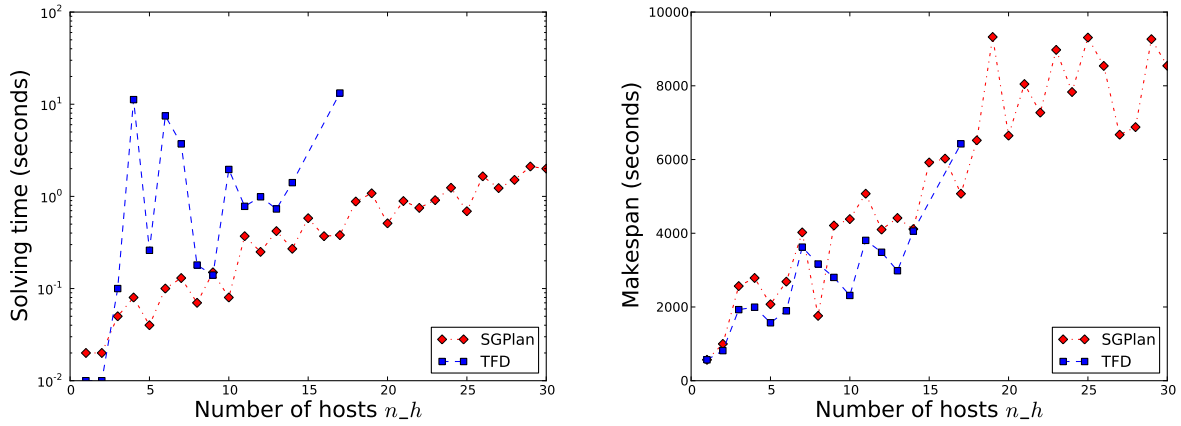


Figure 4: The running time and solution quality (in terms of makespan) of SGPlan and TFD. Note that $n\_t = 2 * n\_h$.

workflow engines for this domain can find a workflow allocation based on the restrictive assumption of knowing and fixing schedules of all hosts a priori. In this paper, we have presented a framework that co-designs the host schedule and workflow allocation in a unified way. We transform the collaboration problem into a temporal planning model and use automated planners to solve it efficiently. Our experimental results show that it is practical to use temporal planners to find feasible and even optimized schedule that coordinates hosts and workflows together under complex temporal, communication, and dependency constraints.

A practical workflow allocation algorithm on MANETs has many applications. Responding to catastrophes such as chemical spills, conducting geological surveys of remote areas, and even managing a community of robots exploring hazardous areas are only a few examples of the many activities that would potentially benefit from this approach. We plan to explore such real applications in our future work.

References

Chen, J., and Yang, Y. 2010. Localising temporal constraints in scientific workflows. *Journal of Computer and System Sciences* 76(6):464 – 474.

Chen, Y.; Wah, B. W.; and Hsu, C.-W. 2006. Temporal planning using subgoal partitioning and resolution in sgplan. *Journal of Artificial Intelligence Research* 26:323–369.

Cushing, W.; Kambhampati, S.; Mausam; and Weld, D. S. 2007. When is temporal planning really temporal? In *Proc. of IJCAI*.

De Maria, E.; Montanari, A.; and Zantoni, M. 2005. Checking workflow schemas with time constraints using timed automata. In *On the Move to Meaningful Internet Systems 2005: OTM Workshops*, volume 3762, 1–2.

Edelkamp, S., and Hoffmann, J. 2003. Pddl2.2l the language for the classical part of the 4th international planning competition. *Technical Report N. 194, ALbert Ludwigs University Institue for Informatik, Freiburg, Germany*.

Eder, J.; Panagos, E.; and Rabinovich, M. 1999. Time constrains in workflow systems. In *Proc. of CAiSE*, 286–300.

Eyerich, P.; Mattmüller, R.; and Röger, G. 2009. Using the context-enhanced additive heuristic for temporal and numeric planning. In *ICAPS*.

Haitjema, M.; Chen, Y.; Roman, C.; and Gill, C. 2010. Automated planning: Workflow allocation in a mobile setting. *http://www.cse.wustl.edu/ chen/workflow/*.

Marjanovic, O., and Orlowska, M. E. 1999. On modeling and verification of temporal constraints in production workflows. *Knowl. Inf. Syst.* 157–192.

Moreno, M. D. R., and Kearney, P. 2002. Integrating AI planning techniques with workflow management system. *Knowledge-Based Systems* 15(5-6):285 – 291.

Myers, K. L., and Berry, P. M. 1999. At the boundary of workflow and AI. In *Proc. of AAAI*.

Schuschel, H., and Weske, M. 2003. Integrated workflow planning and coordination. In *In 14th International Conference on Database and Expert Systems Applications*, 771–781. Springer-Verlag.

Sen, R.; Hackmann, G.; Haitjema, M.; Roman, G.-C.; and Gill, C. D. 2007. Coordinating workflow allocation and execution in mobile environments. In *COORDINATION*, 249–267.

Sen, R.; Roman, G.-C.; and Gill, C. D. 2008. Cian: A workflow engine for manets. In *COORDINATION*, 280–295.

Shi, Y.; Yang, M.; and Sun, R. 2011. Goal-driven workflow generation based on AI planning. *Computer and Computing Technologies in Agriculture IV* 346:367–374.

Son, J. H., and Kim, M. H. 2001. Improving the performance of time-constrained workflow processing. *Journal of Systems and Software* 58(3):211 – 219.

The Emergency Landing Planner Experiment

Nicolas Meuleau[*] and **Christian Neukom**[*] and **Christian Plaunt** and **David E. Smith** and **Tristan Smith**[†]
Intelligent Systems Division
NASA Ames Research Center
Moffet Field, California 94035-1000
{nicolas.f.meuleau, christian.neukom, christian.j.plaunt, david.smith, tristan.b.smith}@nasa.gov

Abstract

In previous work, we described an Emergency Landing Planner (ELP) designed to assist pilots in choosing the best emergency landing site when damage or failures occur in an aircraft. In this paper, we briefly describe the system, but focus on the integration of this system into the cockpit of a 6 DOF full-motion simulator and a study designed to evaluate the ELP. We discuss the results of this study, the lessons learned, and some of the issues involved in advancing this work further.

In a previous paper (Meuleau et al. 2009b), we described an Emergency Landing Planner (ELP) designed to assist pilots in choosing the best emergency landing site when damage occurs to an aircraft. In 2010, we integrated our planning software into the cockpit of a 6 DOF full-motion simulator for 757/767 category transport aircraft, and performed experiments to evaluate the software using crews of professional airline pilots. In this paper we briefly review the Emergency Landing Planner (ELP), but focus on three topics:

- Integration of the software into the aircraft avionics
- Design and results of an experiment to evaluate the system
- Challenges to further advancing and fielding the technology

1. The Emergency Landing Planner

Figure 1 illustrates the type of scenario that the ELP addresses. When damage or failures occur in an aircraft an adaptive controller takes over to help stabilize and control the aircraft. The ELP then provides the pilot with a ranked set of possible emergency landing sites. Fundamentally, the ELP is solving a 3D path planning problem with dynamics. It does this by constructing a probabilistic roadmap of points and edges that includes the current position of the aircraft and an approach point to every possible runway within a viable range. (This may cover hundreds of airports for an aircraft at high altitude.) A sophisticated model of risk is used to assess the probability of success for each edge in the roadmap. This model of risk takes into account:

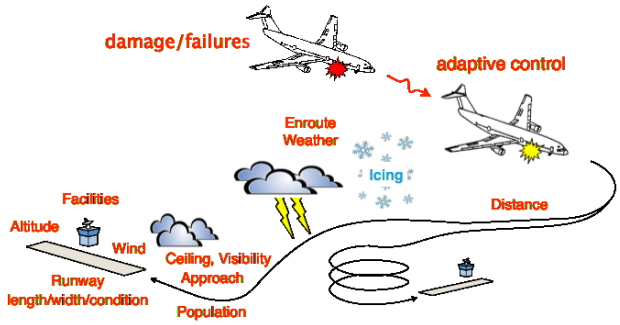


Figure 1: Basic Scenario

- Control capabilities of the (damaged) aircraft
- Weather conditions in the area (e.g. thunderstorms, turbulence, icing)
- Ceiling, visibility and winds at each possible landing site
- Instrument approaches available at the site (if any)
- Characteristics of the landing site (runway length, width, condition)
- Emergency facilities at the site (fire, medical)
- Danger to population along the approach path

A^* search is used to search the roadmap to find the best options. The heuristic used to guide A^* is a combination of the risk associated with flying the remaining (Euclidean) distance to each runway, and the risk associated with approach and landing at that runway.

Currently, the ELP only considers officially recognized airports and runways (large and small). However, there is no fundamental reason that additional sites could not be considered, including fields, highways, and waterways. Such sites should probably not be considered unless the airport options are exhausted or are too risky. The ELP makes two additional assumptions:

1. Real time weather information is available to the aircraft
2. The flight envelope for the (damaged) aircraft is known[1].

[*] Stinger Ghaffarian Technologies
[†] Mission Critical Technologies

[1] For our purposes, the flight envelope of an aircraft is the four dimensional space of airspeeds, bank angles, vertical speeds, and altitudes in which the aircraft can operate.

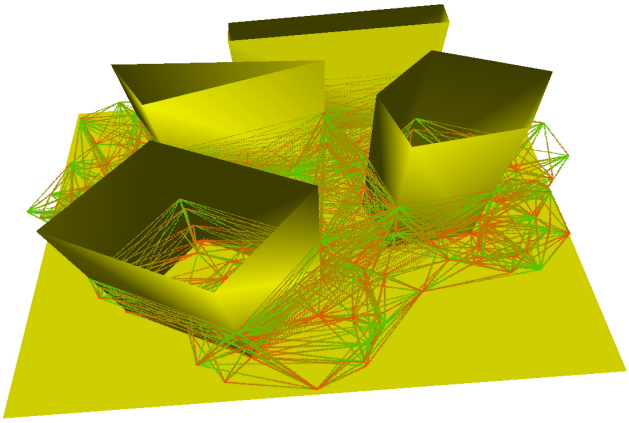


Figure 2: An example roadmap for an ELP scenario. The vertical polygons are areas of thunderstorm or other weather activity. Terrain obstacles (lower) are not shown.

The first of these assumptions is quite reasonable, given the availability of satellite weather services and internet connectivity for large aircraft. The second assumption is more optimistic. We will discuss this more in Section 5. The flight envelope does play a key role in the assessment of risk for various options. For example, if a damaged aircraft must maintain a higher airspeed than normal, additional runway length is needed, and finding a runway with a strong headwind is important to lowering ground speed at touchdown. Similarly, if the aircraft has limited ability to bank to the right, a right crosswind or gusty conditions will be problematic, as will paths that require sharp turns to the right.

The performance of the ELP is largely a function of the number of points and edges in the roadmap. Currently, we generate 1000 points and connect them to their 100 nearest neighbors, which results in a roadmap with 100,000 edges. The A^* search typically expands about 20 percent of those edges for the scenarios we considered. With this sized roadmap, the ELP produces an ordered list of options for the pilot in under 6 seconds. This list can therefore be refreshed and updated as often as desired, to account for the aircraft movement, weather updates, or additional failures.

Our experience has been that paths generated from probabilistic roadmaps of this density can be far from optimal, and just don't look very good when displayed. This problem can be addressed by dramatically increasing the density of points and edges, but this approach also significantly increases search time. The more practical solution is to use local search to shorten and smooth paths. We do this local search by constructing a second roadmap consisting only of points along the path just found, creating a dense network of edges among those points, and re-running A^* on this reduced graph. The resulting paths are shorter, smoother, and seem more natural when displayed.

More details about the risk model, the path planning, and the local search can be found in (Meuleau et al. 2009b; 2009a; 2011)

Figure 3: The Advanced Concepts Flight Simulator (ACFS).

Figure 4: The cockpit of the ACFS.

2. Integration

Figures 3 and 4 show the Advanced Concepts Flight Simulator (ACFS) at NASA Ames Research Center. The simulator is representative of modern glass cockpit twin engine commercial transport aircraft such as the Boeing 757, 767, and Airbus A320. Unlike most large commercial flight simulators, the code of this simulator has been "exposed" to allow for experimentation with adaptive control software, damage models, and experimental pilot aids and displays.

In normal operations, pilots view, enter, and modify destination, route and approach information using a pair of keypads and displays (CDUs) located just above, and on either side of the throttles (Figure 5). Information entered on a CDU is communicated to the aircraft's Flight Management System (FMS), which interfaces with the autopilot and with the various displays in the cockpit. When route information

is entered on a CDU, the route shows up as a dashed white line on the pilot and co-pilots Navigation Displays (Figure 6). Once executed, the previous route disappears, and the route becomes solid magenta.

Figure 5: A CDU showing the Departures/Arrivals page for Denver (KDEN) airport. The emergency prompt appears next to button 6R at the lower right.

To integrate the ELP into the aircraft cockpit, we needed to make it accessible through the CDUs and make it communicate route information to the FMS, so that the emergency routes would appear on the Navigation Displays. Furthermore, we wanted the pilots to be able to edit or change an emergency route just as they can with any other route. As a result, the ELP had to be fully integrated with the CDUs and the FMS. In addition, we wanted to make the style of the interface reasonably intuitive and consistent with existing CDU pages.

The ELP is accessed using button 6R from the Departure/Arrivals page (Figure 5). After a brief splash screen, a set of "Emergency Pages" is displayed, showing the options ordered from lowest to highest risk. Figure 7 shows the first of four emergency pages for a scenario. Each entry shows an airport, runway, runway length, distance, and direction (magnetic bearing). The smaller symbols below each entry indicate the principle risks associated with that option; for example, RL indicates runway length is an issue, and CE indicates that the cloud ceiling is close to the minimums for the best approach to that runway. To select an entry, the button to the left of the entry is pressed. In this case, the first entry has been selected by pressing button 1L, which causes the route for that option to show up as a dashed white line on the Navigation Displays, as shown in Figure 6. Pressing the EXEC key would cause the route to become the current route (solid magenta). The pilots can page through the options using the NEXT PAGE and PREV PAGE buttons as desired. To see more information about a particular option, the pilots

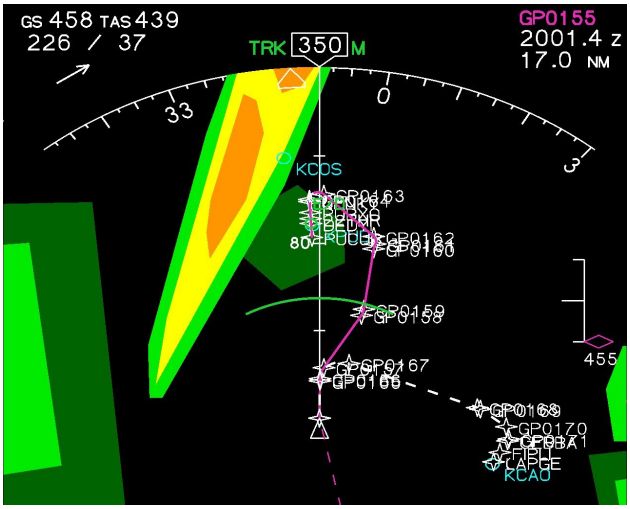


Figure 6: The Navigation Display showing both the current route (magenta) and the new route being considered (dashed white). Green, yellow, and orange areas indicate rain and thunderstorm activity.

can press the button to the right of the option, which brings up an airport information page showing runway information and the current weather at the airport (Figure 8).

The screen size and lack of color on the CDUs limited the amount of information we could convey for each option. With greater screen real estate, we could display winds, ceiling, and visibility information for each option. With color, we could show the severity of the principal risks. It seems likely that the displays and interfaces of future aircraft will not be quite so limited.

In a previous study comparing different adaptive controllers, Campbell et al 2010b; 2010a found that because of the assistance of the adaptive controller, pilots were unaware of when they were approaching the boundaries of the flight envelope. For example, pilots would slow the aircraft too much on final approach, not recognizing that in doing so they were nearing saturation of one or more control surfaces. On reaching saturation, the nose of the aircraft would suddenly drop, or the aircraft would roll inverted, causing them to lose control and crash. Since we were using an adaptive controller in this experiment, we therefore felt that it was essential to give the pilots some additional guidance on the limitations of the flight envelope. To do this, we added color bands to the primary flight display to indicate safe airspeeds, bank angles, and vertical speeds as shown in Figure 9. If the airspeed, bank angle and vertical speed remain in the green regions the aircraft can be readily controlled. However, as airspeed decays down into the yellow (not yet visible in the figure), the green regions for bank and climb rate shrink, ultimately to nothing. The size of the regions is dictated by the 4-dimensional model of the flight envelope, which varies depending on the damage or failure. The green regions for bank can be asymmetric, as is the case when there is damage to a wing or aileron.

Figure 7: The first of four emergency pages for a scenario. Each page can show up to five options.

Figure 8: An Airport Information page showing runways and current weather for KCAO.

3. The Experiment

To evaluate the ELP, we developed a set of scenarios involving different locations, different flight plans, different weather conditions and different damage models. There were three different locations, two flight plans for each location, two different weather severities, and three different damage models, for a total of 36 possibilities. The damage models were previously developed at NASA Langley through a combination of vortex lattice code and wind tunnel testing. The number of scenarios we could consider was necessarily limited by: the number of damage models available in the simulator; the number of realistic weather models we could develop; and the time required to test all the possible emergency flight plans and approaches that might be produced for each location.

In addition to the scenarios, we needed a baseline with which to compare the ELP. We therefore developed a simple aid for the pilots that just listed the nearby airports grouped by runway length. We also developed an intermediate aid that evaluated runways using our risk model, but did not consider en route weather, an did not generate a path for the pilots. The matrix of testing possibilities is summarized in Table 10.

To carry out the experiments, we employed 5 teams of professional pilots for two days each. All of these pilots were either current or recently retired airline pilots with experience in glass cockpit aircraft of the appropriate type. Each pilot team was briefed on the functioning of the ELP and baseline aids, and conducted several short training flights to ensure they were comfortable with the systems and handling of the simulator. The team was then subjected to 16 of the possible scenario/aid combinations. Each run began in cruise flight. Damage was introduced after 1-3 minutes, resulting in a master caution alarm in the cockpit, and indications of the failures on a display of the control surfaces shown in Figure 11. The pilots would then utilize the aid provided, chose an emergency landing site, and fly the aircraft until touchdown or loss of control. In some cases we also terminated the run after a decision was reached, because of time limitations. A typical run lasted about 35-40 minutes. At the end of the run, the pilots were asked to fill out a brief questionnaire about their decisions and about their assessment of the aid provided by the software. At the end of the two day period, the pilots were asked to fill out a longer questionnaire giving their overall impressions, criticisms, and suggestions for the emergency aids and interfaces.

During the runs, we observed the pilot performance from a control room with video screens of all major cockpit instruments (Figure 12). We collected multiple data streams including: video and audio from the cockpit; aircraft state at 30 Hz (location, altitude, airspeed, pitch, bank, control settings, etc.); keystrokes and display from the CDUs; and video of the Primary Flight Display (PFD) and Navigation Display (ND).

From the outset, we were aware that there were some serious limitations with the study:

1. The number of possible runs was limited because of both time and cost.
2. The number of different scenarios was limited because of the amount of data required and the difficulty of constructing the scenarios. As a result, the pilots could become familiar with scenarios and damage models as the study progressed.
3. The pilots could become fatigued, particularly later in the day.

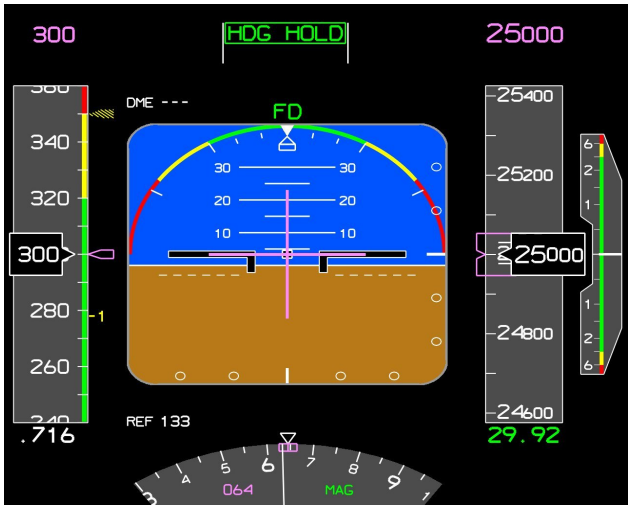


Figure 9: The Primary Flight Display (PFD) showing bank angle, pitch, airspeed, vertical speed, altitude and heading.

Pilot Aid	Damage	Weather	Location
Nearest Airports	Vertical Stabilizer	Mild: Overcast	Arizona: LAS → STL ABQ → SEA
Ranked Airports	Horizontal Stabilizer		Idaho: GEG → DEN GTF → SFO
ELP	Left Wing	Severe: Thunderstorms Low Ceilings	New Mexico: COS → SAT ABQ → MSP

Figure 10: Experiment test matrix.

The first of these limitations makes it difficult to draw statistically significant conclusions. In any study dealing with human subjects, there is a great deal of variability and randomness, so large sample sizes are needed. The cost of the simulator and pilots makes this impractical.

The second limitation, the limited number of scenarios, meant that the pilots became increasingly "contaminated" as the study progressed. We tried to minimize this by mixing up the different damage models, weather conditions, locations, and flight plans. However, the pilots clearly became more familiar with the terrain and airports in each region, and their skill with the different damage models improved over time. To attempt to average out these effects, we ordered the scenarios differently for the different crews.

The third limitation, pilot fatigue, seemed to show up primarily during the afternoon of the second day of testing. We noticed it because there were some cases where the pilots lost control of the aircraft and crashed during easier scenarios.

The combination of these limitations means that many of our results are anecdotal, are based on small sample sizes, or are the results of subjective feedback from the pilots.

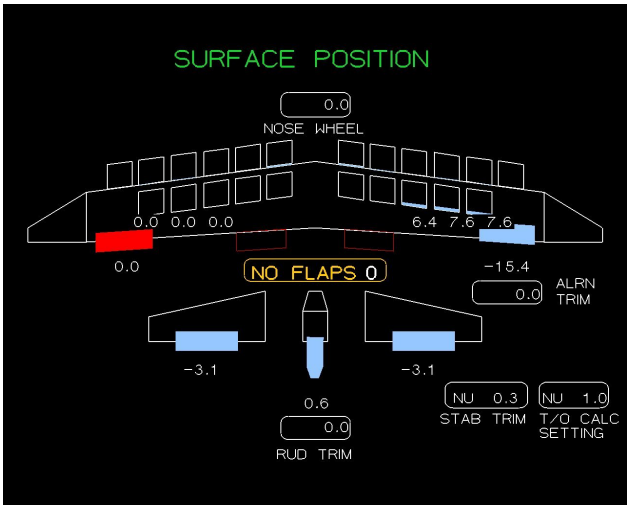


Figure 11: Surface position display showing status and deflection of control surfaces. In this case, the left wing is damaged and the left aileron has failed (red). As a result, the adaptive controller is using right up aileron (blue) and right spoilers (blue) to keep the aircraft from rolling left. When a control surface is saturated (at its limits) it turns yellow.

4. Results

Figure 13 is a trajectory plot showing the options considered by the pilots for one particular run. The red dot indicates the position of the aircraft at the time damage occurs. The black line is the aircraft's actual trajectory. Yellow lines indicate other options considered by the pilots, and the green line indicates the route provided by the ELP at the time they finally made a decision. As can be seen from the plot, the pilots made a tighter turn to the left (back towards the airport) than the ELP recommended. They also chose to intercept and get established on the final approach course further from the airport. In this run, damage was to the left wing and aileron, making it more difficult to turn right. In addition the weather was challenging, with larger airports in the area having low ceilings, poor visibility, or difficult crosswinds. In this case, KCAO runway 02 was the highest ranked option provided by the ELP, and it proved to be one of few choices for which pilots had any success in getting the aircraft on the ground. The blue path shows the route that would have been recommended if the pilots had made their decision instantly after the damage occurred. By the time the decision was made, it was no longer practical to make a right turn towards the chosen runway, given the control characteristics of the aircraft.

Figure 14 shows a run for a different scenario in the same general area. In this run, damage was to the horizontal stabilizer and elevator so turning was not difficult, but a higher airspeed had to be maintained to preserve enough airflow over the remaining elevator. In this case, pilots were tempted by long runways at lower ranked Colorado Springs (KCOS) and Cannon Air Force Base (KCVS), but winds and weather did not favor the available runways. They ended up choosing a more highly ranked option with a shorter runway (KCVN 04), because of the strong headwind straight down the run-

Figure 12: The ACFS control room.

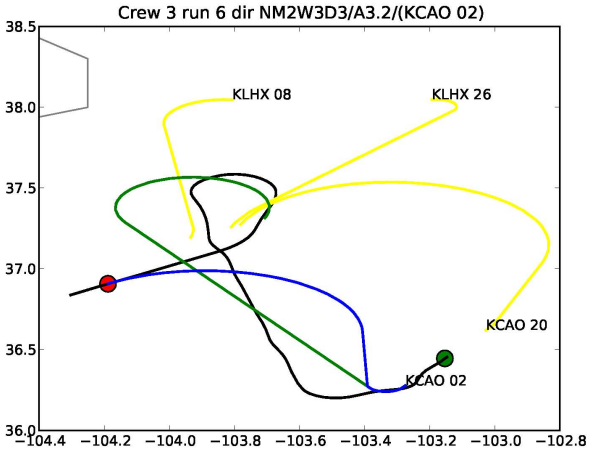


Figure 13: A trajectory plot for a left wing damage scenario.

way. In this case, KCAO 02 would have also been a good choice for the same reason.

In analyzing the data, we considered the time required to make a decision, and the pilots success rate as a function of the damage model, weather conditions, and location for each different emergency aid. Our initial hypothesis was that that the ELP would prove helpful to the pilots in cases where either the damage or weather was severe, but that the pilots would do just fine with the baseline emergency aid when the weather and damage were both benign. This hypothesis is only partially correct; weather severity was a factor, damage severity was not. For the scenarios involving mild weather conditions, the ELP does not seem to offer any objective improvement in pilot performance over the two simpler emergency aids. However, when the weather was poor, the ELP generally led to quicker decisions. The reason for this is that when the weather was mild, one of the nearby large airports with a long runway was usually the best choice. Pilots could

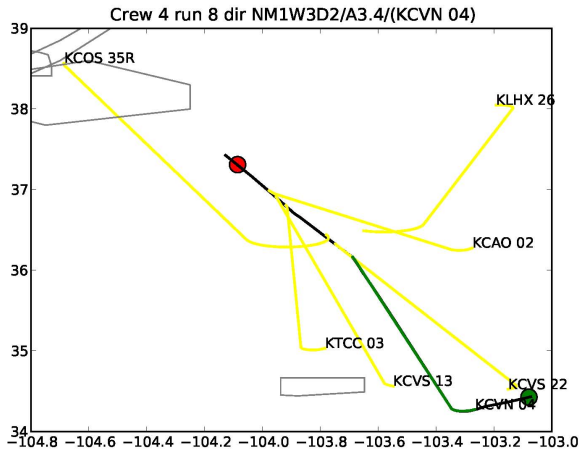


Figure 14: A second trajectory plot for a scenario with elevator damage.

find this choice easily enough using only the simpler emergency aid, and could choose the most appropriate runway by looking at the airport information page for that airport. In contrast, when the weather was poor the pilots would be forced to look at many different options before finding one with acceptable weather conditions. In a few particularly difficult cases, pilots took more than 20 minutes to reach a decision using the baseline emergency aid. With the ELP, decisions for these same scenarios were made in 4-5 minutes.

From the objective data, the paths constructed by the ELP did not seem to offer a significant advantage to the pilots either in terms of decision making time, or in terms of decision quality. When intervening weather was not an issue, a direct route to a point about 10 miles out on the final approach course was appropriate and was relatively easy for them to construct. Even when the weather was more severe, the pilots were able to construct their own paths, although it took them longer to do so. For this reason, pilots subjectively reported that it was a lot easier to have the assistance of the ELP in all cases, both because of the ranking of options, and because the route was constructed automatically and guaranteed terrain clearance. Thus, even for mild weather conditions, pilots preferred the ELP, and felt that it reduced workload.

In contrast to our hypothesis, the severity of the damage does not seem to be correlated with whether or not the ELP provides an advantage. For mild weather, but severe damage, there is no objective evidence that the ELP provides an advantage over the baseline emergency aid. Likewise, for severe weather, the advantage of the ELP over the baseline aid does not become any more pronounced if the damage is severe. We speculate that this is largely due to the adaptive controller – it does such a good job of stabilizing the aircraft that the pilots have time to investigate options and construct routes manually. Without the adaptive controller the workload is much higher since it is much more of a struggle to

maintain control of the aircraft. In this case the pilots would likely not have time to consider multiple options, or construct routes manually.

Overall, we were both surprised and thrilled with the enthusiastic response we received from the pilots, as illustrated by this quote:

> *As a Captain for the past twenty some years I've trained for emergencies frequently and the most difficult part is selecting and getting the aircraft on the ground when a immediate landing is called for. Your software program alleviates the uncertainty about finding a suitable landing site and also reduces workload so the Crew can concentrate on "flying" the aircraft.*

Although the technology was designed for next generation aircraft, several pilots indicated that they wanted to see this capability in their existing aircraft and suggested that it would be particularly valuable in time critical situations like cargo fires, loss of engine power, fuel or hydraulic fluid loss, or medical emergencies. Ironically, during our study a brand new UPS 747 crashed in Dubai as the result of a cargo fire. The pilots chose to return to the takeoff airport, although closer options were available. This proved to be a fatal mistake, as the smoke became so thick that the captain could no longer see his instruments.

Finally, we have several observations relevant to pilot training. The first is that there were significant differences in pilot performance. Two of the three damage scenarios required approach and landing at considerably higher speeds than the pilots were accustomed to. Those pilots with experience in high speed military aircraft did much better at this than those without that experience. Although rare, several actual damage incidents have had this characteristic, so regular simulator training in high speed landings would potentially be valuable.

A second observation is that many pilots preferred long runways with poor weather and wind conditions to shorter runways with better weather and winds. For example, several teams were seduced by the 13,000 ft runway at Colorado Springs, even though it was ranked low because the visibility was 1 mile in blowing snow, with a strong 70 degree crosswind. This proved fatal in almost every case, as the pilots were unable to maintain control during the approach and landing. In contrast the top ranked option only had a 7000 ft runway, but had good visibility and a strong headwind straight down the runway. The crews that chose this option were generally successful. This lends some credence to our risk model, and suggests that pilots should be trained to favor better wind and weather conditions over longer runways in emergency situations.

A third observation is that pilots did not seem to consider the possibility that handling characteristics might further deteriorate, or that there might be subsequent failures. As a result, they were content to fly relatively long distances with a severely damaged aircraft. In several recent accidents, deterioration of handling conditions, and/or subsequent failures have proven fatal. It therefore seems that pilots often underestimate the urgency of getting the airplane on the ground quickly.

5. Challenges and Regrets

As we expected, the experiment made us aware of many ways in which the ELP could be improved. Some of these are concerned with the robustness of the communication interface between the ELP and the cockpit CDUs and FMS, some are improvements to the user interface and information layout on the CDUs. The most important involves improvements to the risk model and to the path planner. For the risk model, we recognized that we need to increase the risk for crosswinds and gusts in cases where the aircraft has limited yaw control (rudder damage). We also recognized that ground speed at touchdown should be weighted more heavily due to the likelihood of tires blowing at high speeds. Finally, we did not consider the terrain roughness along and in the vicinity of the approach path – this was clearly a factor that the pilots considered when choosing options, particularly when controllability was limited.

For the path planner, we found that the pilots tended to prefer a gentler turn to intercept the final approach course and a longer final approach course, as illustrated by Figure 13. This was particularly true when controllability was poor. While these are all relatively simple improvements, they illustrate the need for further testing and refinement of the models. They also indicate that the path planning needs to incorporate more "knowledge" about flying; when close to the ground, more precision is required, causing the pilots to prefer gentler turns, and gentle course intercepts.

If we had it to do over again, we would split the experiment into two phases: in the first phase, we would remove the decision making aspect and have pilots fly approaches to many different airports with various damage models and weather conditions. We could then use this information to improve the risk model and path planning. In a second phase we would then evaluate the role of the ELP in helping the pilots to make quicker and better decisions.

Unfortunately, getting software like the ELP into the cockpit of commercial transport aircraft is a difficult process. The certification process for commercial avionics is both time consuming and costly. It's also not clear how to verify that the ELP will give the best recommendations in all cases, which opens up the manufacturers to additional liability concerns. Although we are beginning to talk to avionics manufacturers, there are other possible ways of fielding some of this technology that may prove much easier. An increasing number of general aviation pilots are now using handheld devices in the cockpit for maps, charts, and GPS navigation. Such devices range from specially designed units like the Garmin GPSMAP 695/696 to Aviation apps like Foreflight HD for the Apple iPad. Much of the ELP's capability could be incorporated into such a unit, with the advantage that certification is not required. The disadvantage is that tight integration with the aircraft avionics and autopilot are not possible with this solution. A second possibility is to work with an airline or freight carrier to make the technology available through the ACARS system. ACARS is a datalink system that allows communication of data between dispatching centers and aircraft cockpits. The information is accessed through special pages on the cockpit CDU. Using this approach, the ELP could be based on the

ground, and recommendations would be sent to the CDU through ACARS. This also has the advantage that certification is unnecessary, and that the dispatching center would be aware of and could assist with the emergency. We are just beginning to explore these possible avenues for fielding this technology, but hope to forge a partnership with one or more of these players.

The biggest assumption behind this work is that the flight envelope for the damaged aircraft is completely known. For certain categories of failures such as engine failures and control surface failures, the flight envelope can be computed and tested in advance and stored in a library. However, the effects of arbitrary damage are more difficult to predict and it therefore seems unreasonable to suppose that complete models for these conditions are available in a library. In stabilizing the aircraft, the adaptive controller explores portions of the flight envelope, and learns how deflections of the control surfaces affect the aircraft. As a result, the adaptive controller can provide a partial model of the flight envelope as well as some knowledge about areas of the envelope that are likely to exceed control limits. However, there may still be areas of the flight envelope that are only partially known. The most conservative approach is to only consider solutions that remain within the known portion of the flight envelope. However, further exploration of the envelope might result in the ability to produce better solutions (for example, slowing the aircraft could allow a shorter runway to be used). Of course, exploration of the flight envelope involves risk, which must be balanced by any potential gains. In general, this problem becomes a POMDP since we have beliefs about the flight envelope, and we can refine those beliefs through actions that explore the flight envelope. But those actions could also throw us into an undesirable and unrecoverable state. Fortunately, flight envelopes do not appear to be this ill-behaved as a rule. As one moves into a particular state in the control envelope, one gains knowledge of the surrounding states, by virtue of how close the control surfaces are to saturation. It is therefore possible to explore the boundaries of the known control envelope and learn what additional states can be explored without undue risk. We have developed a prototype planner that can generate conditional plans that explore portions of the flight envelope and select different landing options based on the outcome of those exploration actions. Surprisingly, this planner is proving to be much more efficient than we expected and we now believe it may be possible to do this in practice. A more detailed description of this work can be found in (Meuleau and Smith 2011; Meuleau et al. 2010). This approach does raise a difficult user interface issue: how does one depict conditional plans of this sort for pilots? Perhaps the best approach is to only display the most probable path and landing site with some indication that there are decision points along the way.

A second assumption we have made is that the flight envelope remains constant once damage has occurred. It is always possible that additional failures may occur, causing the flight envelope to change again. Unless these subsequent failures are predictable, the best that can be done is to run the ELP again when the failure occurs. A more difficult situation is when the flight envelope changes continuously over time. As an example consider the situation where the left wing is damaged and fuel is leaking out at a rapid rate. Initially, there is loss of lift on the left wing and the aircraft has a tendency to roll to the left. As fuel continues to leak out, the left wing becomes lighter counteracting the loss of lift. As more fuel leaks out, the right wing becomes heavy and the aircraft develops a tendency to roll to the right. Whether or not one prefers a left or a right crosswind on landing therefore depends on how long it will take to get to the runway. To our knowledge, the problem of path planning with continuously changing dynamics has not been addressed in the literature. We think that our approach of doing A^* search over a probabilistic roadmap should still be effective, but the heuristic must take into account the estimate of the time that will be required to reach the runway, so that the landing risk can be evaluated using a reasonably accurate estimate of what the flight envelope will be at the time.

Acknowledgments

This work was supported by the NASA Aviation Safety Program and the American Reinvestment and Recovery Act. We thank John Kaneshige, Stefan Campbell, Shivanjli Sharma, Captain Mietek Steglinski, Matt Gregory and the staff of the NASA Ames Crew Vehicle Systems Research Facility for their work in helping to prepare for and conduct the simulator experiment.

References

Campbell, S.; Kaneshige, J.; Nguyen, N.; and Krishnakumar, K. 2010a. An adaptive control simulation study using pilot handling qualities evaluations. In *AIAA Guidance, Navigation, and Control Conference*.

Campbell, S.; Kaneshige, J.; Nguyen, N.; and Krishnakumar, K. 2010b. Implementation and evaluation of multiple adaptive control technologies for a generic transport aircraft simulation. In *AIAA Infotech@Aerospace 2010 Conference*.

Meuleau, N., and Smith, D. 2011. Optimal motion planning with uncertain dynamics. Technical report, NASA Ames Research Center.

Meuleau, N.; Plaunt, C.; Smith, D.; and Smith, T. 2009a. A comparison of risk sensitive path planning methods for aircraft emergency landing. In *ICAPS-09: Proceedings of the Workshop on Bridging The Gap Between Task And Motion Planning*, 71–80.

Meuleau, N.; Plaunt, C.; Smith, D.; and Smith, T. 2009b. An emergency landing planner for damaged aircraft. In *Proceedings of the Twenty First Innovative Applications of Artificial Intelligence Conference*. AAAI Press.

Meuleau, N.; Plaunt, C.; Smith, D.; and Smith, T. 2010. A POMDP for optimal motion planning with uncertain dynamics. In *ICAPS-10: POMDP Practitioners Workshop*.

Meuleau, N.; Neukom, C.; Plaunt, C.; Smith, D.; and Smith, T. 2011. The Emergency Landing Planner experiment. Technical report, NASA Ames Research Center.

Dynamic Management of Paratransit Vehicle Schedules

Zachary B. Rubinstein and **Stephen F. Smith**
The Robotics Institute
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
{zbr,sfs}@cs.cmu.edu

Abstract

In this paper we describe REVAMP, a mixed-initiative tool for real-time management of paratransit vehicle schedules. Like many applications, paratransit scheduling is a dynamic, execution-driven process, where unexpected events (e.g., traffic, breakdowns, new requests, cancelations) continually force changes to precomputed schedules. The design of REVAMP aims at support for this dynamic, real-time scheduling process. Real-time information on the status and location of vehicles and pending trips is used by REVAMP to maintain a "live" schedule and provide the coordinating Dispatcher with early visibility of potential delays. In response to detected problems or opportunities, REVAMP can be used to generate options for rearranging vehicle schedules to achieve better quality of service. REVAMP is being developed to support daily operations at ACCESS transportation systems, which provides an advance reservation, shared ride paratransit service for the greater Pittsburgh area in southwestern Pennsylvania. We are currently integrating REVAMP into the existing technology base used to support daily operations by one of ACCESS's service providers for an initial pilot test. We describe the principal components of REVAMP and the current state of our solution to the ACCESS scheduling problem.

1 Overview

Paratransit transportation is one of the primary means for people with disadvantages to get around in their daily lives. Typically, this door-to-door service is booked in advance and is publicly subsidized. As the demand for paratransit services increases and availability of public funding decreases, there is a constant balancing act between maintaining a high quality of service while keeping the operations financially feasible. One of the key activities in striking this balance is efficient allocation of vehicles to service trips, especially over the myriad of events that make it difficult to anticipate how a schedule is going to have to evolve throughout its execution during the day. The effectiveness and efficiency of paratransit operations depends heavily on the ability to dynamically manage vehicle schedules in response to execution dynamics.

To augment the ability of Dispatchers to manage their vehicles over execution events, we have developed REVAMP (REal-time Vehicle Allocation application for increased Mobility in Paratransit operations), a mixed-initiative, dynamic scheduling system. REVAMP receives real-time status and location information from the vehicles in the field and maintains a "live" schedule that is constantly updated to reflect what actually happens during execution. This active model provides Dispatchers with better situational awareness of the states of current and future trips and a basis for early detection of trips that are in jeopardy of having poor quality of service. For these problematic trips, REVAMP generates options to present to Dispatchers for rerouting the trips in order to improve their service.

The immediate target of REVAMP is to address the daily operations problem faced by service providers of ACCESS Transportation Systems, the largest paratransit organization in the Southwestern Pennsylvania Region. ACCESS oversees seven individual service operations that are apportioned geographically to the region. These service providers make about 6000 trips daily throughout Allegheny County. Of these about forty percent are subscription service (regularly scheduled) and sixty percent are day-ahead reservations. Over the past twelve months, ACCESS has provided service to around 26,000 unique customers. We are currently developing a system that integrates REVAMP into ACCESS's existing technology base, for the purpose of carrying out an initial pilot test with one of their larger service providers. Our overall goals are to improve customer quality of service, while simultaneously decreasing provider costs and providing the opportunity to offer expanded same-day request service.

2 Paratransit Management Problem

The Paratranist management problem is an instance of the Dial-a-Ride Problem (DARP) (Cordeau and Laporte 2003). The objective in a DARP is to design vehicle schedules to satisfy requests for travel between pick-up and drop-off locations at specified times. Typically, as input, DARPs specify a set of requests, a set of available vehicles, and a set of constraints ensuring the quality of the service, e.g., time windows within which pick-ups must be made or maximum allowable ride time for a passenger. DARPs may be oversubscribed, i.e., not all requests can be serviced within their constraints, in which case the constraints may be relaxed in order to help accommodate the extra requests. There often are optimizing objectives for DARPs, such as minimizing cost by minimizing the number of vehicles used while allowing some level of constraint relaxation, or maximizing

service quality by determining the minimal number of vehicles to satisfy the requests without relaxing constraints. Formally, there are two classes of DARPs, static and dynamic. In a static DARP, all the requests and the available vehicle are known up front. In a dynamic DARP, the requests are serviced as they arrive and the available vehicles can be increased. Most real-world DARPs are hybrids, with a majority of the requests but not all being known up front and there mostly being a static number of available vehicles but extra ones are available at a cost.

We will focus on the version of DARP as it is instantiated in ACCESS. Currently, ACCESS does not support day-of requests, so the requests are known in advance. The one exception to this advance knowledge is "will call" requests, where passengers do not know the exact time of a return trip, e.g., for a doctor's appointment. These requests are handled dynamically as soon as the call is received to pick up the passenger.

The number of available vehicles is also known in advance. There are different types of vehicles available, each providing capacity to carry a specified number of passengers. Some types of vehicles provide wheelchair capacity, while the others have strictly ambulatory capacity. In the former case, wheelchair capacity can be converted to ambulatory capacity, so the overall carrying capacity will vary depending on how the wheelchair capacity is used.

The service quality constraints for ACCESS are the following:

- *Pick-Up-Window Constraint* - The constraint on the window of time it is permissible to pick up a passenger is relative to time that ACCESS negotiates with the passenger for his or her pick-up. Specifically, a pick-up should not occur any earlier than ten minutes before the negotiated time or any later than twenty minutes after the negotiated time. For "will-calls", the pick-up should occur no later than forty-five minutes after receiving the call.
- *Ride-Time Constraint* - the ride time for a passenger should be no longer than the maximum of twenty minutes and the minimum of two hours and twice the estimated time to go directly from the pick-up to the drop-off (i.e., $\max(20min, \min(2hrs, 2 * \textit{direct-tranisit-time}))$).

The providers are allowed to relax these constraints, but, overall performance, is based on meeting them. If these constraints are violated too often or by large magnitudes, the provider risks having its service area reduced or loosing its contract. The goal for the provider is to meet these constraints while minimizing the resources it needs to use.

In preparation for the next day, providers generate a solution , i.e., a set of vehicle schedules, to the DARP for the following day's requests. The solution is generated by using an offline scheduler based on (Jaw 1984) to create the base schedules and then hand tweaking them. Complicating this process is that, on average, there is a 15% request cancellation rate that occurs during the day of service. The human schedulers actually over allocate the vehicle timelines in anticipation of these cancellations. That is, the start-of-day schedules would require frequent relaxations of the constraints if they were to execute as scheduled, so the schedules start with a number of trips that are already in jeopardy of missing their service constraints. It is left to the Dispatchers to resolve these problematic trips as cancellations arrive.

Although an advance reservation policy enables the advance development of a daily operations schedule, unexpected events that occur as execution proceeds (e.g., vehicle breakdowns, traffic accidents, "will call" requests, trip cancellations) quickly force changes and degrade the quality of originally planned vehicle itineraries. In current practice, the Dispatchers responsible for coordinating the movements of a service providers vehicles respond to these events in a fire fighting fashion. Since the start-of-day schedule becomes an increasingly distant reference point as the day progresses, they often become aware of problems (e.g., potential late pickups) late, restricting their ability to effectively respond. Although they have visibility of the real-time location of vehicles, Dispatchers must often make decisions to redirect vehicles without good understanding of the downstream consequences to subsequent trips scheduled for various vehicles. During peak request periods, the Dispatcher can often become overwhelmed due to time pressure and decision complexity. Ultimately, the quality of decisions depends heavily on the Dispatchers expertise, and it is difficult to find and retain experienced personnel in this position. As a result, the decisions that are made are often suboptimal with respect to maximizing customer quality of service, and this has a cascading effect through the day. This situation is typical of operations in other paratransit service contexts.

3 REVAMP

REVAMP is designed to address the problems identified above and provide a basis for improving the real-time decision-making of service provider Dispatchers. Most basically, it is designed to incrementally accept real-time updates of vehicle status, and provide the Dispatcher with a live schedule that continually reflects the current execution state. Since this schedule encodes all relevant constraints and requirements (e.g., expected travel durations, negotiated pick-up times, maximum transit time limits), one immediate benefit is the early detection and alerting of emerging problems (e.g., given the customer pick-up that was just reported, the next scheduled pick-up after this current trip is projected to be beyond its acceptable window). This capability alone affords the Dispatcher with more time to take corrective action, and in the worst case, enables the Dispatcher to inform customers in advance of unavoidable delays (a service that ACCESS has expressed an interest in providing). To support Dispatcher response to detected problems, REVAMP is designed to generate options for rearranging trips across vehicles to minimize the impact on customer quality of service. This same options generation mechanism can also be used as a basis for improving the efficiency of vehicle itineraries when opportunities arise (e.g., trip cancelations) and for accommodating new requests that arrive dynamically through the day (e.g., "will call" return trips).

REVAMP consists of a suite of components whose underlying object-model maps client requests and their corresponding pick-up and drop-off tasks to a Simple Temporal Network (STN) (Dechter, Meiri, and Pearl 1991). In-

cluded in that object model are vehicles and their timelines (schedules) populated with the request and travel tasks. REVAMP components include a *schedule loader* for installing the start-of-day schedule, a *schedule updater* for integrating real-time start and finish time information as well as cancellations into the schedule and for identifying problematic trips and opportunities for improving existing schedule quality, and a *gap finder* for generating high-quality options for inserting a request's constituent tasks on vehicle timelines.

3.1 Object Model

The top-level objects in the model for REVAMP are vehicles, requests, and tasks. There are three types of vehicles that differ in capacity and whether or not they are wheelchair accessible. Each vehicle has a corresponding timeline (schedule), which consists of an ordered set of tasks. A request represents the information provided when a client books a trip and has attributes for the negotiated time (preferred pick-up time or the time when a "will call" is received), the latitudes and longitudes of the pick-up and drop-off locations, the number of wheelchair passengers, the number of ambulatory passengers, and whether or not it is a "will call" trip. A request also indicates the provider to which the request has been assigned by ACCESS and a unique identifying number for the request known as a trip id.

When a request is created, both pick-up and drop-off tasks are generated and mapped to the underlying STN. The STN is a graph of time points connected by binary constraints that determine the permitted intervals, i.e., lower and upper bounds, for any given time point. An absolute constraint emanates from the special, anchored calendar zero (cz) time point and sets the lower and upper bounds of the target time point to absolute times. A relative constraint sets the lower and upper bounds of the target time point as offsets from the source time point's bounds. A task is represented in the STN as a start time point and an end time point connected by a duration constraint whose lower and upper bound values are the time it takes to complete the task. As shown in Figure 1, the pick-up-window constraint in ACCESS is represented as an absolute constraint on the start time point of the pick-up task. The ride time constraint is represented as a relative constraint from the start time point of the pick-up task to the start time point of the drop-off task.

In addition to the pick-up and drop-off tasks, there are travel tasks, which are dynamically generated when inserting the request tasks onto a vehicle timeline. The computation for determining the duration of a travel task depends on the origin and destination of the travel and the time of day when the travel occurs. The travel duration is calculated by, first, using a great-circle computation to determine the distance between the origin and destination. Then, a speed is selected based on whether or not the travel will occur during rush hour. The final duration is the distance divided by the speed. The non-rush-hour speed is used if there is enough time within the bounds of the travel task to complete the travel without having to travel during rush hour. Otherwise, the rush-hour speed is used.

One of the complications of this duration model is that, as travel tasks move in time, i.e., become earlier or later due

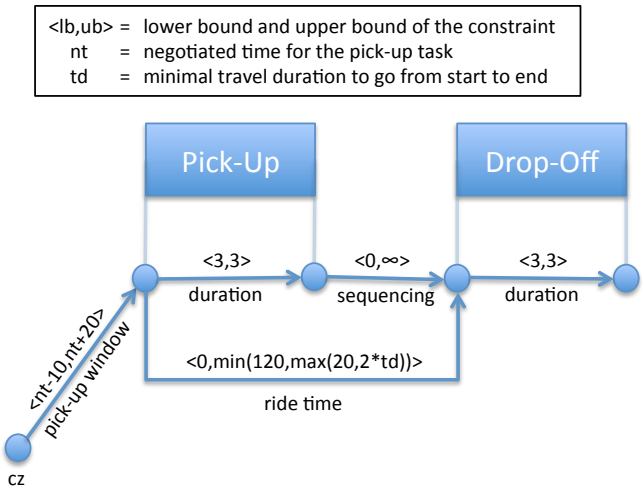


Figure 1: Request Tasks STN Mapping.

to changes in the schedule, their durations may change as a result of moving in or out of rush-hour periods. To ensure proper travel durations, REVAMP adjusts the travel durations of constituent tasks where necessary after any change is made to a vehicle timeline. This process is elaborated below in the descriptions of the components.

3.2 Schedule Loader

The schedule loader converts the provider's start-of-day schedule into the underlying object model. Complicating this procedure is that the start-of-day schedules, in anticipation of the 15% cancellation rate, are oversubscribed and can have many requests scheduled that will violate either or both pick-up-window and ride-time constraints. For example, in the start-of-day schedules of a particular service provider for a 1 week period, approximately $2/3$ of the scheduled requests had constraint violations. A start-of-day schedule of request tasks (i.e., pick-ups and drop-offs) for a given vehicle is loaded into REVAMP by processing each task in order from earliest to latest. Constraint violations are handled by a constraint relaxation process (described below).

A given task is appended to a vehicle timeline by first checking to see if the task is at a different location than that of the previous task. In the case where the timeline is empty, the vehicle is assumed to be at the location of the garage of the provider. If the new task is at a different location, then a travel task is created with the appropriate duration for getting from the location of the previous task to the location of the new task sometime between the earliest finish time of the previous task and the latest start time of the new task. If the travel task is created, it is inserted on the timeline by adding a sequencing constraint between the end time point of the previous task and the start time point of the new travel task. A sequencing constraint dictates that the source time point must occur before (or simultaneously with) the target time point. Finally, the task being processed is added to the timeline by inserting a sequencing constraint between the last task on the timeline and it.

Since the start-of-day schedule can be overloaded, it may not be possible to assert the sequencing constraint to the

new task without violating its upper bound constraint (either the pick-up-window-constraint on a pick-up task or the ride-time constraint on a drop-off task). If it cannot be added, then the upper-bound constraint is *relaxed* by remembering its original upper bound value and setting the upper bound on the constraint to ∞, i.e., unbounded. Then, once the sequencing constraint is asserted, the upper bound on the upper bound constraint is anchored to ensure that the current tardy time does not slip any later.

The end result of the schedule loader is a complete start-of-day schedule modeled with all necessary relaxations and in a form that can continue to be modified to reflect the impact of execution-time events on the schedule.

3.3 Schedule Updater

The schedule updater is responsible for keeping the internal model of the schedule in sync with how the schedule is actually executing. It incorporates into the model execution-time updates on the actual start and finish times of request tasks and on cancellations. In each case, constraints on downstream tasks are relaxed or tightened when necessary/possible. The schedule updater also detects impending requests that are in jeopardy of missing service quality constraints, and, in the case of cancellations, triggers the generation of options for the most imperiled requests in order to improve overall schedule quality.

Actual start and finish times for request tasks are reported by the vehicles as they make their stops. When the actual start time for a task is received, the start time point of the task is fixed at the reported time by adding an absolute constraint with the lower and upper bounds set to the reported time. In the straightforward case where schedule updates on the start and finish times of tasks are received in the order specified by the schedule, the updater relaxes any constraint that it must in order to achieve consistency with actual times. If a task finishes later than expected, then the duration constraint on that task is lengthened. If there is a violation when attempting to assert the new duration length, the STN returns the set of constraints that are inconsistent with the new constraint and the amount that the new constraint violates the existing constraints. The returned constraints are used to relax the downstream constraints in order to make it possible to assert the new duration constraint. First, the returned set of constraints are searched for the first upper bound constraint found. Then, that constraint is relaxed by the magnitude of the violation. Next, the new duration constraint is attempted once more. If it fails, then the relaxation process is invoked again with the new set of returned constraints. This iteration continues until the duration constraint can be asserted.

In the case where a task finishes early, the duration constraint is shortened, all subsequent relaxed tasks on the timeline are tightened when possible. If a pick-up task starts earlier than its pick-up-window constraint allows, that constraint is relaxed to accommodate the actual start time. After any changes are made to the timeline, the scheduled tasks are mapped over to ensure that relaxed constraints are as tight as possible and travel tasks have their appropriate durations. A possible side-effect of these modifications is that the upper-bound constraints on downstream tasks may have to be tightened or relaxed, depending on whether or not the new travel durations are shorter or longer. Note that previously relaxed constraints are never tightened beyond their original values.

Drivers are not required to pick up passengers in the order of the schedule and, therefore, can report starting a task that is not the next task expected to execute on the schedule. In this case, the schedule updater moves the reported task up in the schedule to follow the last completed task, inserting travel tasks where needed to get from the last completed task to the reported task and from the reported task to the task that was scheduled to execute next. This insertion may require relaxation of some of the upper-bound constraints on pending tasks that were expected to execute earlier. After the insertion, the durations of the travel tasks among the subsequent tasks are corrected when necessary.

As the actual start and finish times are reported, the schedule updater keeps a running list of all tasks that have had one or more of their constraints relaxed. This list is used to determine if REVAMP should alert the human dispatcher of trips that are likely to violate the service constraints. The current policy alerts the dispatcher of trips in jeopardy that are to be serviced within the next two hours. However, if desired the dispatcher can also request to view all endangered trips, along with the expected magnitude of delay in each case.

The schedule updater also updates the schedules of vehicles when requests are cancelled. When a request is cancelled, its pick-up and drop-off tasks are removed from the timeline as well as their corresponding travel tasks. Then, when necessary, new travel tasks are inserted to bridge the distance between tasks on either side of the removed tasks. As with all timeline modifications, the travel tasks on the timeline are checked to ensure that they have the correct durations. There is a restricted cancellation that occurs when a driver arrives at a pick-up task and discovers that no one is there. In this "no-show" case, the drop-off task for the request is removed from the timeline. A cancellation potentially represents an opportunity for remedying an endangered trip by moving it to the timeline with the cancellation that now has more slack on it. The schedule updater will trigger the gap finder on cancellations to present rescheduling options to the dispatcher for rerouting problematic trips.

3.4 Gap Finder

The gap finder generates scheduling options for servicing a given request. It is triggered either automatically by the schedule updater in the case of cancellations or deliberately by the Dispatcher via the list of problematic or unscheduled requests. In the latter case, if the request is already scheduled, it is temporarily unscheduled before searching for options. The general strategy of the gap finder is to search all vehicle timelines for feasible slots between scheduled request tasks for the request's pick-up and drop-off tasks. A slot is feasible for a task if *a*) the time window of the slot can accommodate the constraints of the task plus any additional travel required, and *b*) the current available ambulatory and wheelchair capacity is sufficient to accommodate the passenger demands on the request

All pairs of feasible slots for a request on a given vehicle timeline are found by traversing the timeline and trying the eligible slots for the pick-up task on the request. For each slot within the pick-up window, the pick-up task is inserted and required travel tasks are inserted and adjusted as required. If no violation occurs, the slot is feasible. Then, with the pick-up task temporarily inserted, the downstream slots on the timeline are checked to see if they are feasible slots for the drop-off task. The valid slots for the drop-off task are found by searching the remaining slots on the timeline that have time bounds within the constraints of the drop-off task and hypothetically inserting the drop-off task and associated travel into each of these slots. Any time a drop-off task is successfully added, the travel tasks of other surrounding tasks are adjusted to ensure that they have the correct durations. If those durations cannot be enforced without a violation, the drop-off slot is not feasible. For each pair of feasible slots found, a candidate is generated and saved. Then, the drop-off task is unscheduled and the subsequent slots are checked to see if they are valid for the drop-off task. Once there are no more slots to search for the drop-off task, the pick-up task is unscheduled and the subsequent slots on the timeline are examined to see if they are valid slots for the pick-up task. This process continues until all feasible candidates for that request on that timeline have been generated. All possible candidates are generated by searching all the vehicles' timelines.

For each candidate option generated, the following performance measures are computed and associated as the candidate is generated:

- *Pick-Up Tardiness* - the maximum of zero or the amount of time by which the earliest start time of the pick-up task exceeds the negotiated time.
- *Drop-Off Tardiness* - the amount of time by which the earliest start time of the drop-off task exceeds the earliest possible start time of drop-off task (assuming direct travel from pick-up location to drop-off location).
- *Additional Travel Duration* - the amount of travel time that had to be added to the vehicle timeline to accommodate the candidate.
- *Additional Total Tardiness* - the sum of the pick-up tardiness and any tardiness introduced to other tasks on the timeline due to the shifting required to accommodate scheduling the candidate.

These metrics are used to prioritize the candidates. Which metrics are optimized depends on what is most valued in the organization. For ACCESS, the most important quality is to service all the requests without violating their service constraints. To match this priority, the gap finder prioritizes candidates to minimize Additional Travel Duration, which minimizes the amount of overall time that has to be allocated to service a request and, thus, leave more available capacity to meet future requests. A subset of the top candidates are presented to the Dispatcher as possible options, since the Dispatcher may have additional knowledge about the executing environment that might give preference to a lesser ranked candidate. In automated mode, REVAMP would simple select the top-ranked choice and add this commitment to the schedule.

When the timelines are tight, it is possible that no candidate can be found for a request. In this case, the upper bound constraints on the pick-up and drop-off tasks of the request are iteratively relaxed by set amounts until candidates can be found for the request. Before a candidate is generated, the upper bound constraints are tightened to their original (satisfied) values, the feasible relaxation values determined for these upper bound constraints by this search are stored with the candidate. As before, the candidates are prioritized. But, for relaxation, we chose to minimize the sum of the pick-up and drop-off tardiness, since in this case this alternative metric tends to better minimize worst case tardiness. When a relaxed candidate is scheduled, the upper bound constraints on the pick-up and the drop-off tasks are first relaxed to their stored values on the candidate.

4 Current State

The individual components of REVAMP are implemented and have been tested on the types of events present in historical data provided by ACCESS. These events include start-of-day requests and routes, actual start and finishes time, cancellations, no-shows, and vehicle downtimes. We are currently integrating REVAMP into the existing technology base provided by ACCESS to its providers, in preparation for an initial pilot test of developed capabilities with a particular ACCESS provider. This effort involves developing an API for the base system to convey to REVAMP the initial schedule, real-time tracking information, and the changes made to the schedule by Dispatchers, and for REVAMP to inform the base system of changes in the schedule during execution, of updates to the list of requests that are currently expected to violate service constraints, and of options for reassigning requests to vehicles to improve overall schedule quality. Where appropriate, we will use the base system's user interface to minimize user training, but we are also exploring use of other graphical display formats that better convey real-time schedule status and dynamic scheduling options.

We are currently running experiments to evaluate the potential improvement REVAMP can provide to Dispatchers. We have from ACCESS the start-of-day information, including all requests, initial schedules, and vehicles used, and the corresponding end-of-day information, including the actual times for the stops on the routes, the cancellations (including no-shows), and the information on the "will call" requests. We are using this data to simulate the use of REVAMP in fully autonomous mode to manage the schedule over the events of those days and compare its performance to the actual performance of the Dispatchers on those days. The metrics we use for comparing performances are the number of relaxed requests, the number of relaxed pick-ups, the mean/median/maximum relaxation for pick-ups, the average relaxation for pick-up over all pick-ups, the number of relaxed drop-offs, the mean/median/maximum relaxation for drop-offs, and the average relaxation for drop-offs over all drop-offs. This comparison is obviously an approximation,

since the moment REVAMP diverges from the actual schedule, actual travel times are no longer available nor are the effects of drivers servicing stops in a different order than the schedule. To make the experiments more realistic, we have formed models based on the year's worth of data that we currently have. These models vary the duration appropriately on the new trips and report stops out of order in appropriate contexts with representative frequency.

As a preliminary result to demonstrate that there is room for improvement, we took start-of-day schedules, loaded them, and collected the comparison metrics. Then, we used the gap-finder (in automated mode) to schedule all requests in the start-of-day schedules using the same configuration of vehicles. We ran this experiment on a week's worth of data from one of the providers. Table 1 shows the schedule comparison for one day, which is representative of the other days' results. As can be seen by the far fewer relaxations, reduced maximum relaxations, and mostly reduced averaged relaxations, there appears to be substantial room for improvement. A caveat to this result is that the human schedulers at the provider may be considering other constraints, such as stability of subscription routes, driver familiarity with particular areas, etc., that REVAMP does not currently model. And, as noted earlier, the start-of-day schedule is formed with the expectation of a 15% cancellation rate. Nevertheless, the difference is so dramatic that it appears that substantial improvement is likely. In particular, just having the start-of-day schedule that is better aligned within customer service constraints reduces the initial demand on the Dispatcher to pay attention and manage those trips that are already in jeopardy of being delayed.

	provider	REVAMP
total-requests	902	902
total-relaxed-requests	577	17
total-relaxed-pickups	386	17
avg-relaxed-pickup-overall	25m 25.84s	44.80s
avg-relaxed-pickup	59m 25.57s	39m 37.29s
median-relaxed-pickup	31m 9s	42m 29s
max-relaxed-pickup	5h 1m 17s	1h 15m 17s
total-relaxed-dropoffs	327	9
avg-relaxed-dropoff-overall	7m 59.65s	13.63s
avg-relaxed-dropoff	22m 3.07s	22m 46s
median-relaxed-dropoff	13m 14s	16m 5s
max-relaxed-dropoff	4h 10m 11s	50m

Table 1: Start-of-Day Schedule Comparison

5 Future Work

Our short-term focus is on refining and hardening REVAMP in its integration with ACCESS's existing technology base to ensure its performance and reliability in the pilot and in wider deployment in ACCESS. A large part of that effort will be focused on developing the user interface that will allow the Dispatchers to grasp the situational awareness and options provided by REVAMP, evaluate the alternatives, and take quick effective actions. If the pilot is successful and REVAMP is able to quickly offer good scheduling solutions to unscheduled requests, we would like to work with ACCESS to offer a limited same-day service.

We are also investigating incorporation of more sophisticated option generation capabilities within REVAMP. First, the current method for relaxing a request that cannot be feasibly scheduled has a limited degree of freedom in that the only constraints relaxed are those of the tasks constituting the unscheduled request. It may be possible to find better solutions by adjusting multiple trips on timelines to accommodate a given unscheduled request. One approach we are exploring is a variant of task swapping (Kramer and Smith 2004), where other temporally overlapping requests are temporarily unscheduled to create space for a target request and then reinserted in a feasible manner.

Second, we would like to expand REVAMP to include backward search from the drop-off time, i.e., given when a person needs to be at a destination, schedule backward to establish the pick-up time. In this case, the drop-off time would be treated as a hard, upper-bound constraint. In addition to better servicing appointment constraints of passengers, it also could be used to improve multi-modal traveling where paratransit transportation is used to connect with other fixed route services. In general, REVAMP should provide a basis for better coordination of multi-modal trips involving ACCESS transport to a public transit gateway (or other fixed route service) and vice versa. By incorporating external schedules and capitalizing on real time information of the locations and status of ACCESS vehicles (and, in the future, potentially the real-time location of public transit vehicles), we believe that better synchronization of multi-leg trips can be achieved, resulting in less customer wait time and shorter overall travel times. To facilitate this type of planning by passengers, we envision use of on-line browser-based and mobile-based applications for planning and booking both basic paratransit trips and multi-modal trips.

Acknowledgements The research described in this paper was funded in part by a grant from the 2010 Federal Transit Administration New Freedoms Initiative and the Robotics Institute at Carnegie Mellon University.

References

Cordeau, J.-F., and Laporte, G. 2003. The dial-a-ride problem (darp): Variants, modeling issues and algorithms. *Quarterly Journal of the Belgian, French and Italian Operations Research Societies* 1:89–101.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49:61–95.

Jaw, J.-J. 1984. *Solving large-scale dial-a-ride vehicle routing and scheduling problems*. Ph.D. Dissertation, Massachusetts Institute of Technology. Dept. of Aeronautics and Astronautics., Cambridge, MA.

Kramer, L., and Smith, S. F. 2004. Task swapping for schedule improvement - a broader analysis. In *Proceedings 14th International Conference on Automated Planning and Scheduling (ICAPS 04)*.

The MMP: A Mixed-Initiative Mission Planning System for the Multi-Aircraft Domain

Ruben Strenzke and Axel Schulte

Universität der Bundeswehr München
Department of Aerospace Engineering, Institute of Flight Systems
Werner-Heisenberg-Weg 39, 85577 Neubiberg, GERMANY
{ruben.strenzke, axel.schulte}@unibw.de

Abstract

The Universität der Bundeswehr München is conducting research in the field of single-operator multi-aircraft guidance. This article describes the Mixed-initiative Mission Planner (MMP) as far as requirements, concept, design and implementation are concerned. The MMP is applied to a time-constrained multi-aircraft in-flight mission planning problem. It works in conjunction with a cognitive assistant system for an Uninhabited Aerial Vehicle (UAV) operator, who generates and modifies multi-aircraft mission plans incrementally. The assistant system is able to evaluate, complete, and generate such plans with the aid of the MMP and to communicate with the operator, which results in a mixed-initiative planning approach. Mixed-initiative planning systems need to be able to evaluate the partial or complete human plans as well as the problem itself. For this reason there are two instances of the MMP in our system design. One is configured as a slave to the human input to assume what the human is planning and the other as a free planner that generates reference plans. The assumed human plan includes temporal information about already planned and future tasks, and it can be compared to the reference plan by the assistant system, allowing it to decide whether and when to take initiative. The MMP prototype implementation consists of a Planning Process Manager that dynamically generates problem descriptions, a freely available PDDL 2.2 compatible planner, and multiple domain descriptions.

Introduction

Uninhabited Aerial Vehicles (UAVs) in use today are typically performing preprogrammed missions that can be manually altered in-flight by a crew of at least two human operators. With the advent of multi-UAV scenarios and Manned-Unmanned Teaming (MUM-T), which stands for the joint operation of manned and unmanned assets, the operator-to-vehicle ratio shall be inverted in future applications. For this reason the Institute of Flight Systems at the Universität der Bundeswehr München is conducting research on artificial cognitive systems that aid the UAV operator in coping with high work demands caused by multi-vehicle guidance and mission management. On the one hand, these systems can be deployed onboard of the UAVs to let them become semi-autonomous, cooperative, and guidable on a task-based level, which is more abstract than programming waypoints (Uhrmann, Strenzke, and Schulte 2010). On the other hand, the operator shall be supported in mission planning and UAV tasking by a cognitive assistant system (Donath, Rauschert, and Schulte 2010). In this article we describe the mission planning module, which enables interactive online multi-vehicle mission planning, in which both the human (UAV operator) and the machine (assistant system) can take initiative. Hence, we call it Mixed-initiative Mission Planner (MMP).

Of the many mixed-initiative planning systems that we examined, to our knowledge only (Funk et al. 2005) explicitly deals with online planning, i.e. dynamic changes of the situation or the goals during the human problem-solving process and the associated re-planning under time pressure. Their approach allows the user to delegate tasks on different levels of a task hierarchy, which leaves the planning of details unspecified by the user to the automation. This human-automation integration approach is called *supervisory control* (Miller et al. 2005). In contrast to this, we implemented *cooperative control* with real automation-initiative, i.e. the assistant system can initiate dialogs with the user, which is a step into the direction that (Ferguson and Allen 1998) have chosen. But our approach differs by following the Cooperative Automation and assistant system paradigms of (Onken and Schulte 2010), who proclaim a rather passive and invisible assistant, that only interacts with the human operator on its own initiative and does this solely in case he/she shows suboptimal behavior. The assistant system is otherwise invisible and silent, thereby leaving mission responsibility to the human and keeping him/her in the control and decision loop.

First, this article describes the requirements posed by the MUM-T application problem and our assistant system approach. Then, the concept of the MMP is derived from these requirements. After that, we explain its design and give an overview of the implementation. Also, evaluation approaches and first results are presented, and current problems as well as future work are outlined.

Requirements for the MMP

The UAV operator's workplace is located inside a helicopter cockpit, which is part of a large-scale Manned-Unmanned Teaming simulation (Uhrmann, Strenzke, and Schulte 2010). The MMP shall enable the helicopter crew to accomplish experimental MUM-T missions with manageable workload. In this section, we describe the MUM-T application scenario and the assistant system with respect to the requirements they generate for the MMP.

Helicopter and UAV Mission Application Scenario

In our MUM-T scenario, a manned transport helicopter is supposed to carry troops from a pickup zone to an operation area with two possible drop zones nearby (cf. figure 1). In order to get there, the helicopter has to cross the forward line of own troops (FLOT) by the use of defined corridors within specified time windows. In total, there are about 25 mission-relevant locations in the scenario. Three UAVs are taking over the preceding reconnaissance of the helicopter routes and of the drop zone. The more UAVs perform the reconnaissance of a route, the broader is their sensor footprint, thereby increasing safety for the manned high value asset. Prior to the start of the mission, a mission order is provided, which includes certain constraints, e.g. the preferred drop zone, the preferred ingress and egress corridors, the mandatory pickup zone, the permitted times to land at the pickup and drop zones, the corridor opening times, the takeoff clearance time (earliest mission start), as well as the final destinations of all aircraft and troops. Hence, the automated planning of such a mission makes concurrent actions and temporal planning necessary.

During execution of the mission the operator enters a mission plan into the system stepwise by allocating a series of individual tasks to each of the UAVs via a task-based guidance graphical user interface (GUI) (Strenzke et al. 2011). The generated tasks are chronologically ordered, but do not contain any time tags. The following task types can be given to the UAVs: take off, land, transit, cross FLOT, recce route, recce area, and object surveillance. The semi-autonomous UAVs possess restricted planning capabilities that allow them to insert mandatory tasks that the operator left out. In the course of the mission, ground objects are spotted by the UAVs. These events can lead to the necessity of re-planning the mission (e.g. primary landing site or corridor are threatened, cf. figure 1). Also, after the first mission is accomplished, a second troop transport order is given to the helicopter crew, making re-planning of the egress phase necessary.

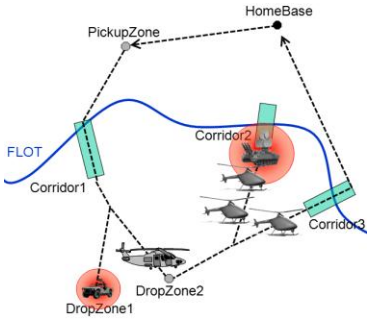


Figure 1: Manned-Unmanned Teaming Mission Scenario

In this simulated scenario we do not regard the movement of any dynamic objects which are not under control of the human (e.g. ground vehicles). The helicopter is also under his/her control (constraint-based guidance, e.g. which corridor and landing site to use) because in our MUM-T scenario the UAV operator and the helicopter commander is the same person. Hence, the mission planning problem is deterministic.

In the helicopter domain, time plays an important role. A mission plan always has to be complete, i.e. including landing in safe territory before fuel runs out. In addition to that, timely coordination is necessary (e.g. site has to be reconnoitered before landing). Therefore, all agents have to be included in the plan. This makes the problem complex. The solution process and end state are not well-defined (i.e. there are multiple possible ways of solving the problem and these are not known in advance). Hence, we need optimality criteria, merged together in a cost function. This function shall represent the minimization of risk to human life (i.e. the helicopter crew), risk to equipment (i.e. risk to the manned and unmanned aircraft), violation of the mission order, as well as financial costs (i.e. short flight paths).

Manned-Unmanned Teaming Assistant System

The MUM-T assistant system is realized as a knowledge-based system that supports the UAV operator upon detection or anticipation of suboptimal behavior. It mainly holds knowledge about the modes of interaction with the human operator (Donath, Rauschert, and Schulte 2010). In the mentioned cases, the assistant system has three options to aid the operator. It can provide a warning, suggest an action proposal, or initiate an action (e.g. reconfiguration of some system). To communicate with the operator, the assistant system instantiates a dialog or makes an announcement via speech synthesis and the displaying of a message box in the task-based UAV guidance GUI. Whenever appropriate, this message box includes a few buttons

that allow the operator to invoke further aid by the assistance system or to either accept or reject its proposals. E.g.

- Assistant takes initiative: *"UAV1 needs follow-up task"*
- Operator presses *"proposal"* button
- Assistant proposes: *"Add task transit HB PZ for UAV1"*
- Operator presses *"accept"* button
- Assistant affirms: *"Added task for UAV1"*

These dialogs can either refer to a single task to be allocated to or executed by a UAV, or to a complete plan to cover the remaining mission goals. Following the Cooperative Automation and assistant system paradigms of (Onken and Schulte 2010) the assistant system initiates these dialogs only if it is found necessary to support the human operator, i.e. he/she made an error (i.e. his/her behavior is below a certain optimality threshold) or an error is anticipated by the assistant system (i.e. his/her plan seems below a certain optimality threshold). To decide whether, when, and how assistance should be provided to the operator

- the assistant system has to be able to anticipate, which tasks the operator has to execute and when he/she is supposed to do this (i.e. *plan is incomplete* and has to be evolved soon due to time constraints), and
- the assistant system has to be able to notice suboptimality in the past planning performance of the operator (i.e. his/her *plan is too suboptimal*).

Accordingly, during the execution of the mission the assistant system has to check the completeness and optimality of the operator-given UAV tasks upon any operator input that is conflicting with the current plan, any relevant tactical situation change (e.g. new threat enters the scenario), and any mission order change (i.e. new mission objectives received or mission objectives have already been met). The assistant system also needs the ability to propose a new plan to the operator in case his/her plan is infeasible or suboptimal. Taking all this together, the system requires

- the *ability of temporal planning* to complete, generate from scratch, and monitor a task agenda, as well as
- the *ability of plan evaluation* (cost comparison).

Hence, the world model of the MMP has to incorporate the conceptions of time and costs, and it has to be able to perform the necessary calculations in order to provide a basis for assistant system decisions. These decisions have to be made fast, therefore anytime planning is useful.

Concept of the MMP

In order to develop a Mixed-initiative Mission Planner for the MUM-T domain, we first have to analyze the role of the human and of the automation as well as their mixed-initiative interplay. The starting point is that the human shall be able to enter mission plan into the system completely on his own. This is due to our passive assistant system approach and the mentioned responsibility he/she has concerning the mission and the involved systems. This planning task has to be performed by means of a graphical user interface that only allows the incremental generation of a single plan.

Now we suppose that the human has at least one plan in his/her mind and that he/she enters this (the best one or the primary) into the UAV guidance system. Therefore, we can distinguish between the *plan(s) in the human mind* (*human plans – HuP*) and the *plan that is stored in the system* (*system plan – SyP*), i.e. the current active plan for the automatic UAV guidance, that is recognized and modified by the human operator and the assistant system in mixed-initiative fashion. Because it is difficult to get hold of the HuP, the assistant system can only evaluate the SyP.

Plans generated by the assistant system (*assistant system plans – AsP*) constitute a third type of plans in our concept. Figure 2 shows the concept that incorporates all these plan types. Both human and machine shall be able to take initiative in order to manipulate the system plan according to their understanding.

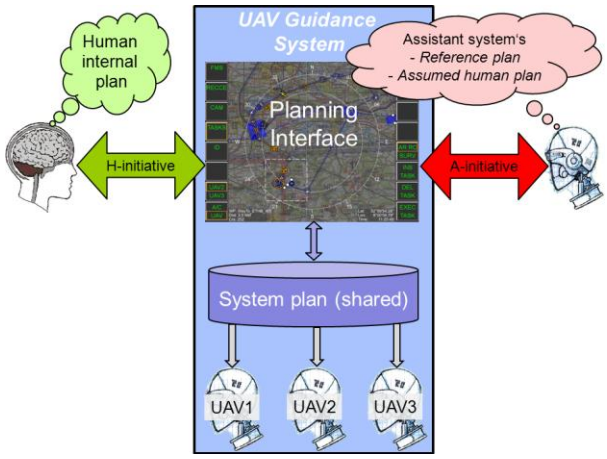


Figure 2: Mixed-initiative planning concept for the MMP

The assistant system plans (AsP) can be further divided into *what the machine computes as the best possible plan* (*reference plan – ReP*) and *what the machine supposes that the human is planning* (*assumed human plan – aHuP*). In theory, multiple plans of each subtype can be stored by the assistant system, but this is not regarded in the following. The assumed human plan (aHuP) converges to the true human plan (HuP) with each additional detail the human discloses by tasking the UAVs and thereby expanding the SyP. Furthermore, the operator is driven to detail his plan by the warnings and proposals of the assistant system, possibly letting the HuP and SyP converge to either the aHuP or the reference plan of the machine (ReP).

Similar to (Miller et al. 2005) our concept follows a shared task model that allows human and machine to communicate about tasks for the aircraft, goals and plans.

This is explained in more detail in (Strenzke et al. 2011). Furthermore, our MMP concept is based on planning the mission as reaching a defined world state (which is in our case the end state of the mission) and optimizing the way to this state. Hence, partial human plans must be completed in the machine's mind to evaluate them.

Design of the MMP

During the mission, the assistant system receives information about the mission order, the current tactical situation, and the aircraft task agendas out of SyP (cf. figure 3). From this information, the assistant system has to generate the aHuP as well as the ReP. In order to accomplish this, different constraint sets have to be transferred to the MMP. For this purpose the assistant system uses the Simple Temporal Constraint Interface (STCI) as input interface of the MMP. We therefore deploy two instances of the MMP. One is intended to create the aHuP, and the other shall generate the ReP at the same time.

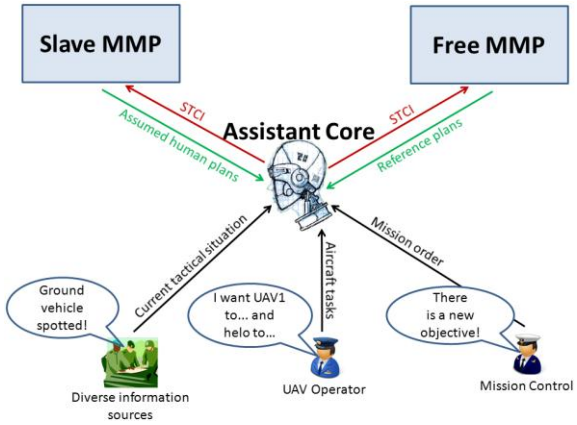


Figure 3: Integration of the assistant core and MMP instances

Slave and Free MMP Instances

Both instances of the MMP receive the information about the current tactical situation, but they differ with respect to the constraints they take into consideration.

The so-called *Slave* instance of the MMP is slave to the human input, i.e. it uses the constraints expressed by the SyP (aircraft tasks) and the mission order to check the feasibility and completeness SyP. If the SyP is feasible, the assistant core receives the start times and durations of the tasks that were calculated by the MMP, which is needed for monitoring the execution of already planned tasks. In case the SyP is incomplete (partial), the missing tasks will be added by the MMP, thereby assembling the aHuP, which allows the assistant system to monitor if the operator evolves the plan early enough to stay in schedule.

The *Free* instance of the MMP is responsible for the generation of the ReP. It receives only the mission order constraints, i.e. it is meant to disregard the SyP completely. Thereby, it checks if the problem is solvable in general, and in this case it generates a complete Free plan (i.e. the ReP), which can then be compared with the best scoring Slave plan (i.e. the aHuP) by the assistant system (see Evaluation chapter). This comparison reveals if the human operator inserted some elements into the SyP which might be suboptimal or even counterproductive and can therefore be used by the assistant system as basis for the decision whether to offer the ReP as the new SyP to the human.

Simple Temporal Constraint Interface

The STCI has been developed as an interface for the assistant system core to the MMP to transfer planning constraints. Each constraint refers either to a task to be performed or a state to be reached by an agent (i.e. aircraft or troops). State constraints are “be at ground position”, “be at air position” and task constraints include “transit”, “unload troops”, “load troops”, “cross FLOT”, “land”, “take off”, “recce route”, “recce area”, “object surveillance”. To generate multiple (and also open) time windows the temporal specifiers for constraints are “at beginning”, “at end”, “anytime”, “not before”, “not after”. The latter two are associated with a single time value. “Anytime” means the task has to be done or the state has to be reached at some unspecified point in time or interval. The specification of a closed or half-open interval is possible with the addition of “not before” and/or “not after” constraints. An “at end” constraint specifies a goal state for the planner (non-temporal) and “at-begin” constraints are needed to model tasks that are already in progress at the time of planning and therefore can be finished (before the agent starts executing any other task). The constraints can be specified as either hard (mandatory) or soft (associated with definable violation costs).

Implementation of the MMP

Although many mission planning systems with symbolic focus used in the aerospace domain are based upon the HTN (Hierarchical Task Network) knowledge-based planning approach, there is a reason for us to prefer a classical operator-based planner. An HTN planner is designed to explore different predefined possibilities of task decomposition and perform scheduling. This provides less flexibility compared to an operator-based planner, which is exploring combinations of atomic actions. Also, HTN planners have problems at planning for individual and interleaving actions for multiple agents (Goldman 2006). For example in the HTN-based Playbook™ Approach, a play (cooperative action of multiple UAVs) has to be defined before it can be invoked by the operator (Miller et al. 2004), which lowers flexibility. To find fine-granular solu-

tions of non-prescribed multi-agent cooperation in a central planning approach seems to be a strong advantage in the complex and dynamic environment of a MUM-T mission, although it poses a heavy burden on solution search performance. This article briefly shows that for the MUM-T scenario described above good solutions can be found sufficiently fast by the MMP in principle. Performance details can be found in (Strenzke and Schulte 2011b).

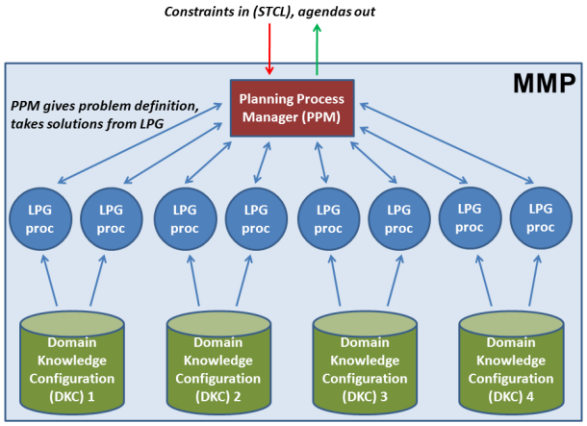


Figure 4: MMP internal structure and functionality

We chose the PDDL (Planning Domain Definition Language) 2.2 (Edelkamp and Hoffmann 2004) representation due to its temporal expressiveness. In our implementation, the assistant system sends plan requests and constraints via the STCI to the Planning Process Manager (PPM). The PPM then translates the constraints dynamically into a PDDL 2.2 problem definition and starts multiple planner processes, which work upon this problem and use different static Domain Knowledge Configurations (DKCs) (see figure 4). The DKCs contains slightly varying MUM-T world models in order to try out different problem-solving heuristics. The generated plans (task agendas) are finally collected by the PPM and provided to the assistant system.

Planning Process Manager

As soon as the PPM receives a planning command via the STCI, it dynamically generates a PDDL problem file containing the complete MUM-T problem with all aircraft. The current tactical situation, which includes all vehicle data (type, position, state) and all mission-relevant locations, is used as the initial state for the problem description. Also, all distances between the locations are calculated and set as numerical values (i.e. PDDL functions). As mentioned before, the human operator guides the UAVs on a task-based level, i.e. he/she provides tasks to the individual UAVs. Each task can be seen as a declaration of the operator's intent. Hence, these operator-given UAV tasks constitute constraints to the further planning process in addition to the externally given mission order when generating the aHuP. These and the constraints from the mission order are processed as follows.

The conversion of hard temporal constraints into PDDL works with timed initial literals (Edelkamp and Hoffmann 2004) in combination with denying or allowing preconditions for actions defined in the domain. E.g. if a constraint states that the takeoff of a specific aircraft from a specific location is allowed only after 10:00 ("not before" constraint), then via a timed initial literal at 10:00 a predicate "takeoff_denied" for this aircraft at this airport becomes false, which is a precondition for the "takeoff" action.

The hard "at end"- and "anytime"-constraints are directly translated into goal states for the PDDL planner (see figure 5). In case of task constraints ("anytime"), a post-condition is defined for the action corresponding to the task, which leads to the fulfillment of the predicate contained in the goal state upon action execution. Soft constraints are realized via a benefit that is calculated into the total costs of the solution in case the constraint is met (e.g. normally the costs for landing are zero, but if landing is preferred at a specific location then the cost function for this location is set to a negative value). Unfortunately it is not possible to generate soft temporal ("not before"/"not after") constraints with the current implementation.

```
(:goal
 (and
  (landed HELO)
  (ac_pos HELO MOB)
  (landed UAV1)
  (ac_pos UAV1 MOB)
  (landed UAV2)
  (ac_pos UAV2 MOB)
  (landed UAV3)
  (ac_pos UAV3 MOB)
  (troop_pos LEADER_SQUAD_A TARGET)
  (location_cleared_by ISAR2 UAV1)
  (section_cleared_by HOA_ENTRY ISAR2 UAV1)
  (flot_crossed_by ZULU_FRIEND ZULU_FOE UAV1)
  ...
 )
)
```

Mission order constraints

UAV task constraints / user constraints

Figure 5: PDDL goals example including constraint sources

Planning Engine

As planning engine we use the PDDL 2.2 compatible LPG-td (*L*ocal Search for *P*lanning *G*raphs – *t*imed initial literals and *d*erived predicates) 1.0 (Gerevini, Saetti, and Serina 2004) due to its full support of temporal planning capabilities defined in PDDL 2.2 and its good performance. Also, due to its local search algorithm, it seems to work well with giving it the SyP as a goal chain that will be followed strictly by the resulting plan (i.e. each next UAV action fulfills one goal in the chain). The planner is used in *best-quality* mode, i.e. it incrementally puts out the best plan found so far (evaluated by cost minimization constraint) until there has been provided a new planning request by the assistant system. The different LPG planning processes

that are set off by the PPM perform their search each with a distinct initial random seed. Each MMP instance uses 12 LPG processes in the current setup.

Domain Definition

The PDDL domain definition includes the description of object types, predicates, functions and actions. In the MUM-T world model there are *location*, *aircraft* and *troop* objects. In total, around 60 predicates and functions have been defined for the locations and their interconnections (in order to allow coarse route planning) and the description of the agents (aircraft, troops), e.g. location, speed etc.

All tasks that can be assigned via the task-based UAV guidance interface are represented as durative actions. Like the UAVs, the helicopter is of the aircraft object type but it is excluded from reconnaissance and surveillance actions. However, it has additional abilities, i.e. the loading and unloading of troops. Some tasks need multiple action models to cover different situations (e.g. "finish departure" as a special case of "departure" when this task is already in progress while starting the planner). This results in 30 different durative actions implemented in total.

The MMP works with multiple PDDL domain configurations in parallel in order to favor different heuristics. E.g. one DKC contains the additional very costly action "land at unrecc'd site", allowing a solution including this action in principle. Because not all DKCs include this action and the pool of LPG processes is fed with the different DKCs in equal amounts (cf. figure 4), certain effort is spent on the search for solutions excluding this costly action per se.

All costs can be implemented via functions in PDDL and therefore need not to be part of the domain model but can be generated dynamically in the problem file. The cost model is still being tuned to satisfy test persons' needs and optimize MMP as well as assistance system performance, therefore we do not present any numbers here.

Evaluation of the MMP

In this chapter we give preliminary evaluation results from the first experimental campaign that includes the MUM-T assistant system and the MMP. Then, some facts about typical problem sizes are provided and an evaluation method for assistant system decisions based on data generated by the MMP is outlined. More detailed data about the subjective and objective MMP evaluation can be found in (Strenzke and Schulte 2011b).

Subjective Evaluation

The test persons were four German Army helicopter pilots acting as UAV operator, each solving two training missions with assistance system aid, then solving two experimental missions without, and then two experimental missions again with assistance system aid. All of the missions were different but following the scheme described above. Subjective questionnaires about the MUM-T assistant system delivered the following results, which show a slightly positive trend, but also improvement potential:

- Slightly better efficiency through automated task insertion by the assistant system (when automated task execution was turned off)
- Proposals to insert tasks lowered workload slightly
- Proposals to insert tasks were considered rather useful
- Proposals to insert tasks seemed rather necessary

Objective Evaluation

The most critical situation for the MMP in terms of performance is the early phase of the mission after the operator has entered the UAV tasks into the system. In this situation the longest action sequence has to be generated in order to accomplish the mission and bring the aircraft back home again. Some rounded benchmark data for this case are given below:

- 35 tasks to be given to the UAVs to fulfill mission
- 600 facts about the 25 locations and their relations
- 45 facts about the 5 agents
- 35 timed initial literals
- 50 goal predicates (Slave MMP)
- 75 action steps in the solution
- 20-30 seconds[1] to find a satisfactory plan (viewpoint of developer)

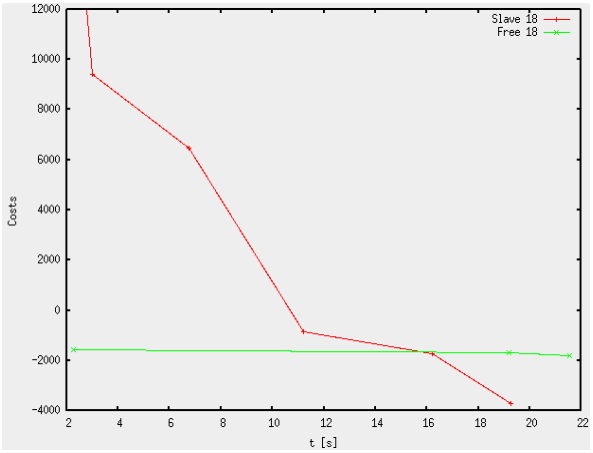


Figure 6: Example of a good aHuP (Slave plan, red)

To analyze situations of false alarms as well as actions missed by the assistant system, it is interesting to compare the plan quality generated over time by the Free and the Slave MMP instances because a future version of the assistant system will perform exactly this comparison in order to decide whether to take initiative to propose not only a

[1] Each MMP instance is running on a high-performance PC with 6 hyper-threading processors (i.e. 12 virtual processors, 1 per LPG process).

single additional UAV task but a completely new mission plan. At the moment this is only done in special use cases. Two thresholds can be set in order to tune the decision process: the time to wait until a decision is made and the cost difference between the Free and the Slave plan. Figure 6 shows an example of a good Slave plan (aHuP) beats the Free plan (ReP) after 16 seconds of incremental planning. This means that the human planning heuristics were more effective than those of the machine. Because of the changes in the situation (e.g. aircraft moving, threat blocking primary corridor) and the goals (e.g. follow-up mission), it is not possible to compare the aHuP against any baseline or against the optimal solution because it is not known. Therefore, it is necessary to analyze these graphs in certain mission situations, where re-planning should be proposed by the assistance system. The threshold values could be set to start checking for a cost difference of e.g. 10.000 after waiting 20 seconds after starting both planners (see figure 7). This analysis process is not yet completed.

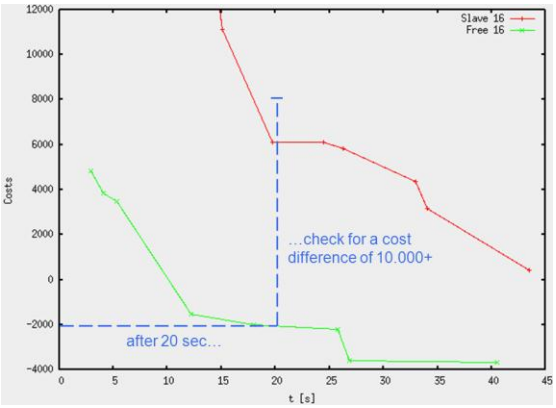


Figure 7: Applying time and cost thresholds to compare quality

A Critical View on the MMP Implementation

Our Cognitive Skill Merging approach to mixed-initiative has the goal to combine general human strengths and generals machine strengths in order to optimize overall human-machine system performance and compensate each other's weaknesses (Strenzke and Schulte 2011a). However, we see weak points in our implementation that are in conflict with the concept of this approach.

The first is the uninformed search concerning route plan generation. Routes that are longer than necessary can become optimized through the incremental mode of the LPG over time. But these optimizations are associated with relatively small costs (e.g. in comparison to landing at threatened site) and therefore can lead to a time-intensive optimization process, while the sub-optimality of the route is easily visible for the human operator. From the human-automation integration standpoint one would think that the route planning is a machine's strength and not a weakness. The missing of explicit geometrical planning and reasoning leads also to problems concerning reconnaissance coverage optimization, which is an important issue for reconnaissance UAV mission planning.

Another weak point of the MMP is the lack of continuous planning. This means that plan fragments that have already proven to be useful are not re-used. Instead every planning request by the assistant system makes the MMP generate a completely new plan (however, certain fragments will reoccur due to the constraints the MMP receives). On the one hand, this leads to the problem that the operator can be confronted with a new machine plan that differs in many aspects from the previous one, which can cause confusion. However, this problem arises rarely in our current configuration because the Slave planner regards the human input as hard constraints, and therefore all aHuPs overlap in all tasks that these constraints refer to. On the other hand, the plan optimization process is interrupted and reset very often, even if there are only minor changes to the problem to solve. Hence, it is difficult to maintain or improve plan quality in the long term. This problem could be addressed by remembering constraints that improved the solution and re-applying them. But this leads into the problem of having to try out hard constraint combinations.

A further problem is associated with the plan feasibility checking feature of the MMP. Because the search of the LPG does not terminate in case only temporal constraints deny the solution and there is no other possibility than to define these constraints as hard, the only workaround is to set a timeout concerning the waiting for planner output. To relieve the problem a little, one of the twelve LPG processes is fed with an "emergency plan" problem file with increased helicopter travel speed (near helicopter v_{max}).

One drawback of operator-based planning in comparison to HTN planning also is that *critical decision points* (e.g. which corridor, which drop zone) are rather implicitly modeled and "lost" in combinatorial space. Test persons reported that they do not tend so much to plan hierarchically. Instead they liked our forward planning style interface. However, they are indeed used to plan their missions by means of suchlike critical decision points.

Future Work

Figure 8 visualizes two main trends in the development of planners today, which can be seen as the generalization and flexibilization on the one hand (e.g. triggered by the International Planning Competition's current focus on domain-independent planning). On the other hand, many real-world applications need not only the flexibility of a planning engine but also performance, which is why Hierarchical Task Network (HTN) planning is used widely in real-world applications (Nau et al. 2005). As depicted in our Sigma-

Delta scheme (figure 8), such domain-configurable planners usually lack the flexibility of classical, domain-independent planning. We explained before, that this was one reason for choosing a domain-independent planner. For planning more complex missions during human-in-the-loop experiments with an acceptable response time, we consider a hybrid approach of classical operator-based and HTN planning, which would be similar to (Estlin, Chien, and Wang 1997; Biundo and Schattenberg 2001; Castillo, Fernández-Olivares, and González 2001), who set a trend towards efficient planning in a dynamic and unforeseeable real world. Further steps that alleviate real-world planning problems, like portfolio-based planning (Gerevini, Saetti, and Vallati 2009) or the situation-dependent assemblage of algorithms and a modular planner (Jameson et al. 2005), will not be taken in the near future.

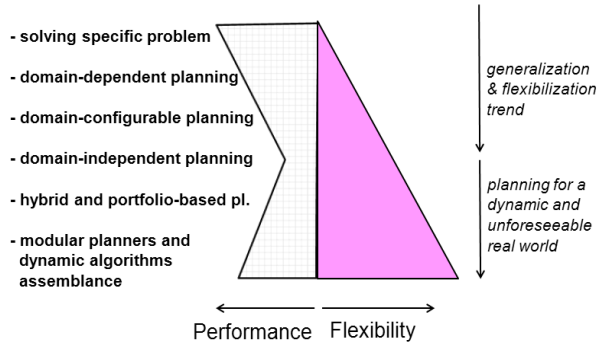


Figure 8: Planning approaches: Trends, performance, flexibility

Our future human-machine interaction research will address brittle (suboptimal) machine solutions (plans and advices) (Strenzke and Schulte 2011b). We will also regard different interaction timing configurations, i.e. the threshold concerning the cost difference between the assumed human plan and the reference plan can be lowered to increase and antedate automation-initiative or raised to make the assistant system intervention less intrusive. Further human-in-the-loop experiments to prove the concept of the MMP and investigate the above mentioned research topics are planned for the near future.

References

Biundo, S., and Schattenberg, B. 2001. From abstract crisis to concrete relief – A preliminary report on combining state abstraction and HTN planning. In *Proceedings of the 6th European Conference on Planning*.

Castillo, L.; Fernández-Olivares, J.; and González, A. 2001. On the adequacy of hierarchical planning characteristics for real-world problem solving. In *Proceedings of the 6th European Conference on Planning*.

Donath, D.; Rauschert, A.; and Schulte, A. 2010. Cognitive assistant system concept for multi-UAV guidance using human operator behaviour models. In *HUMOUS'10*, Toulouse, France.

Edelkamp, S., and Hoffmann, J. 2004. PDDL2.2: The Language for the Classical Part of the 4th International Planning Competition. In *Technical Report 195, Albert-Ludwigs-Universität Freiburg, Institut für Informatik*, Germany.

Estlin, T.A.; Chien, S.A.; and Wang, X. 1997. An argument for a hybrid HTN/operatorbased approach to planning. In *Proceedings of the 4th European Conference on Planning*.

Ferguson, G., and Allen, J. 1998. TRIPS: An Intelligent Integrated Problem-Solving Assistant. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, Madison, WI.

Funk, H.; Goldman, R.; Miller, C.; Meisner, J.; and Wu, P. 2005. A Playbook™ for Real-Time, Closed-Loop Control. In *Proceedings of the First International Conference on Computational Cultural Dynamics*, August 27-28; College Park, MD.

Gerevini, A.; Saetti, A.; and Serina, I. 2004. Planning in PDDL2.2 Domains with LPG-TD. In *Proceedings of ICAPS-04.*

Gerevini, A.; Saetti, A.; and Vallati, M. 2009. An Automatically Configurable Portfolio-based Planner with Macro-actions. In *Proceedings of ICAPS-2009*.

Goldman, R. 2006. Durative planning in HTNs. In *Proceedings of ICAPS-06.*

Jameson, S.; Franke, J.; Szczerba, R.; and Stockdale, S. 2005. Collaborative Autonomy for Manned/Unmanned Teams. In *Proceedings of the American Helicopter Society 61th Annual Forum.*

Miller, C., Funk, H., Wu, P., Goldman, R., Meisner, J., Chapman, M. 2005. The Playbook Approach to Adaptive Automation. In *Proceedings of the Human Factors and Ergonomics Society's 49th Annual Meeting.* Orlando, FL.

Miller, C.; Goldman, R.; Funk, H.; Wu, P.; and Pate, B. 2004. A Playbook Approach to Variable Autonomy Control: Application for Control of Multiple, Heterogeneous Unmanned Air Vehicles. In *Annual Meeting of the American Helicopter Society*.

Nau, D.; Au, T.-C.; Ilghami, O.; Kuter, U.; Wu, D.; Yaman, F.; Munoz-Avila, H.; and Murdock, J.W. 2005. Applications of SHOP and SHOP2. In *Intelligent Systems*, IEEE, Vol. 20 Issue: 2.

Onken, R., and Schulte, A. 2010. *System-ergonomic Design of Cognitive Automation: Dual-Mode Cognitive Design of Vehicle Guidance and Control Work.* Heidelberg, Germany: Springer.

Strenzke, R., and Schulte, A. 2011(a). Mixed-Initiative Multi-UAV Mission Planning by Merging Human and Machine Cognitive Skills. In *8th Conference on Engineering Psychology & Cognitive Ergonomics*, in conjunction with HCI International.

Strenzke, R., and Schulte, A. 2011(b). Design and Evaluation of a Mixed-Initiative Multi-Vehicle Mission Planning System. In *IJCAI-11 Workshop AI in Space: Intelligence beyond Planet Earth*. Barcelona, Spain. 17 July 2011.

Strenzke, R.; Uhrmann, J; Benzler, A.; Maiwald, F.; Rauschert, A.; and Schulte, A. 2011. Managing Cockpit Crew Excess Task Load in Military Manned-Unmanned Teaming Missions by Dual-Mode Cognitive Automation Approaches. In: *AIAA Guidance, Navigation, and Control (GNC) Conference*. Portland, Oregon.

Uhrmann, J.; Strenzke, R.; and Schulte, A. 2010. Task-based Guidance of Multiple Detached Unmanned Sensor Platforms in Military Helicopter Operations. In *COGIS*, Crawley, UK.

Planning for Human-Robot Teaming

Kartik Talamadupula[†] and **Subbarao Kambhampati**[†] and **Paul Schermerhorn**[§] and **J. Benton**[†] and **Matthias Scheutz**[§]

[†]**Department of Computer Science**
Arizona State University
Tempe, AZ 85287 USA
{krt,rao, j.benton} @ asu.edu

[§]**Cognitive Science Program**
Indiana University
Bloomington, IN 47406 USA
{pscherme,mscheutz} @ indiana.edu

Abstract

One of the most important applications of planning technology has been – and continues to be – guiding robotic agents in an autonomous fashion through complex problem scenarios. Increasingly, real-world scenarios are evolving in a way that includes humans as actors in the loop along with the robot and the planning system. These humans are stakeholders whose roles may vary between that of a commander or a system or domain expert; the one common thread is that together with the robot, they form a team that shares common goals. In this paper, we consider challenges posed by such *human-robot teaming* scenarios from a purely planning-centric perspective, and discuss the dimensions of variation within application problems in such scenarios. We seek to differentiate planning for human-robot teaming from the general area of human-robot interaction, since we are mainly interested in the planning tools that facilitate such teaming. We look at some problems that are encountered in deploying existing planning techniques in such teaming scenarios, and illustrate these with our experience in a real world search and rescue scenario. We follow this up with results from runs involving a robot controlled by a planner whose goal handling capabilities are augmented.

Introduction

One of the earliest motivations for Artificial Intelligence as a field of study was to provide autonomous control to robotic agents that carry out useful service tasks. Application scenarios for these kinds of tasks span a wide spectrum that includes military drones and mules, household assistance agents (Goebelbecker et al. 2010) and search and rescue robots (Schermerhorn et al. 2009). The concept of *teaming* between humans and robots is central to all these applications – the notion of robotic agents that support a human agent's goals while executing autonomously is a recurring theme. The level of autonomy that is desired of these robotic agents is often achievable only by integrating them with planning systems that can not only plan for the initially specified goals, but also updates to these goals as well as changes to the world and to the agent's capabilities.

Recent years have seen the emergence of fast planning algorithms and systems that can account for a large number of the features that distinguish a real world application from a theoretical scenario – time, cost, resources, uncertainty and execution failure. Though planners of the past have been able to model many of these features (Penberthy and Weld 1995), the scale-up that is required to support real world time windows has only come about in the past decade due to the use of heuristic search methods for plan synthesis. Current planners still operate under a number of restrictive assumptions, and classical planners like LAMA (Richter and Westphal 2010) are clearly the fastest of the lot. The challenge then is one of identifying the features that are essential when considering planning support for such joint human-robot endeavors, and of providing a general framework for these problems. This problem is quite distinct from the existing field of human-robot interaction (HRI), since we are interested more in what existing planning techniques can be used or extended in order to facilitate teaming scenarios. Towards this end, we discuss a new class of problems under the collective term *human-robot teaming* (HRT), and present the essential dimensions of such problems with respect to planning. The teaming aspect of these problems arises from the fact that the human and the robot are both acting towards achieving the same set of shared goals, and the relationship between them can be defined in terms of known modes of interactions in teams (e.g. colleagues, commander-subordinate, etc.). Though there has been work in the past on the intersection of tasks involving humans, robots and planners, most of that work has concentrated on a system-centric view of the interaction. Our focus in this paper is instead on the teaming, and on describing the characteristics of this problem as applicable to planning.

The rest of this paper is organized as follows: we first discuss the concept of human-robot teaming and list the dimensions of interest to planning in such scenarios. Following this, we look at a search and rescue scenario that we had to provide planning support for as a case study, and place it within the HRT spectrum. We then detail some initial work that we have undertaken in tackling some planning challenges inherent in teaming scenarios. We discuss two specific problems – handling incomplete models, and the problem of goal specification and revision. As part of the latter, we also detail our work with a robot executing in a real-world search and rescue scenario, and present the aggregated results of the robot's runs through this task guided by our planning system. Our hope is for this paper to serve as a catalyst that spurs the planning community into further defining and mapping the application-rich field of human-

robot teaming, and the specific planning challenges that lie in this area.

Human-Robot Teaming

In this paper, we focus our attention towards the planning challenges that must be tackled in order to support human-robot teaming scenarios. Our motivation in pushing this problem is to present a larger class of applications that the planning community can provide tightly-knit support for, and to open the door to discussions on the nature of such support. Our experience with this problem stems from our work with a search and rescue scenario (detailed in the next section) and the planner extensions that were required in order to support tasks from that scenario. In later sections, we detail some of these extensions along with references to more detailed work.

We begin by clarifying the nomenclature: *human-robot teaming* scenarios are those involving (possibly multiple) human and robotic agents that acquire their "teaming" nature from the autonomous behavior of the robotic agent(s). Though it is assumed that the top-level goals are determined and specified by the human, the robot is completely autonomous in that it only receives a set of goals that it must accomplish within some specified constraints while respecting the notion of optimizing some pre-defined metric. For the robot to exhibit this autonomy, it is imperative that the system be equipped with a planner that can handle the various dimensions of the environment that the team must operate in. We briefly describe these dimensions and the challenges in supporting them here, as a list of the defining components of a human-robot teaming (HRT) scenario:

Scenario One of the most important factors in HRT is the problem scenario in which the team is executing – here, we use "scenario" in a ubiquitous sense to describe the particular task or collection of tasks at hand that the team is interested in solving. More often than not, these scenarios are very close approximations of real-world applications in which essential tasks were carried out by humans before the advent of the human-robot team. As such, there is almost always a large amount of domain knowledge available about the scenario that may be exploited – such knowledge resides either with humans who have been actors in the scenario previously, or in carefully compiled manuals and technical documents. Taking the scenario into consideration when planning for a human-robot teaming problem is key, since it determines the kinds of tasks that must be supported. The scenario also determines the kind of features a planner must support in order to guide the robot in pursuit of the team's goals; we elucidate on this in subsequent sections.

Robot The robot is the central actor in an HRT scenario, since it has upon itself the responsibility of executing all the actions and gathering sensory feedback from the world to relay back to the human team member and the team's repository of knowledge. Teaming must account for the fact that there exist various types of robots with varying capacities, and that the type of robot in use may change according to the scenario. A planning system that is providing support in such HRT scenarios must be able to deal with robots with different capabilities (for e.g., mobile robots versus grippers) and must take into account the constraints that arise. The model of a robot's capabilities determines the actions that may be used in fulfilment of the scenario's top-level goals, and hence this is an imperative feature when evaluating an HRT scenario.

Human (User) The human user is a key player in an HRT scenario, since the robot (and consequently the planner) are often achieving goals specified by the user. These users can belong to one of three general categories, contingent on their level of familiarity with the robot's working and the overall integrated system in use:

1. Novice: A novice user is one who does not understand the intricacies of the system (representation, component interaction etc.) and is merely using the robot (and the system) as an assistant. An example of a scenario featuring such a novice user might be a robot designed to assist the elderly at home – in such scenarios, allowances have to be made in the system for information that may be incompletely or wrongly specified by running extra validation via the robot.
2. Domain Expert: A domain expert is a user who is an authority on the environment that the robot is executing in, and is in a position to pass on new goals and information to the robot as they become available. Such a user if often the dominant force in the human-robot team, and it can generally be assumed that knowledge gleaned from this user is reliable and may be acted upon without further deliberation. Examples of such users include commanders in military or rescue teams who have a very good understanding of their environment, yet may not be at all familiar with the system's internal representations.
3. System Expert: A user who is a system expert exhibits a high degree of familiarity with the integrated system that manages the robot – its setup, the representation scheme used, and its various capabilities and shortcomings. Such a user is often the person that set up the system integration in the first place (or someone who is debugging it); as such, examples from real-world scenarios may be researchers and programmers who are in-charge of maintaining the overall system. System experts may also be domain experts, but this is not necessarily always the case.

Model Management The model is the system's internal representation of the dynamics of the environment and world that it must handle. Various components within the system may have different models at different levels of detail; for example, the planner may maintain a PDDL model of the system, while lower level components like the path-finding apparatus may use a grid-like approximation of the world. On a higher level, it is almost always the case that the robotic and human actors in the team have different versions of the world model, and one of the planner's responsibilities might be to bridge this gap and produce plans that will succeed upon execution. The model is usually encoded before the scenario begins, but may also be provided piecemeal by the human component of the team. Learning is another possibil-

ity – though we do not currently support it, learning certain characteristics and parameters of the environment automatically through the robot is a promising direction for future work.

Goal Management Distinct from the notion of encoding and using the world model is the idea of specifying and updating goals. The very utility of a human-robot teaming scenario stems from the assumption that smaller goals will be specified by a human in the pursuit of some overarching purpose, and that the robot will autonomously perform actions in the world that will lead to the fulfilment of those goals. In such scenarios, it is imperative that the definition of "goal" be expanded so that the user may specify different kinds of goals (and possibly plan trajectory constraints as goals (Baral and Zhao 2007)). Since the responsibility of planning for robot's actions falls on the planner, providing means to specify new types of goals is a key planning challenge in such scenarios. Flexibility in goal specification includes supporting ideas like goals with different priorities (using rewards), goals that need not be achieved (using soft goals) and goals that condition on currently unknown facts – and just as important, the ability to specify changes to goals on the fly. We talk about this problem and the work we have done in this direction in detail in a later section.

Communication Given that the team is composed of separate human and robotic components, communication is a non-trivial issue, and is quite often the bottleneck in specifying and achieving goals optimally. Communication affects everything in the scenario, from the specification and modification of the domain model and goals, to the human user's knowledge of the evolution of the world – which may sometimes only be via the robot. The problem with respect to communication is unifying the various representations used by different components in a manner that minimizes information loss and processing time. For example, a commander may only specify information using natural language, but the system must translate this into a form suitable for the planner (and robot) to use; the system must also convey the results of action execution and goal achievement (whether failure or success) to the commander.

Related Work

Though there has been no prior work that directly addresses the problem of planning for human-robot teaming, there is definitely a large volume of work that is related to various aspects of this problem. As shown in Figure 1, previous work can be classified into three parts – human-robot interaction, human-planner interaction and planner-robot interaction. More specifically:

- Planning and execution monitoring deals with the interactions between a fully autonomous robot and a planner.
- Human-robot interaction (HRI) works toward smooth interactions between a human user and a robot.
- Mixed initiative planning relates to interactions between humans who are receiving plans and the automated planners that generate them.

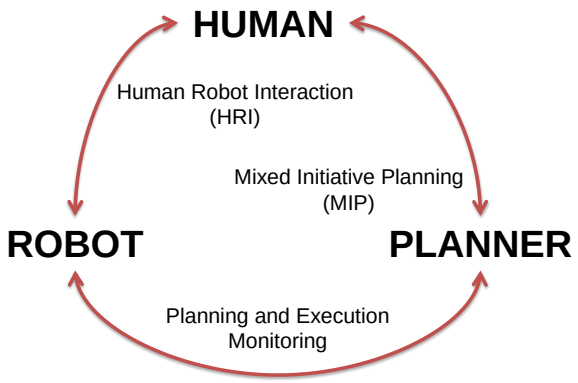


Figure 1: The various modes of interaction in human-robot scenarios.

Since the focus of this work is on providing planning support for human-robot teams, the most interesting work is that which relates planning and execution monitoring to mixed initiative planning. A lot of work has been done in both these areas, and their intersection; the closest work seems to be Bagchi et al.'s (Bagchi, Biswas, and Kawamura 1996) system for controlling service robots. In their system, the robot is equipped to handle the user's changing goals and advice at different levels of detail via a planner that can refine and modify goals dynamically. There has also been work on how humans interact with planners, and how the process of accepting user input can be streamlined. In particular, work by Myers [1996, 1998] has dealt specifically with *advisable planning* that allows a human to specify partial plans, recommendations or methods to evaluate plan quality, all in natural language. Space precludes a detailed description of all past related work; the reader is directed to (Talamadupula et al. 2010a) for a complete listing.

However, planning for human-robot teaming tasks has received a significant amount of attention very recently as well. In particular, two sub-problems have seen a lot of interest. The first is the idea of using two (or more) distinct models during the planning process – a higher, more task-oriented model while trying to come up with actions that support end goals; and a lower-level model to decompose those tasks in tune with the capabilities of the robotic agent being used. The other idea that has received some attention has been that of robotic proactiveness, and the notion that a robot may "ask for help" if it (the planner) is unable to come up with a course of action to fulfil a particular goal. Both these problems have much scope for work, and there exists some work in the planning community currently under review that addresses them.

An HRT Case Study: Search and Rescue

One of the primary applications of human-robot teams is in scenarios where a human actor has a plethora of knowledge about the problem at hand, yet cannot act in the world due to inherent dangers to human life – emergency responders and firefighters are among the best examples of such hu-

Feature \ Task	Search and Report	Reconnaissance	Kitchen Robot
Robot	Mobile	Mobile	Mobile and Manipulator
Human (User)	Domain Expert	System Expert	Novice
Model	Less Dynamic	Dynamic	Highly Dynamic
Goals	Changing	Static	Changing
Communication	Natural Language	APIs	Natural Language

Table 1: The dimensions of a Human-Robot Teaming scenario illustrated for a few example tasks.

mans. In such cases, having a robotic agent as part of the team greatly increases the chances of achieving the desired end-goals (rescuing people, putting out a fire etc.) without exposing the human component of the team to risks.

In this paper, we use a specific scenario that we had to provide planning support for to illustrate the challenges that crop up when planning for human-robot teaming. This is the urban search and rescue (USAR) scenario – a team consisting of a human and a robot is tasked with finding and reporting the location of critical assets (e.g. injured humans) in an urban setting (usually a building). A given USAR scenario may consist of multiple problems, each with different challenges that depend on the end goals that must be satisfied - examples of such problem tasks are given in table 1, along with an analysis of the how the various HRT features apply to these tasks. The human member of the team usually has intimate knowledge of the setting of the scenario, but cannot perform the required tasks due to inherent dangers like fires, gas leaks, collapsed structures etc. Examples of tasks in the USAR scenario include transporting essential materials to a specified location or entity; and reconnaissance tasks like reporting the locations of trapped or injured humans to the commander and taking pictures of objects or areas of interest. In the following, we present two USAR tasks that are of particular interest to us as examples to illustrate the planning challenges that are inherent in human-robot teaming.

Search and Report

In this problem, the robot's main goal is to deliver essential medical supplies within the area of interest – during its run, the robot may be given additional information and goals about other assets. The human component of the team (the commander) has intimate knowledge of the building's layout, but is removed from the scene and can only interact with the robot via on-board wireless communication. The robot begins in a long hallway that has doors leading off into rooms on either side.

Initially, the robot is unaware that that these rooms may contain injured or trapped humans, and its goal is to reach the end of the hallway to deliver the supplies by a given deadline. As the robot executes a plan to achieve that goal, the human commander notes that it is passing the rooms and recalls that injured humans may be trapped in these rooms. The commander then passes on this information linking rooms to injured humans to the robot, and specifies a new goal on reporting the location of as many such humans as possible given the time and resource constraints imposed by the achievement of its original goal. In a succeeding section, we talk about the changes to goal specification that had to be handled to enable a planner to handle such a task, and present some past work on this application.

Reconnaissance

The other task that we touch on is based on a classic robotic application - reconnaissance. Quite often in urban settings, there arise situations where more information is needed about certain objects before making a decision, yet the cost or risk of obtaining that information manually (for humans) is too high. A sterling example is defusal of explosives; often, a closer look and more accurate imagery is required in order to determine whether an unrecognized objects is dangerous and should be disposed off in a suitable manner. Given the high costs of adverse events in such scenarios, it is much more preferable to have robotic agents do the close-up examination required. The challenge in these scenarios is to support changes to the model of the robotic agent's capabilities as conveyed to the planner – it is entirely possible that given a new task like taking a zoomed-in picture of a suspicious box, the commander may either give the robot new effectors that accomplish such a task, or may simply specify a way of using the robot's existing capabilities to simulate such an effect. These are changes to the model that the planner is planning with, and hence need to be accomodated to enable goal achievement.

Incomplete Models

We now turn our attention to the first of two important planning challenges that arise when supporting human-robot teaming scenarios, that of changes to the model while the agent is executing in the world. Take the example of the first task in the USAR scenario – search and report – where the human commander is removed from the scene due to the inherent dangers of the situation. The agent thus needs to act in an autonomous manner to achieve the goals prescribed to it. To this end, the agent follows a domain theory that is provided by a domain expert; however, *updates* to this domain may be specified while the agent is executing a plan in the world. Updates to the agent's knowledge of the world may also arrive in tandem with execution. In such circumstances, two things are of essence: first, we need a representation for *specifying* such updates and integrating them into the knowledge base of the planner that is guiding the agent. Subsequent to this, the problem changes to one of *reasoning* about the changes and their effect on the current plan's validity and metrics. Replanning from scratch is a trivial approach – however, with sophisticated update methods and reward models, such a method can account for com-

Figure 2: A map of a sample search and report scenario; boxes in rooms are stand-ins for humans, where green (at left) indicates injured and blue (at right) indicates normal.

mitments (Cushing, Benton, and Kambhampati 2008). Nevertheless, such methods ignore the fact that many changes are localized to a certain portion of the domain and may not require the (often expensive) re-computation of a new plan.

As automated planning systems move into a support role for real world problems that involve such teaming, the problem of incompletely or incorrectly specified domain theories is a recurring one. The shortcomings associated with not having a completely specified domain theory manifest themselves as reduced robustness in synthesized plans, and subsequent failures during the execution of such plans in the world. One way of dealing with such contingencies is to employ a reactive approach that replans from scratch (as mentioned above) – however, this approach will fail if there are parts of the domain model that are never revealed to the system (and planner) at all. Consider for example the act of opening doors that are locked, yet the planner's model does not support the notion of locks on doors. A reactive planner would keep trying an 'open' action with no success, and very little information in terms of why the action was failing.

More generally, it is the case in many scenarios that though plan synthesis is performed using a rudimentary domain model and less than complete information about the world, there are domain experts who specify changes to the specific problem instance and sometimes the domain model itself during the planning process. Quite often it is useful to take this new information into account, since it may help prevent execution failures when the plan is put into action. Additionally, new information about the domain or the problem may open up new ways of achieving the goals specified, thus resulting in better plan quality as well as more robust plans. Given the progress in automated planning technology and a planning system that can be engineered to handle such changes, it would be wasteful to stick to reactionary planning and not exploit the plan improvements that are possible when changes in the model are taken into account.

Model Management

As described above, model updates are a reality that many real world integrated systems have to contend with. When viewed through the prism of the search and rescue scenario that we support, the process of updating these models breaks down into one of two kinds of tasks:

1. **Model Maintenance**: This is a more complete solution to the problem of model updates, founded in the formal representation of domain models. The expectation in this kind of update is that the domain expert will specify changes to the model that will ultimately be in the same format that is used for representation. In this type of update, post-update consistency of the model is a trivial issue since the specification mechanisms will ensure that only permissible updates can be specified. This kind of update is suited to scenarios where the planner is part of a larger integrated system and communicates via some kind of API that supports the planner's internal representation.
2. **Model Revision**: This approach seeks to incorporate changes that are specified in a representation that is less formal than the one used by the planner. An example of such a scenario would be a human commander specifying (via natural language) that there have been certain changes to the problem at hand – the planner needs to make the best it can of these updates and change the current plan accordingly. Note that it is significantly harder to offer guarantees about goal achievement, plan validity, or model consistency with this approach.

In the context of our current work, the first scenario is more relevant - even though the updates to the domain are specified through natural language, the planner only receives them via an established API that allows interaction with the the agent architecture. However, the bigger point is that the the kinds of update tasks that one has to consider in these kinds of scenario are really a manifestation of the agents in the system and the architecture underlying it. Most of the changes to the current world state can be described as part of a specific problem description and changes to it. However, there is another kind of update that is possible; an update to the domain model. It is quite unlikely that these kinds of updates are "discovered" as changes to the world; the more likely eventuality is that such updates are specified to the planner by a domain expert – perhaps even the person who crafted the domain in the first place. Domain design is not an

Run	Cost	Reward	Soft	Enter R_1	Report GB	Enter R_2	Report BB	Enter R_3
1	+	+	+	Yes	Yes	No	No	No
2	+	+	-	Yes	Yes	$\perp$	$\perp$	$\perp$
3	+	-	+	No	No	No	No	No
4	+	-	-	Yes	Yes	$\perp$	$\perp$	$\perp$
5	-	+	+	Yes	Yes	No	No	No
6	-	+	-	Yes	Yes	$\perp$	$\perp$	$\perp$
7	-	-	+	No	No	No	No	No
8	-	-	-	Yes	Yes	$\perp$	$\perp$	$\perp$

Table 2: Results of trial runs with a deadline of 100 time units. $\perp$ denotes that there is no feasible plan from that point on that fulfils all hard goals.

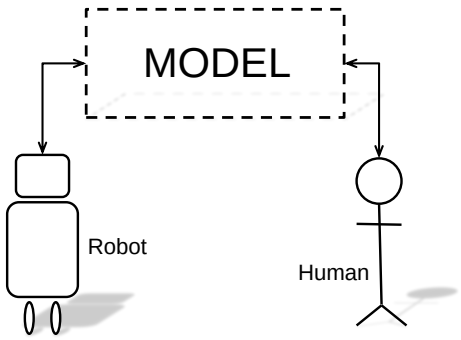


Figure 3: The robot and the human user share a model, but have different perceptions of that model.

exact science, and creating even an approximation of a real world model is fraught with errors of omission and commission. However, most domains are designed such that the first few versions are never completely off-base. Very rarely is there a need to change a significant percentage of a domain model, and more often than not, changes are updates to small portions of the description.

This is especially true in human-robot teaming scenarios like search and rescue – (human) domain experts are more likely to provide additional information that is relevant to the immediate tasks that are being performed. In terms of symbolic planning, this translates into the operators that are currently being executed as part of the overall plan. In such scenarios, it makes more sense to provide a way of updating the existing domain description and the plan that is currently executing than to throw out all the search effort and replan from scratch, since the changes to the domain may not affect a significant portion of the plan. In addition, this kind of approach is preferable even in scenarios where domain descriptions are learned (and updated) automatically through repeated planning and execution episodes.

Goal and Knowledge Revision

Along with information about the model, the other important user-specified information that the robot (and planner) must deal with is the specification of the goals that must be achieved in a given task. This goal specification may the actual goals to be achieved, as well as the values of achieving such goals, and priorities and deadlines (if any) associated with these goals. The fact that the system's goals are determined and specified by the human in the loop also introduces the possibility that goals may be specified incompletely or incorrectly at the beginning of the scenario. Such a contingency mandates a need for a method via which goals, and the knowledge that is instrumental in achieving them, can be updated.

The biggest planning challenge when it comes to the problem of goal update and revision is that most state-of-the-art planning systems today assume a "closed world" (Etzioni, Golden, and Weld 1997). Specifically, planning systems expect full knowledge of the initial state, and expect up-front specification of all goals. Adapting them to handle the "open worlds" that are inherent in real-world scenarios presents many challenges. The open world manifests itself in the system's incomplete knowledge of the problem at hand; for example, in the search and report scenario, neither the human nor the robot know where the injured humans may be. Thus an immediate ramification of the open world is that goals may often be conditioned on particular facts whose truth values may be unknown at the initial state. For example, the most critical goal in the USAR scenario – reporting the location of injured humans – is conditioned on finding injured humans in the first place. In this section, we describe recent work on bridging the open nature of the world with the closed world representation of the planner that has been done in the context of the USAR problem.

Open World Quantified Goals

Open world quantified goals (OWQG) (Talamadupula et al. 2010b) combine information about objects that *may be* discovered during execution with partial satisfaction aspects of the problem. Using an OWQG, the domain expert can furnish details about what new objects may be encountered through sensing and include goals that relate directly to the sensed objects. Newly discovered objects may enable the achievement of goals, granting the opportunity to pursue reward. For example, detecting a victim in a room will allow the robot to report the location of the victim (where reporting gives reward). Given that reward in our case is for each reported injured person, there exists a quantified goal that must be allowed partial satisfaction. In other words, the universal base, or total grounding of the quantified goal on the real

Run	Cost	Reward	Soft	Enter R_1	Report GB	Enter R_2	Report BB	Enter R_3
9	+	+	+	Yes	Yes	Yes	No	Yes
10	+	+	-	Yes	Yes	Yes	No	Yes
11	+	-	+	No	No	No	No	No
12	+	-	-	Yes	Yes	Yes	No	Yes
13	-	+	+	Yes	Yes	Yes	No	Yes
14	-	+	-	Yes	Yes	Yes	No	Yes
15	-	-	+	No	No	No	No	No
16	-	-	-	Yes	Yes	Yes	No	Yes

Table 3: Results of trial runs with a deadline of 200 time units.

world, may remain unsatisfied while its component terms may be satisfied.

As an example, we present an illustration from our scenario: the robot is directed to "report the location of all injured humans". This goal can be classified as open world, since it references objects that do not exist yet in the planner's object database; and it is quantified, since the robot's objective is to report *all* victims that it can find. In our syntax, this information is encoded as follows:

```
1 (:open
2   (forall ?z - zone
3     (sense ?hu - human
4      (looked_for ?hu ?z)
5       (and (has_property ?hu injured)
6            (in ?hu ?z))
7   (:goal (reported ?hu injured ?z)
8            [100] - soft))))
```

We evaluated the efficacy of OWQGs via experimental runs in the search and report scenario introduced earlier. We used the planner `SapaReplan` (Cushing, Benton, and Kambhampati 2008), an extension of the metric temporal planner `Sapa` (Do and Kambhampati 2003) in order to implement and test the OWQGs. The task at hand was the following: the robot is required to deliver essential supplies (which it is carrying) to the end of a long hallway – this is a hard goal. The hallway has doorways leading off into rooms on either side, a fact that is unknown to the robot initially. When the robot encounters a doorway, it must weigh (via the planner) the action costs and goal deadline (on the hard delivery goal) in deciding whether to pursue a search through the doorway. In the runs described here, green boxes acted as stand-ins for victims, whereas blue boxes denoted healthy people (whose locations need not be reported) as shown in figure 2. The experimental setup consisted of three rooms, which we represent as R_1, R_2 and R_3. The room R_1 contained a green box (GB), representing a victim; R_2 contained a blue box (BB), representing a healthy person; and R_3 did not contain a box. The respective doorways leading into the three rooms R_1 through R_3 are encountered in order as the robot traverses from the beginning of the hallway to its end.

To achieve our goal of demonstrating the use of the OWQGs, we conducted a set of experiments where we varied four parameters – each of which could take on one of two values – thus giving us 16 different experimental conditions through the scenario. The factors that we varied were:

1. **Hard Goal Deadline**: The hard goal deadline was fixed at 100 time units, resulting in the runs in Table 2, and 200 time units to give the runs in Table 3.
2. **Cost**: Presence or absence of action costs to demonstrate the inhibiting effect of costly sensing actions on the robot's search for injured people.
3. **Reward**: Presence or absence of a reward for reporting injured people in rooms.
4. **Goal Satisfaction**: Label the goal of reporting injured people as either soft or hard, thus modulating the bonus nature of such goals.

In the tables provided, a + symbol stands for the presence of a certain feature, while a - denotes its absence. For example, run number 5 from table 2 denotes an instance where the deadline on the hard goal (going to the end of the hallway) was 100 time units, action costs were absent, the open world goal of reporting people carried reward, and this goal was classified as soft.

The experimental runs detailed in this section were obtained on a Pioneer P3-AT robot as it navigated the USAR scenario with the initial hard goal of getting to the end of the hallway, while trying to accrue the maximum net benefit possible from the additional soft goal of reporting the location of injured people. The robot starts at the beginning of the hallway, and initially has a plan for getting to the end in fulfilment of the original hard goal. An update is sent to the planner whenever a doorway is discovered, and the planner subsequently replans to determine whether to enter that doorway. In the first set of runs, with a deadline of 100 units on being at the end of the hallway, the robot has time to enter only the first room, R_1 (before it must rush to the end of the hallway in order to make the deadline on the hard goal).

In spite of this restriction, the robot exhibits some interesting behavior. The planner directs the robot to enter R_1 in all the runs except 3 and 7 – this can be attributed to the fact that there is no reward on reporting injured people in those cases, and the reporting goal is soft; hence the planner does not consider it worthwhile to enter the room and simply ignores the goal on reporting. The alert reader may ask why it is not the case that entering R_1 is skipped in runs 4 and 8 as well, since there is no reward on reporting injured people in those cases either; however, it must be noted that this goal is hard in cases 4 and 8, and hence the planner *must* plan to achieve it (even though there may be no injured person in that room, or reward to offset the action cost). This exam-

ple illustrates the complex interaction between the various facets of a typical HRT scenario – the robot's capbilities, user-specified goals, and model parameters like costs and rewards.

In terms of computational performance, the planning time taken by the planning system was typically less than one second (on the order of a hundred milliseconds). Our empirical experience thus suggests that the planning process always ends in a specific, predictable time frame in this scenario—an important property when actions have temporal durations and goals have deadlines. Additionally, in order to test the scale-up of the system, we evaluated it on a problem instance with ten doors (and consequently more runtime objects) and found that there was no significant impact on the performance.

Conclusion

In this paper, we presented the problem of providing planning support for human-robot teaming scenarios, and outlined some of the prominent planning challenges that must be addressed in conjunction with this problem. We presented the Urban Search and Rescue scenario as a case study in which a planner actively supports a human-robot team, and showed the dimensions along which two tasks that are part of this scenario require planning support. Motivated by our applied work in this scenario, we delved deeper into two particular planning challenges – model management and dealing with incomplete and dynamic models, and the problem of goal and knowledge revision. In these sections, we outlined the extensions that had to be made to existing planning technology in order to fulfill the support role that the planner plays in the team. We concluded with a look at results from a search and report task from the USAR scenario where our planner was able to guide the robot's pursuit of fairly complex and dynamic goals.

Acknowledgements

We thank William Cushing for helpful discussions and the development of *Sapa Replan*. We also thank the anonymous reviewers for their suggestions which were incorporated in the final version of this paper. This research is supported in part by ONR grants N00014-09-1-0017 and N00014-07-1-1049, and the NSF grant IIS-0905672.

References

Bagchi, S.; Biswas, G.; and Kawamura, K. 1996. Interactive task planning under uncertainty and goal changes. *Robotics and Autonomous Systems* 18(1):157–167.

Baral, C., and Zhao, J. 2007. Non-monotonic temporal logics for goal specification. In *IJCAI*, volume 7, 236–242.

Cushing, W.; Benton, J.; and Kambhampati, S. 2008. Replanning as a Deliberative Re-selection of Objectives. *Arizona State University CSE Department TR*.

Do, M., and Kambhampati, S. 2003. Sapa: A multi-objective metric temporal planner. *Journal of Artificial Intelligence Research* 20(1):155–194.

Etzioni, O.; Golden, K.; and Weld, D. S. 1997. Sound and efficient closed-world reasoning for planning. *AIJ* 89(1-2):113–148.

Goebelbecker, M.; Keller, T.; Eyerich, P.; Brenner, M.; and Nebel, B. 2010. Coming up With Good Excuses: What to do When no Plan Can be Found.

Myers, K. 1996. Advisable planning systems. *Advanced Planning Technology* 206–209.

Myers, K. 1998. Towards a framework for continuous planning and execution. In *Proceedings of the AAAI Fall Symposium on Distributed Continual Planning*.

Penberthy, J., and Weld, D. 1995. Temporal planning with continuous change. In *Proceedings of the national conference on Artificial Intelligence*, 1010–1010. JOHN WILEY & SONS LTD.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research* 39(1):127–177.

Schermerhorn, P.; Benton, J.; Scheutz, M.; Talamadupula, K.; and Kambhampati, S. 2009. Finding and exploiting goal opportunities in real-time during plan execution. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*.

Talamadupula, K.; Benton, J.; Kambhampati, S.; Schermerhorn, P.; and Scheutz, M. 2010a. Planning for human-robot teaming in open worlds. *ACM Transactions on Intelligent Systems and Technology (TIST)* 1(2):14.

Talamadupula, K.; Benton, J.; Schermerhorn, P.; Scheutz, M.; and Kambhampati, S. 2010b. Integrating a Closed-World Planner with an Open-World Robot. In *AAAI 2010*.

Temporal Optimization Planning for Fleet Repositioning

Kevin Tierney and **Rune Møller Jensen**
IT University of Copenhagen
Rued Langgaards Vej 7, DK-2300, Copenhagen S, Denmark
{kevt,rmj}@itu.dk

Abstract

Fleet repositioning problems pose a high financial burden on shipping firms, but have received little attention in the literature, despite their high importance to the shipping industry. Fleet repositioning problems are characterized by chains of interacting activities, but state-of-the-art planning and scheduling techniques do not offer cost models that are rich enough to represent essential objectives of these problems. To this end, we introduce a novel framework called Temporal Optimization Planning (TOP). TOP uses partial order planning to build optimization models associated with the different possible activity scenarios and applies branch-and-bound with tight lower bounds to find optimal solutions efficiently. We show how key aspects of fleet repositioning can be modeled using TOP and demonstrate experimentally that our approach scales to the size of problems considered by our industrial collaborators.

Introduction

Liner shipping networks transport containerized cargo through regularly scheduled shipping routes. Fleet repositioning problems consist of moving ships between services in a liner shipping network in order to better orient the overall network to the world economy and to ensure the proper maintenance of ships. Thus, fleet repositioning involves sailing and loading activities subject to complex handling and timing restrictions. As is the case for many industrial problems, the objective is cost minimization (including costs for CO_2 emissions and pollution), and it is important that all cost elements, including the ones that are only loosely coupled with activity choices, can be accurately modeled.

Optimization problems that involve chains of activities with complex interactions, like fleet repositioning problems, are hard to represent as mathematical programs. AI-planning and OR-scheduling offer a wide range of approaches to allocate interacting activities over time, but it has been observed in both fields (e.g., (Karger, Stein, and Wein 1997; Smith, Frank, and Jónsson 2000)) that the compound objectives of real-world problems often are hard to express in terms of the simple objective criteria like makespan and tardiness minimization considered in these approaches.

In this paper, we introduce a novel general framework called Temporal Optimization Planning (TOP) and show that it can model key aspects of fleet repositioning problems. The core idea of TOP is to associate durative planning actions with optimization model components and use planning algorithms to build and search through complete optimization models that are associated with the different activity scenarios of the problem. In contrast to advanced temporal planning languages (Fox and Long 2006), TOP does not enforce a strong semantic relation between planning actions and optimization components. The format of optimization components can be chosen freely by the model designer as long as the set of complete plans corresponds to the set of activity scenarios of the problem.

In this way, TOP allows general optimization models to be constructed, but at the same time makes it possible to naturally represent and explore complex interactions between durative activities with the expressive action models of AI-planning and its powerful search algorithms. In fact, TOP accommodates any optimization model over real-valued variables and thus is a simple way to reason about interacting durative activities in such models.

We solve TOP problems by an optimization version of partial-order planning (Penberthy and Weld 1992) based on a branch-and-bound and demonstrate the approach for linear optimization models. We define a general lower bound for partial plans in the naturally occurring case where the minimum costs of optimization components are non-negative. We show that this bound can be improved by an extension of the h_{max} heuristic (Haslum and Geffner 2000) that makes it possible to estimate the cost of required actions not currently in the plan.

The TOP framework is validated experimentally on a set of fleet repositioning problems developed in collaboration with a liner shipping company. We provide a basic model of a fleet repositioning problem that gives a first step towards solving such problems in TOP. We include key aspects such as the temporal interaction between ships and complex cost objectives such as sailing fuel consumption and the fixed cost of ships over time. Our experimental evaluation shows that the TOP ap-

proach scales to the size of problems faced by the industry and that our lower bounds are tight enough to reduce the total computation time by orders of magnitude.

The remainder of the paper is organized as follows. First we describe fleet repositioning problems, followed by a formalization of TOP along with a description of the h^{cost}_{max} heuristic for estimating lower bounds of partial plans. Then we describe our model of a simple fleet repositioning problem and we present experimental results showing the performance of the TOP framework with several plan selection heuristics. Finally, we draw conclusions and discuss directions for future work.

Fleet Repositioning

Liner shipping networks consist of multiple *services*, which are circular routes that connect a sequence of ports. Each service consists of one or more ships that maintain a weekly schedule. That is, each port on a service is visited by a ship on the same day each week, and a service is assigned as many ships as are necessary to maintain weekly frequency. Ships are periodically *repositioned*, in which they are moved from their current service to some other service. Shipping firms reposition their ships so that they can undergo regular maintenance, be returned to their owner, or to better match ships to markets.

Consider the example repositioning problem shown in Figure 1 in which a new service is being started and requires ships to sail on it. This example involves the temporal alignment of several ships. First a ship from the service on the left must sail to the dashed line service, but the ship on the dashed line service may not leave for its goal service until its replacement ship has arrived, and cargo has been transshipped. Second, the two ships sailing to the service on the right must have a time spacing of exactly one week. Complicating matters further is that shipping firms want to reposition their ships as cheaply as possible.

Fleet repositioning problems can be even more complicated than the example given, with several ships replacing one another in long chains. Aligning the ships temporally and finding a minimum cost plan of activities presents a significant challenge. The possible activities that a ship may undertake during repositioning is given as follows.

- Ships can leave or enter a service only at a port on days when the port is scheduled to be called.
- Ships must sail between their minimum and maximum speed, the cost of which is a function of the ship's speed and draft,
- Ships can idle at a port, incurring a fixed cost per hour (in shipping parlance, *hotel cost*).
- Cargo can be on/off loaded, resulting in a fee per container moved.
- Cargo can be directly transhipped between ships, incurring a reduced per container fee.

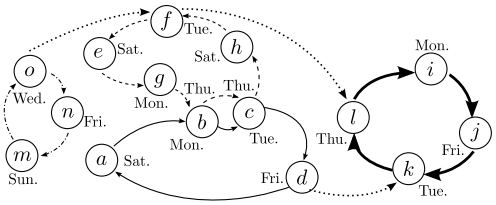


Figure 1: An example fleet repositioning problem involving three ships. Two ships are being repositioned from their initial services (solid, dashed) to a new service (bold solid). The ship on the dashed service is being replaced by a ship from the dashed-dotted service, which will cease operations. Circles represent ports and are labeled with the day the service calls the port.

- Ships can move equipment (empty containers) to where it is needed, reducing the overall cost of repositioning.
- Ships may satisfy certain demands in the shipping network for a profit.

A number of constraints pose restrictions on when the given activities may take place. Ships must be replaced by another ship in order to leave their service, except for certain, designated ships. Ships may not load or unload cargo in certain ports due to cabotage restrictions, which are laws that prevent foreign ships from offering domestic services. If multiple ships are entering the same service, they must enter one week apart in distance or time from one another. In addition, multiple ships must alternate in size such that if there are several ships entering a service, no two ships of the same capacity should follow one another.

Given the high expense of repositioning, the goal of fleet repositioning problems is to find a scenario of activities, which involve continuous decisions regarding sailing time and cost configurations, associated with a lowest cost optimization model.

The fleet repositioning problem is difficult to solve for existing scheduling and planning methods.

Scheduling is concerned with the optimal allocation of scarce resources to activities over time (Karger, Stein, and Wein 1997), and scheduling research has focused on problems that only involve a small, fixed set of choices, while planning problems like fleet repositioning often involve cascading sets of choices that interact in complex ways (Smith, Frank, and Jónsson 2000). Another limitation is that mainstream scheduling research has focused predominately on optimization of selected, simple objective criteria such as minimizing makespan or minimizing tardiness (Smith 2005). More general objective criteria are required in order to solve fleet repositioning problems.

Integer Programming (IP) has successfully been applied to solve classical planning problems using competitive encodings based on the planning graph heuristic (Van Den Briel, Vossen, and Kambhampati 2005). An earlier version of this approach has also been used

to extend classical planning with possibly continuous state variables over which linear constraints and objectives can be stated (Kautz and Walser 1999). A limitation of these encodings is, though, that they are unable represent continuous time. In general, it is difficult to represent partial-order planning with IP since the number of actions in a partial-order plan in principle is unbounded. A SAT encoding of actions with continuous duration (Shin and Davis 2005), however, shows that other encodings of continuous time using IP may be possible, but this encoding currently does not cover any objective criteria except minimizing the number of unique time points in the plan.

Within AI, there exists domain independent planners for temporal planning languages and specialized application planners.With respect to the former, while early approaches had limited scalability (e.g., (Ghallab and Laruelle 1994; Penberthy and Weld 1995)), a number of powerful solvers have recently been developed for planning languages with durative actions, real-valued state variables, and linear change of quantities during action execution (e.g. (Coles et al. 2009; 2010; Li and Williams 2008; Shin and Davis 2005)). These planning languages can model domains where activities depend on shared resources like electric power during execution, which is a typical situation for popular application domains within robotics and aerospace systems (e.g., (Frank, Gross, and Kurklu 2004; Muscettola 1993)). However, the fleet repositioning problem involves decoupled actions that intersect only temporally. Furthermore, most of these domain independent and application specific planning systems only allow simple objective criteria like makespan minimization. A notable exception is Sapa (Do and Kambhampati 2003), which can represent multi-criteria objectives covering any combination of makespan minimization and minimization of fixed action and resource costs. On the other hand, Sapa only handles discrete time and fixed action durations.

Recently, the 2008 International Planning Compeition (Helmert, Do, and Refanidis 2008) featured a net benefit optimization category with several entries: HSP*, MIPS-XXL, and Gamer. These planners are unable to reason temporally, and only support fixed action costs in their objective, preventing them from handling many aspects of fleet repositioning problems, such as hotel costs and variable action cost. Linear Programming has been used to strengthen plan length estimation (e.g. (Bylander 1997; Van Den Briel et al. 2007)), however these approaches do not handle a cost-based objective or temporal setting, with the ordering relaxation in (Van Den Briel et al. 2007) being particularly troubling for a temporal planner.

Temporal Optimization Planning

In the absence of a suitable method for solving fleet repositioning problems, we introduce Temporal Optimization Planning (TOP). TOP diverges fundamentally from classical AI-planning approaches by introducing two sets of variables that decouple the planning problem from the optimization model. Thus, the optimization model is not tightly bound to the semantics of actions. Actions are merely used as handles to optimization components that are built together to complete optimization models using partial-order planning. This decoupling makes it possible to formulate any objective that can be expressed by the applied optimization model. Moreover, computationally expensive action models including real-valued state variables and general objective functions are avoided.

In contrast to the current trend in advanced temporal planning languages, TOP bypasses computationally expensive dense time models of shared resources like electric power consumption during activities. These models are important for the robotic or aerospace applications that often are targeted in AI-planning (e.g., (Frank, Gross, and Kurklu 2004; Muscettola 1993)), but TOP focuses on more physically separated activities where resources are exclusively controlled.

On the other hand, while this decoupling offers some new possibilities, it makes TOP less capable of solving traditional planning problems where a strong coupling is assumed as well as problems that fit within a classical scheduling model.

TOP is built off a state variable representation of propositional STRIPS planning (Fikes and Nilsson 1971). TOP utilizes partial-order planning (Penberthy and Weld 1992), and extends it in several ways. First, an optimization model is associated with each action in the planning domain. This allows for complex objectives and cost interactions that are common in real world optimization problems to be easily modeled. Second, instead of focusing on simply achieving feasibility, TOP minimizes a cost function. Finally, begin and end times can be associated with actions, making them durative. Such actions can have variable durations that are coupled with a cost function.

Formally, let $\mathcal{V} = \{v_1, \cdots, v_n\}$ denote a set of *state variables* with finite domains $D(v_1), \cdots, D(v_n)$. A *state variable assignment* ω is a mapping of state variables to values $\{v_{i(1)} \mapsto d_{i(1)}, \cdots, v_{i(k)} \mapsto d_{i(k)}\}$ where $d_{i(1)} \in D(v_{i(1)}), \cdots, d_{i(k)} \in D(v_{i(k)})$. We also define $vars(\omega)$ as the set of state variables used in ω.

A TOP problem is represented by a tuple

$$P = \langle \mathcal{V}, \mathcal{D}, \mathcal{A}, \mathcal{I}, \mathcal{G}, pre, \mathit{eff}, \boldsymbol{x}, obj, con \rangle,$$

where $\mathcal{D}$ is the Cartesian product of the domains $D(v_1) \times \cdots \times D(v_n)$, $\mathcal{A}$ is the set of actions, $\mathcal{I}$ is a total state variable assignment (i.e. $vars(\mathcal{I}) = \mathcal{V}$) representing the initial state, $\mathcal{G}$ is a partial assignment (i.e. $vars(\mathcal{G}) \subseteq \mathcal{V}$) representing the goal states, pre_a is a partial assignment representing the precondition of action a, eff_a is a partial assignment representing the effect of action a[1], $\boldsymbol{x} \in \mathbb{R}^m$ is a vector of optimization vari-

[1]In practice, is it often more convenient to represent actions in a more expressive form, e.g. by letting the precondition be a general expression on states $pre_a : S \to \mathbb{B}$ and

ables[2] that includes the begin and end time of each action, x_b^a and x_e^a respectively, for all actions $a \in A$, $obj_a : \mathbb{R}^m \to \mathbb{R}$ is a cost term introduced by action a, and $con_a : \mathbb{R}^m \to \mathbb{B}$ is a constraint expression introduced by action a with $con_a \models x_b^a \leq x_e^a \wedge x_b^a \geq 0 \wedge x_e^a \geq 0$.

Let $S = \{\omega | vars(\omega) = \mathcal{V}\}$ denote the set of all the possible states. An action a is applicable in $s \in S$ if $pre_a \subseteq s$ and is assumed to cause an instantaneous transition to a successor state defined by the total assignment

$$succ_{a,s}(v) = \begin{cases} eff_a(v) & \text{if } v \in vars(eff_a), \\ s(v) & \text{otherwise.} \end{cases}$$

We further define $M_a = min\{obj_a | con_a\}$, which is the optimization model component introduced by action a.

A temporal optimization plan is represented by a tuple $\langle A, C, O, M \rangle$, where A is the set of actions in the plan, C is a set of causal links, O is a set of ordering constraints of the form $a \prec b$, where $a, b \in A$, and M is an optimization model associated with the plan. M is defined by

$$\begin{aligned} \min \quad & \sum_{a \in A} obj_a(\boldsymbol{x}) \\ \text{s.t.} \quad & x_e^{a_i} \leq x_b^{a_j} && \forall a_i \prec a_j \in O \quad (1) \\ & con_a(\boldsymbol{x}) && \forall a \in A \quad (2) \end{aligned}$$

The objective of M is to minimize the sum of the costs introduced by actions. The first constraint ensures that the start and end time of actions are consistent with the plan, and second constraint requires that each action constraint is satisfied. Let $cost(\pi)$ denote the cost of an optimal solution to M to a partial plan π.

An *open condition* $\xrightarrow{\mu} b$ is an unfulfilled precondition μ of action $b \in A$, that is, $\mu \in pre_b$ and $\forall a \in A$, $a \xrightarrow{\mu} b \notin C$. An *unsafe link* is a causal link $a \xrightarrow{\mu} b$ that is threatened by an action c such that *i)* $vars(\mu) \in eff_c$, *ii)* $\mu \notin eff_c$, and *iii)* $\{a \prec c \prec b\} \cup O$ is consistent.

To deal with durative actions in TOP we need to keep track of another type of flaw called *interference*. We adopt an interference model based on the exclusive right to state variables (Sandewall and Rönnquist 1986). Thus, two actions a and b interfere if $vars(eff_a) \cap vars(eff_b) \neq \emptyset$ and O implies neither $a \prec b$ nor $b \prec a$.

An open condition flaw $\xrightarrow{\mu} b$ can be repaired by linking μ to an action a such that $\mu \in eff_a$ and by posting an ordering constraint over a and b. Thus, $C \leftarrow C \cup \{a \xrightarrow{\mu} b\}$ and $O \leftarrow O \cup \{a \prec b\}$. In the case that $a \notin A$, $A \leftarrow A \cup \{a\}$ and $O \leftarrow O \cup \{a_0 \prec a, a \prec a_\infty\}$.

An unsafe link $a \xrightarrow{\mu} b$ that is threatened by action c can be repaired by either adding the ordering constraint $c \prec a$ (*demotion*) or $b \prec c$ (*promotion*) to O. Similar to unsafe links, an interference between actions a and b can be fixed by posting either $a \prec b$ or $b \prec a$ to O.

Together, open conditions, unsafe links and interferences constitute *flaws* in a plan. Let $flaws(\pi) = open(\pi) \cup unsafe(\pi) \cup interfere(\pi)$ be the set of flaws in the plan π, where $open(\pi)$ is the set of open conditions, $unsafe(\pi)$ is the set of unsafe links, and $interfere(\pi)$ is the set of interferences. We say that π is a *complete plan* if $|flaws(\pi)| = 0$, otherwise π is a *partial plan*. A plan π^* is optimal if it is feasible and for all feasible solutions π, $cost(\pi^*) \leq cost(\pi)$.

Algorithm 1 Optimization planning algorithm, based on (Williamson and Hanks 1996).

```
1: function TOP(I, G)
2:     Π ← {INITIALTOP(I,G)}
3:     π_best ← null
4:     u ← ∞ ▷ Cost of the incumbent (upper bound)
5:     while Π ≠ ∅ do
6:         π ← SELECTPLAN(Π)
7:         Π ← Π \ {π}
8:         if NUMFLAWS(π) = 0 ∧ COST(π) < u then
9:             u ← COST(π)
10:            π_best ← π
11:        else if ESTIMATECOST(π) < u then
12:            f ← SELECTFLAW(π)
13:            Π ← Π ∪ REPAIRFLAW(π, f)
14:    return π_best
```

Linear Temporal Optimization Planning

To solve fleet repositioning problems, we introduce linear temporal optimization planning (LTOP). In LTOP, all of the optimization models associated with planning actions have a linear cost function and a conjunction of linear constraints. Thus, obj_a is of the form $\mathbf{c^a}\boldsymbol{x'}$, where $\mathbf{c^a} \in \mathbb{R}^m$ and con_a is of the form $\bigwedge_{1 \leq i \leq n_a} (\boldsymbol{\alpha}_i^a \boldsymbol{x'} \leq \beta_i)$, where $\boldsymbol{\alpha}_i^a \in \mathbb{R}^m$, $\beta_i \in \mathbb{R}$ and n_a is the number of constraints associated with action a. Thus, M_a and M are linear programs (LPs).

Algorithm 1 shows a branch-and-bound algorithm that finds an optimal plan to an LTOP problem. First, an initial plan π_{init} is created by the INITIALTOP function (line 2). We define $\pi_{init} = \langle \{a_0, a_\infty\}, \emptyset, \{a_0 \prec a_\infty\}, M_{init} \} \rangle$, where a_0 is an action representing $\mathcal{I}$ with $pre_{a_0} = \emptyset$ and $eff_{a_0} = \mathcal{I}$, a_∞ is an action representing $\mathcal{G}$ with $pre_{a_\infty} = \mathcal{G}$ and $eff_{a_\infty} = \emptyset$, and M_{init} is an optimization model with no objective and two constraints, con_{a_0} and con_{a_∞}, which are special constraints on the dummy actions a_0 and a_∞ such that $con_{a_0} = (x_b^{a_0} = x_e^{a_0} \wedge x_b^{a_0} \geq 0)$ and $con_{a_\infty} = (x_b^{a_\infty} = x_e^{a_\infty} \wedge x_b^{a_\infty} \geq 0)$. The optimization variables $x_b^{a_0}, x_e^{a_0}, x_b^{a_\infty}$ and $x_e^{a_\infty}$ represent the begin and end times of actions a_0 and a_∞

represent conditional effects like resource consumption by letting the effect be a general transition function, depending on the current state of S, $eff_{a,s} : S \to S$. Such expressive implicit action representations may also be a computational advantage. We have chosen a ground explicit representation of actions because it simplifies the presentation and more expressive forms can be translated into it.

[2] In a slight abuse of notation, we sometimes let $\boldsymbol{x}$ denote a set rather than a vector.

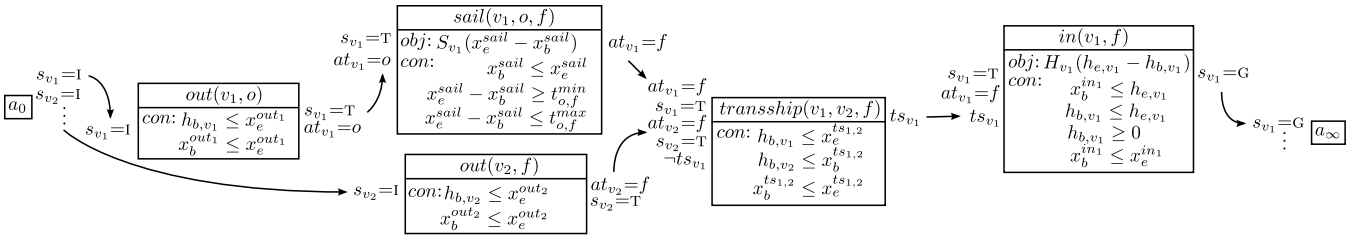


Figure 2: A partial temporal optimization plan from Figure 1 showing the ship sailing from the dash-dotted service to the dashed service (v_1) and the ship sailing from the dashed service to the bold, solid service. Boxes represent actions and contain associated optimization models. Arrows between boxes show causal links. The optimization variables x_b^a and x_e^a represent the begin and end time of action a, and $h_{b,v}$, $h_{e,v}$ are the begin and end hotel time of vessel v respectively. Each vessel v is associated with a state variable s_v with a domain of $\{\text{I}, \text{T}, \text{G}\}$ which indicate that v is on its initial service, in transit or at its goal service, respectively.

respectively.

The algorithm then selects a plan from Π (line 6) and checks to see if it is a complete plan. If the plan is complete, its cost is compared with the current upper bound (u), and if the cost is lower, the *incumbent* π_{best} is replaced with the current plan π and the upper bound is updated (lines 9 and 10). When π is a partial plan, an estimated lower bound of the plan is compared with the cost of the incumbent solution (line 11). If the estimated cost of the plan is higher, the plan is discarded. Otherwise, a flaw is selected and repaired (lines 12 and 13). This process is repeated until Π is empty, at which point the current incumbent is returned, if there is one.

Algorithm 1 is guaranteed to find the optimal solution (if there is one) as long as EstimateCost does not overestimate the cost of completing a partial plan. To prune as much of the branch-and-bound tree as possible, we need tight lower bounds. If we require that the cost of each action subject to its constraints is non-negative, we can prove that $cost(\pi)$ is such a lower bound.

Proposition 1. *Given any valid partial plan* $\pi = \langle A, C, O, M \rangle$ *where* $M_a \geq 0 \ \forall a \in A$, $cost(\pi) \leq cost(\bar{\pi})$ *for any completion* $\bar{\pi}$ *of* π.

Proof. Let π' be π with a single flaw repaired. The flaw is either *i*) an unsafe link, *ii*) an interference, *iii*) an open condition being satisfied by an action in the plan, or *iv*) an open condition being satisfied by an action not in the plan.

In cases *i* and *ii* the flaw is repaired by adding an ordering constraint to π, which further constrains π, thus $cost(\pi) \leq cost(\pi')$. Case *iii* results in a new causal link and an ordering constraint, and is therefore the same as cases *i* and *ii*. In case *iv*, the action's optimization model is added to π, but since the cost function of the action must be non-negative under its constraints, $cost(\pi')$ cannot be less than $cost(\pi)$. By applying this argument inductively on the complete branch-and-bound subtree grown from π, we get $cost(\pi) \leq cost(\bar{\pi})$ for any completion $\bar{\pi}$ of π. □

Heuristic Cost Estimation

Although $cost(\pi)$ provides a reasonable lower bound for π, the bound is only computed over actions in the plan. It can be strengthened by also reasoning over actions that are not in the plan. We present an extension of the h_{max} heuristic (Haslum and Geffner 2000), h_{max}^{cost}, which estimates the cost of achieving the open conditions of a plan π. Let

$$h_{max}^{cost}(\omega, \pi) = \begin{cases} 0 & \text{if } \omega \subseteq \mathit{effs}_\pi, \text{ else} \\ f(\omega, \pi) & \text{if } \omega = \{\mu\}, \text{ else} \\ g(\omega, \pi) & \text{if } |\omega| > 1, \end{cases}$$

$$f(\omega, \pi) = \min_{\{a \in \mathcal{A} \setminus A | \mu \in \mathit{eff}_a\}} \{M_a + h_{max}^{cost}(\mathit{pre}_a, \pi)\},$$
$$g(\omega, \pi) = \max_{\mu \in \omega} \{h_{max}^{cost}(\{\mu\}, \pi)\},$$

where ω is a partial state variable assignment, μ is a single state variable assignment $v \mapsto d$, and $\mathit{effs}_\pi = \bigcup_{a \in A} \mathit{eff}_a$. The heuristic takes the max over the estimated cost of achieving the elements in the given assignment ω. The cost is zero if the elements are already in π, otherwise the minimum cost of achieving each element is computed by finding the cheapest way of bringing that element into the plan.

Proposition 2. *Given any valid partial plan* $\pi = \langle A, C, O, M \rangle$ *where* $M_a \geq 0 \ \forall a \in A$, $cost(\pi) + h_{max}^{cost}(open(\pi), \pi) \leq cost(\bar{\pi})$ *for any completion* $\bar{\pi}$ *of* π.

Proof. We have $h_{max}^{cost}(\omega, \pi) = \sum_{a \in R} M_a$, where R is a set of actions not currently in π ($R \cap A = \emptyset$) that are required to resolve ω and among such sets has the minimum cost. Thus, any completion $\bar{\pi}$ of π as described in Proposition 1 must at least increase $cost(\pi)$ by $h_{max}^{cost}(\omega, \pi) = \sum_{a \in R} M_a$. □

Modeling Fleet Repositioning

We describe an LTOP model of fleet repositioning problems that represents a first step towards modelling and solving fleet repositioning problems. Our model represents a subset of fleet repositioning that captures its difficult elements, while excluding excessive detail. We focus on several key components of the fleet repositioning

problems that form the basis for more difficult versions including *i*) the temporal interaction between ships in a sequence of replacements, *ii*) variable ship sailing durations, *iii*) sailing costs that vary linearly with the sailing duration, and *iv*) ship hotel costs that must span multiple actions.

Given a set of services, the goal is to carry out a sequence of repositionings in which a ship on each service is moved to the next service in the sequence at minimal cost. A ship may not be moved from its service until replaced by another ship, except in the case of the first service in the sequence.

Ships cease operations on their initial service through the action *out* and begin operations on a service through the *in* action. During the time in between *out* and *in*, ships are charged their hotel cost, which is represented in the objective of the *in* action. The hotel cost objective is given in the *out* action for ships that do not have a target service, and will therefore not use the *in* action, such as a ship being returned to its owner or heading for repairs.

Ships must *sail* between ports within their minimum and maximum speed, incurring a cost that varies linearly with the speed. In order for a ship to replace another ship on a service, they must transship their cargo. Transshipments in our model are instantaneous, free actions that require both ships to simultaneously be at a port where it is lawful for the ships to perform a *transshipment*. Transshipments are only allowed at ports in which a ship is lawfully able to transfer cargo, which we are able to represent by simply not including transshipment actions between ships at ports where it would be unlawful.

Implicit in our model are wait actions. Such an action does not need to be explicitly given, because actions, even those that are ordered or linked, are not required to start directly after one another. The objective of this implicit action is given by the hotel cost computed in the *out* and *in* actions.

Figure 2 shows a fleet repositioning partial plan in the LTOP framework in which two ships are moved off of their initial services (*out* action), and meet at port f where they transship cargo (*transship* action). Ship v_1 then begins service at port f (*in* action), freeing v_2 to continue to a different service or undergo maintenance. Notice how the objective for the hotel cost for v_1, represented by h_{b,v_1} and h_{e,v_1}, is computed in the $in(v_1, f)$ action, but the bounds for the hotel cost are updated throughout the partial plan. This allows the LP to compute a more accurate bound throughout the planning process. Multiple actions are therefore contributing to the hotel cost, meaning that the interaction of actions can have interesting cost consequences.

Even this simple version of the fleet repositioning problem is not solvable with existing scheduling or planning approaches like ZENO (Penberthy and Weld 1995) due to the lack of general optimization criteria or Sapa (Do and Kambhampati 2003) due to the lack of continuous time. But even if it was, an advantage of TOP from an OR-perspective is that it is easy to use any optimization model within the framework without changing the representation of activities.

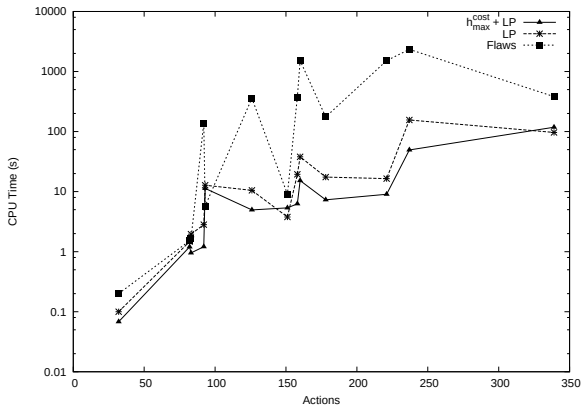


Figure 3: A plot of the performance of LTOP with the planning heuristics $h^{cost}_{max} + LP$ (*solid line*), LP (*dashed line*) and the number of flaws, $Flaws$ (*dotted line*).

Experimental Evaluation

We created a dataset of sample problems based on discussions with a liner shipping company. The instances range in size from 4 to 12 ports with 2 or 3 services over time frames of 2 to 3 weeks.

Table 1 and Figure 3 display the performance of our LTOP solver using several plan selection heuristics on our dataset. Results were computed on dual six-core 2 GHz AMD Opteron 2425 HE processors, and each execution was allowed a maximum of 4 GB of RAM. In addition, our LTOP solver uses the linear programming solver in COIN-OR 1.5.0 (Lougee-Heimer 2003). The number of actions in the optimal plan varies with the number of ships being repositioned. Instance CR1's optimal plan has five actions, while CR13 has 8 actions.

The $h^{cost}_{max} + LP$ plan selection heuristic selects the cheapest plan available using the sum of the real cost of a plan and the estimated cost of the plan's completion, the LP heuristic only uses the real cost of the plan, and the $Flaws$ heuristic selects the plan with the lowest number of flaws. The $LP + h^{cost}_{max}$ heuristic performs the best, taking on average 61% of the time of the LP heuristic, and only 3.3% of the time of the $Flaws$ heuristic. The geometric mean of $h^{cost}_{max} + LP$ is over 12 times as fast as $Flaws$ and almost twice as fast as LP, indicating that $h^{cost}_{max} + LP$ performs well across the entire dataset, and not just on a few instances. Thus, the superior search guidance provided by the h^{cost}_{max} heuristic is worth the extra computation time. The TOP framework is therefore able to scale to solve real world sized problems with the LP and h^{cost}_{max} heuristics.

Instance	Actions	$h^{cost}_{max} + LP$	LP	$Flaws$
CR1	32	**0.07**	0.10	0.20
CR2	82	**1.20**	1.47	1.52
CR3	83	**0.95**	1.96	1.64
CR4	92	**1.21**	2.82	138.53
CR5	93	11.32	12.80	**5.74**
CR6	126	**4.96**	10.50	358.81
CR7	151	5.36	**3.78**	9.06
CR8	158	**6.27**	19.40	376.84
CR9	160	**15.25**	38.00	1,519.92
CR10	178	**7.30**	17.44	174.89
CR11	221	**9.02**	16.46	1,519.92
CR12	237	**49.45**	156.02	2,339.51
CR13	339	118.20	**96.67**	382.12
Mean		**17.74**	29.03	525.28
Geo. Mean		**4.94**	8.55	61.35
Std. Dev.		32.81	46.03	762.30

Table 1: Results from our LTOP solver on a crafted dataset for our sample fleet repositioning problem with several plan selection heuristics. All times are CPU times given in seconds.

Conclusion

We presented a novel framework called Temporal Optimization Planning (TOP) for modeling and solving fleet repositioning problems with compound objectives spread throughout interconnected activities. We introduced an extension to the domain independent h_{max} heuristic, h^{cost}_{max}, and showed that by using this heuristic to estimate the costs of actions required to complete a plan, the TOP framework is capable of scaling to the size of real fleet repositioning problems.

We gave a model of a fleet repositioning problem that represents the first step to understanding and solving problems in the fleet repositioning domain. The model captured key aspects of fleet repositioning that cannot be modeled with current methods, such as hotel cost, variable sailing durations and duration linked costs.

The TOP framework shows promise as a new method for tackling many industrial cost minimization problems that are difficult to solve using state-of-the-art AI or OR methods. We intend to further investigate TOP on more realistic versions of the fleet repositioning problem and related problems, including problems that are non-linear and problems that do not have non-negativity restrictions on the optimization models. In addition, we will investigate forward-chaining methods for TOP, similar to those in (Coles et al. 2010). Finally, we also intend to see if it is possible to implement TOP within a mixed integer programming framework.

Acknowledgements

We would like to thank our industrial collaborators Mikkel Muhldorff Sigurd and Shaun Long at Maersk Line for their support and detailed description of the fleet repositioning problem. This research is sponsered in part by the Danish Council for Strategic Research as part of the ENERPLAN research project.

References

Bylander, T. 1997. A linear programming heuristic for optimal planning. In *Proceedings of the National Conference on Artificial Intelligence*, 694–699.

Coles, A.; Coles, A.; Fox, M.; and Long, D. 2009. Temporal planning in domains with linear processes. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

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

Do, M., and Kambhampati, S. 2003. Sapa: A multi-objective metric temporal planner. *Journal of Artificial Intelligence Research* 20(1):155–194.

Fikes, R., and Nilsson, N. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial intelligence* 2(3-4):189–208.

Fox, M., and Long, D. 2006. Modelling mixed discrete-continuous domains for planning. *Journal of Artificial Intelligence Research* 27(1):235–297.

Frank, J.; Gross, M.; and Kurklu, E. 2004. SOFIA's choice: an AI approach to scheduling airborne astronomy observations. In *Proceedings of the National Conference on Artificial Intelligence*, 828–835. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999.

Ghallab, M., and Laruelle, H. 1994. Representation and control in IxTeT, a temporal planner. In *Proceedings of AIPS*, volume 94, 61–67.

Haslum, P., and Geffner, H. 2000. Admissible heuristics for optimal planning. In *Proc. AIPS*, 140–149. Citeseer.

Helmert, M.; Do, M.; and Refanidis, I. 2008. The sixth international planning competition, deterministic track. http://ipc.informatik.uni-freiburg.de

Karger, D.; Stein, C.; and Wein, J. 1997. Scheduling algorithms. *CRC Handbook of Computer Science*.

Kautz, H., and Walser, J. 1999. State-space planning by integer optimization. In *Proceedings of the National Conference on Artificial Intelligence*, 526–533.

Li, H., and Williams, B. 2008. Generative planning for hybrid systems based on flow tubes. In *Proc. 18th Intl. Conf. on Automated Planning and Scheduling (ICAPS)*.

Lougee-Heimer, R. 2003. The Common Optimization INterface for Operations Research: Promoting open-source software in the operations research community. *IBM J. Res. Dev.* 47:57–66.

Muscettola, N. 1993. HSTS: Integrating planning and scheduling. In Zweben, M., and Fox, M., eds., *Intelligent Scheduling*. Morgan Kaufmann. 169–212.

Penberthy, J., and Weld, D. 1992. UCPOP: A sound, complete, partial order planner for ADL. In *Proceed-*

ings of the Third International Conference on Knowledge Representation and Reasoning, 103–114.

Penberthy, J., and Weld, D. 1995. Temporal planning with continuous change. In *Proceedings of the national conference on Artificial Intelligence*, 1010–1015.

Sandewall, E., and Rönnquist, R. 1986. A Representation of Action Structures. In *Proceedings of 5th (US) National Conference on Artificial Intelligence*, 89–97. American Association for Artificial Intelligence, Morgan Kaufmann.

Shin, J., and Davis, E. 2005. Processes and continuous change in a SAT-based planner. *Artificial Intelligence* 166(1-2):194–253.

Smith, D.; Frank, J.; and Jónsson, A. 2000. Bridging the gap between planning and scheduling. *The Knowledge Engineering Review* 15(1):47–83.

Smith, S. 2005. Is scheduling a solved problem? *Multidisciplinary Scheduling: Theory and Applications* 3–17.

Van Den Briel, M.; Benton, J.; Kambhampati, S.; and Vossen, T. 2007. An LP-based heuristic for optimal planning. In *Principles and Practice of Constraint Programming–CP 2007*, 651–665. Springer.

Van Den Briel, M.; Vossen, T.; and Kambhampati, S. 2005. Reviving integer programming approaches for AI planning: A branch-and-cut framework. In *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS 2005)*, 310–319.

Williamson, M., and Hanks, S. 1996. Flaw selection strategies for value-directed planning. In *Proceedings of the Third International Conference on Artificial Intelligence Planning Systems*, volume 23, 244.

Knowledge representations for high-level and low-level planning

Franziska Zacharias and Christoph Borst
Institute of Robotics and Mechatronics,
German Aerospace Center (DLR), Wessling, Germany

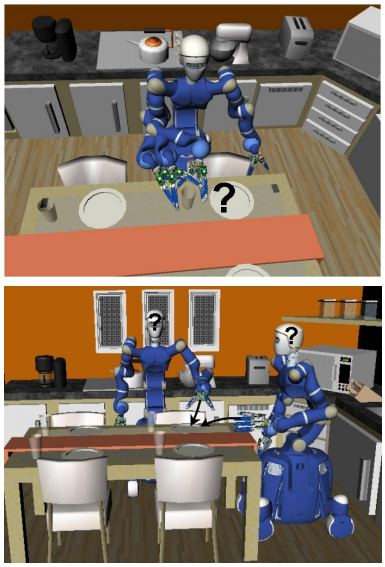


Figure 1: A humanoid robot is shown in a kitchen performing several tasks. (top) With a multi-fingered hand, a glass can be grasped in different ways. (bottom) Two robot placements are shown for grasping the plate on the table.

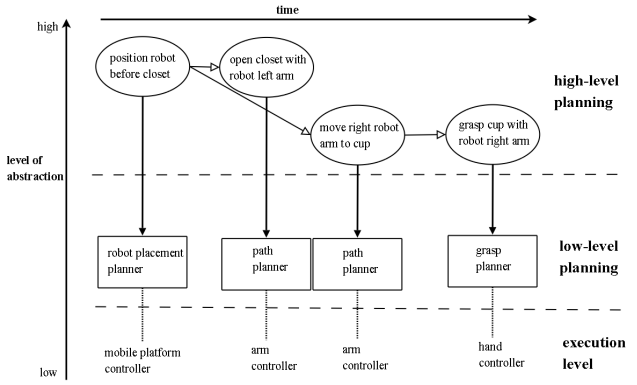


Figure 2: Different levels of abstraction during the planning and execution of a service task. High-level planning is equivalent with logical planning. Possibly parallel running actions are mapped to low-level planners. The planning results are then executed by divers robot controllers.

Problem statement

In general a service task like setting the table can be further resolved into several subtasks. When objects have to be grasped, choosing grasps and approach directions are problems that have to be solved. In Figure 1 two service tasks and their challenges are illustrated. Objects have to be moved from one location to another. A number possibilities for grasping a glass are shown in Figure 1 (top). The robot has to decide where to place itself to be able to reach an object. The number of robot placements is infinite (Figure 1 (bottom)). These issues have to be taken into account by a logical planner when planning a task.
Logical planners are expected to divide a task into a set of subtasks, e.g. first the closet is opened, then the dishes are taken out of the closet, the closet is closed and the dishes are transported to the table. Path planners are used for moving the robotic arm without collisions between two positions. A grasp planner provides good grasps for handling an object. A robot placement planner positions a robot for performing a grasp or a trajectory. The logical planner has to trigger the execution of the subtasks by appropriately parameterizing and using the low-level planners, i.e. the path planner, the grasp planner and the robot placement planner (Figure 2). Depending on the parameterization, e.g. the start and the goal robot arm configuration, a low-level planning problem may not be solvable. Either no collision-free path is found or the object cannot be grasped from the queried direction. However, a logical planner has no knowledge about the geometry of the scene and works with an abstract scene model where objects are represented by labels. It does not know e.g. from which direction an object can be approached best. Therefore a chosen subtask may not be executable. For the high-dimensional planning problems in service robotics, determining whether a solution exists is computationally too expensive. A brute force approach lets the logical planner propose a plan and test whether the plan is valid. In this process the low-level planners try to find a solution for assigned subtasks to determine the truth value of associated labels. This is repeated until a solution is found or a termination criterion is met (Dornhege et al. 2009), (Kaelbling

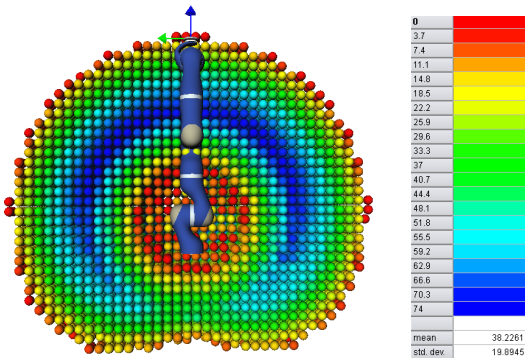


Figure 3: Visualization of the capability map. Regions in the center of the workspace (blue) can be reached with the largest number of poses.

and Lozano-Perez 2010). Since each of the low-level planning problems is already very complex, even simple tasks can take a long time to be solved. Furthermore, for objects like a coffee mug, directions exist from which the object is better graspable. To open a closet door or a dish washer, not every placement of the humanoid robot results in successful task execution. The humanoid robot may place itself so that it can grasp the door handle but opening the door is not possible. Therefore, models are needed that e.g. describe the capabilities of a robot. They can support decision processes, their parameter choices and reduce the search space.

The capability map

In the reachable workspace volume of the robot arm, positions can be reached in *at least one* orientation. In the dexterous workspace volume positions can be reached in *all* orientations (Craig 1989). However, in general seldom all orientations are needed. Let the *versatile workspace* of a robot arm describe with which orientations a position can be reached. A representation of the versatile workspace for a robot arm can be exploited by high-level and low-level planner types. It enables a task planner to predict whether an object is graspable or whether a certain trajectory is executable for the robot. This information helps to estimate whether an action is valid. Given a set of grasps for an object and a scene description, the representation can be used to estimate the difficulty of the planning problem. For instance, if a lot of grasps are unreachable, the scene could be very crowded and the target object could be difficult to reach resulting in long planning times. This information can be used by the task planner to e.g. consider a rearrangement of the scene to make the planning problem easier.

The *capability map* is a representation of the versatile workspace of a robot arm (Zacharias, Borst, and Hirzinger 2007). Using this knowledge representation good parameterizations for planners can be determined or the search space can be reduced. The capability map of a robotic arm describes how well regions of the workspace are reachable (Figure 3). A visualization of the versatile workspace facilitates analysis and interactive planning. The representation can be used to guide planning processes, make reliable predictions about the feasibility of tasks and avoid unsuccessful planning runs. The generation of the model is performed offline once and can then be used in online algorithms.

Applications

(Zacharias et al. 2009) presented an algorithm that uses the capability map to determine where given 3D trajectories are executable. The search method can be used to evaluate how well a robot is suited for specific environments or tasks. The determined number of solutions for the trajectory correlates with the ability of the robot to cope with disturbances e.g. objects left behind by a human. The method is especially suited to decide whether or not a task can be performed. This information can be used by a task planner to decide which planner or execution component to trigger. In (Zacharias et al. 2011) an ergonomics criterion in combination with the capability map was used to determine whether objects are graspable in a human-like manner. Good parameterizations for a path planner were derived. The path planner was then able to plan more human-like robot arm motion. The computation times and quality of the robot motion were significantly improved. This method can help a logical planner evaluate the feasibility of grasping tasks and obtain good planner parameterizations. (Pandey and Alami 2010) introduce the *mightability map* to reduce the search space in planning human-robot interaction tasks.

To be able to use logical planning to efficiently solve service robotics tasks, more knowledge representations like the capability map are needed to reduce the search space dimensionality and provide an intermediate layer between logical planning, geometrical planning and robotics.

References

Craig, J. 1989. *Introduction to Robotics: Mechanics and Control*. Addison-Wesley.

Dornhege, C.; Eyerich, P.; Keller, T.; Trüg, S.; Brenner, M.; and Nebel, B. 2009. Semantic attachments for domain-independent planning systems. In *Intl. Conf. on Automated Planning and Scheduling (ICAPS)*.

Kaelbling, L. P., and Lozano-Perez, T. 2010. Hierarchical planning in the now. In *IEEE Intl. Conf. on Robotics and Automation, Workshop on Mobile Manipulation*.

Pandey, A. K., and Alami, R. 2010. Mightability maps: A perceptual level decisional framework for co-operative and competitive human-robot interaction. In *Proc. IEEE Intl. Conf. on Intelligent Robots and Systems (IROS)*.

Zacharias, F.; Sepp, W.; Borst, C.; and Hirzinger, G. 2009. Using a model of the reachable workspace to position mobile manipulators for 3-d trajectories. In *Proc. IEEE Intl. Conf. on Humanoid Robots*.

Zacharias, F.; Schlette, C.; Schmidt, F.; Borst, C.; Rossmann, J.; and Hirzinger, G. 2011. Making planned paths look more human-like in humanoid robot manipulation planning. In *IEEE Intl. Conf. on on Robotics and Automation (ICRA)*.

Zacharias, F.; Borst, C.; and Hirzinger, G. 2007. Capturing robot workspace structure: Representing robot capabilities. In *Proc. IEEE Intl. Conf. on Intelligent Robots and Systems (IROS)*.