Sixth International Conference on
Artificial Intelligence Planning and Scheduling

Workshop on

On-line Planning and Scheduling

Toulouse, France
April 24, 2002

Background

Whereas the first automatic planning and scheduling systems were designed to run in an *off-line* setting (planning/scheduling and execution performed in sequence), current systems are frequently designed to run in an *on-line* setting (planning/scheduling and execution running concurrently). Application domains include manufacturing processes, supply chains, workforce organizations, power or communication networks, air or ground transportation, robot control, autonomous spacecraft, and unmanned air vehicles.

New requirements and new technical challenges arise in such a setting. Among them:

- the planning/scheduling problem is no longer a static problem; it becomes a *dynamic* problem, with typically small changes in the definition of the instance to solve between successive calls to the planning/scheduling function;
- new requirements appear, such as *robustness* and *stability* in the face of modeling uncertainties and unexpected events that may occur during execution; these requirements must be combined with the usual requirement of producing a plan/schedule of the best possible intrinsic quality;
- hard and soft constraints on the *time* taken by the planning/scheduling function are introduced, requiring the planning/scheduling module to monitor and control its reasoning process;
- it becomes necessary to take into account inside the planning/scheduling reasoning the *time* at which the plan/schedule will be provided and executed.

Topics of interest

The workshop will focus on these requirements and the various ways of satisfying them. Topics of interest include:

- *analysis* of the on-line setting and of the new arising requirements;
- planning/scheduling as a *dynamic* problem: methods and algorithms;
- *reactive* planning/scheduling; *repair-based* planning/scheduling;
- plan/schedule *quality*, *robustness* and *stability*: how to define, to combine and to optimize them?
- planning/scheduling at various *abstraction* levels with various temporal *horizons*;
- *constraints* on the *time* taken by the planning/scheduling function : what are they? how to set and to meet them?
- *control* of the planning/scheduling reasoning; what trade-offs exist between *quality* and *time*?
- reasoning *about time* and *within time*;
- what trade-offs exist between *off-line* and *on-line* computing?
- task sharing and interactions between the *planning*, *scheduling*, and *execution* functions;
- *validation* and *assessment* of an on-line planning/scheduling system;
- *real-world* on-line planning/scheduling *applications*; selected approaches, implementation issues, experimental results, comparisons between approaches or implementations.

Organization

This workshop is both an AIPS 2002 workshop and a workshop of the TCU (Technical Coordination Unit) *On-line Planning and Scheduling* of the PLANET network (*European Network of Excellence in Artificial Intelligence Planning*). It has been significantly sponsored by PLANET (see http://www.planet-noe.org/).

Workshop chair

- Gérard Verfaillie, ONERA, France
  verfaillie@cert.fr

Program committee

- Chris Beck, Ilog, France
  cbeck@ilog.fr
- Markus Fromherz, Xerox PARC, USA
  fromherz@parc.xerox.com
- Simon de Givry, Thales Research & Technology, France
  simon.degivry@thalesgroup.com
- Félix Ingrand, LAAS/CNRS, France
  felix@laas.fr
- Ari Jonsson, NASA Ames, USA
  ajonsson@arc.nasa.gov
- Russell Knight, JPL, USA
  knight@aig.jpl.nasa.gov
- François Laburthe, Bouygues/DTN, France
  flaburthe@bouygues.com
- Angelo Oddi, IP-CNR, Italy
  oddi@www.ip.rm.cnr.it
- Jon Spragg, APSOLVE, UK
  john.spragg@bt.com
- Shlomo Zilberstein, University of Massachusetts, USA
  shlomo@cs.umass.edu

Contents

Invited presentation

Long papers

Short position papers

Technologies for Dynamic Scheduling

Stephen F. Smith
The Robotics Institute
Carnegie Mellon University
sfs@cs.cmu.edu

The practical goal of scheduling is to orchestrate an optimized behavior of some resource-limited system or organization over time. This goal is complicated in dynamic domains, where evolving execution circumstances tend to force changes to planned activities, and hence plans and schedules have a limited lifetime. In such domains, the problem becomes more than a classical optimization problem, and the design of effective solutions must address several additional complications. For example:

- How to manage solution change - As execution events (e.g., unexpected outcomes, new requirements) signal the need for reassessment and revision of the current solution, there is usually also contradictory pressure to keep things the same. Once wheels have been set in motion there is generally a cost in changing course, and this fact must be accounted for during any re-optimization effort.
- When to optimize and when to hedge - though optimized performance is the goal of advance planning and scheduling, highly optimized plans/schedules tend to be very brittle. On the other hand, the construction of plans/schedules that hedge against uncertainty and ensure executability typically guarantee very little in the way of optimized system behavior. How does one get an optimizing behavior while avoiding jitter and keeping pace with execution?
- When to plan and when to react - Similarly, depending on the level of unpredictability in the operating environment, it may be more or less productive to plan and schedule in advance. However, event-based execution policies are susceptible to sub-optimal (and sometimes chaotic) decision-making.

In this talk, I will describe ongoing research aimed at answering these questions in different application contexts and, in the process, developing core techniques and tools for dynamic scheduling. For the past several years, we have been developing search procedures and heuristics for controlled, incremental solution change. These techniques have been embedded in a now-operational tool for day-to-day management of airlift and tanker missions at the USAF Air Mobility Command; and another current project is integrating the same core technology with a commercial Manufacturing Execution System for use as a dynamic shop-floor control system. I will summarize the principles underlying this approach and indicate current research toward complementing this incremental change framework with capabilities for retaining execution flexibility. I will also discuss other recent ideas and work toward the development of so-called self-scheduling systems, which rely on local but adaptive scheduling policies to achieve globally optimizing behavior. Along the way, I will give my views of the principal challenges that remain.

Anticipatory Scheduling for Dynamic Job Shop Problems

Jürgen Branke
Univ. of Karlsruhe, AIFB, 76131 Karlsruhe
<jbr@aifb.uni-karlsruhe.de>

Dirk C. Mattfeld
Univ. of Bremen, FB 7, Box 330440, 28334 Bremen
<dirk@aifb.uni-bremen.de>

Abstract

Many real-world optimization problems change over time and require frequent re-optimization. We show that in these cases the overall system performance can be improved by anticipating the necessity to adapt a solution in the future. Therefore the optimization algorithm should search for solutions that are not only good, but also flexible, i.e. easily adjustable if necessary in the case of problem changes.
For the example of a job shop with jobs arriving non-deterministically over time, we demonstrate that the incorporation of a flexibility term as secondary goal into the evolutionary algorithm used for scheduling can greatly enhance performance.

Introduction

Many real-world optimization problems are dynamic and stochastically change over time. One standard approach to deal with such a dynamic problem is to constitute and solve a new deterministic sub-problem every time the problem data changes. However, considering the sub-problems as being completely independent disregards the impact a solution may have on the system's state, and thus on the problems encountered in the future. In this paper, we make the following conjecture:

If a problem requires sequential decision making under an uncertain future, and if the decisions impact the future state of the system, an optimization algorithm should anticipate future needs. This can be done by not just focusing on the optimization criterion at hand, but by additionally trying to move the system into a flexible state, i.e. a state that facilitates quick adaptation if necessary.

We are going to examine this conjecture, considering a minimum summed tardiness job shop scheduling problem with new jobs arriving non-deterministically over time. Clearly, the scheduling decisions influence the shop fbor's future state, e.g. its work in process, the interdependencies between jobs, or the distribution of machine idle times.

As we will demonstrate, the state's flexibility in a job shop is largely determined by machine capacity, and can be effectively preserved by avoiding early idle times. Therefore the evolutionary algorithm used in this research integrates a secondary goal, penalizing early idle times, into the objective function. We then show empirically that such an anticipatory scheduling can greatly improve the system's performance when applied in a dynamic shop fbor environment.

For an early investigation of scheduling with a rolling time horizon on a minimum summed tardiness job shop problem see (Raman & Talbot 1993). The shorter the rescheduling-interval is chosen, the more responsive the approach becomes as information is taken into account earlier. Previous work (Adam & Surkis 1980; Farn & Muhlemann 1979; Muhlemann, Lockett, & Farn 1982) suggests that an increasing rescheduling frequency can improve the scheduling performance. (Yamamoto & Nof 1985) propose event triggered rescheduling, i.e. deterministic sub-problems are created as soon as new jobs arrive in the manufacturing system. (Church & Uzsoy 1992) have demonstrated the advantages of event triggered rescheduling over interval based rescheduling. (Fang & Xi 1997) propose a combined technique which performs event based rescheduling in case of machine breakdowns and interval based rescheduling to cope with the arrival of new jobs.

If either the problem changes so quickly that the required frequency for generating global schedules becomes prohibitive, or if the problem changes so drastically that no meaningful information can be obtained from pre-schedules, priority based dispatching offers a serious alternative. Because only up-to-date local information is taken into account priority based dispatching is an obvious candidate for dynamic scheduling problems (Panwalkar & Iskander 1977; Morton & Pentico 1993). On the other hand, this approach is limited by the lack of a global goal function used for optimization. As a remedy (Kutanoglu & Wu 1998) suggest to combine global optimization with local dispatching by determining job priorities on a global level beforehand, while deferring the priority driven dispatching as long as possible.

None of the approaches described so far anticipates future changes. Anticipating future events is required when searching for robust schedules, i.e. when a schedule's expected quality in an uncertain environment is optimized. Examples include the work by (Leon, Wu, & Storer 1994) who derive an explicit measure for robustness, or (Jensen 2001), who uses a stochastic evaluation as part of an evolutionary algorithm. (Mehta & Uzsoy 1998) produce robust schedules by inserting idle-times in anticipation of ma-

chine breakdowns. But while robustness is required for solutions that are assumed to persist unchanged, in this paper we seek flexibility, i.e. we assume the necessity of repeated adaptations of a solution (Branke & Mattfeld 2000; Branke 2001). These two aspects are not necessarily unrelated, as (Jensen 2001) has observed that robust solutions are often also flexible.

Only recently, the authors have proposed to penalize machine idle-times in the objective function of the scheduling algorithm in order to maintain flexibility with respect to the arrival of future jobs (Branke & Mattfeld 2000). This approach tends to exhaust machine capacity and withholds machine idle-time in anticipation of future demand. An attempt to make the penalty term subject to evolutionary optimization has been made by (Snoek 2001), with limited success. In this paper, we discuss our approach much more thoroughly, with more empirical evidence and from a fresh and clearer perspective. We show that anticipatory scheduling provides excellent scheduling performance in an inexpensive and immediate way.

Dynamic Job Shop Scheduling

We consider a dynamic job shop problem where n jobs are to be processed on m machines. The processing of a job on a certain machine is referred to as operation. Processing times are deterministic such that the processing times of operations belonging to job i add up to the job's total processing time p_i. Every job is processed in a prescribed technological order which does not necessarily cover all machines. A job can be processed by one machine at a time only and one machine can process just one job simultaneously, preemption is not allowed.

Due-dates d_i indicate the point in time at which a job should be completed, the actual time c_i of completing job i, however, is subject to optimization. The spread between the actual and desired completion time is taken into account by minimizing the mean tardiness $\overline{T} = \frac{1}{n}\sum_{i=1}^{n} \max\{c_i - d_i, 0\}$.

Jobs become known non-deterministically over time which implies that job i cannot start before its release at arrival time r_i. The inter-arrival times of jobs in the manufacturing system affect its workload, i.e. the number of operations in the system which await processing. The mean inter-arrival time λ can be determined by dividing the mean processing time of jobs $\overline{P}$ by the number of machines m and a desired utilization rate U, i.e. $\lambda = \overline{P}/(mU)$. A utilization rate of $U = 0.7$ represents a relaxed situation of the manufacturing system. A moderate load is produced by $U = 0.8$ whereas a utilization rate of $U = 0.9$ corresponds to a heavy workload. We simulate a simplified manufacturing system as follows:

- The manufacturing system consists of 6 machines.
- Each job passes 4 to 6 machines resulting in 5 operations on average.
- Technological orders are generated from a uniform probability distribution.
- Processing times of operations are uniformly distributed in the range of $[1, 19]$ resulting in a mean processing time of $\overline{P} = 5 \cdot 10$.
- Relatively tight due-dates are generated by $d_i = r_i + 2 \cdot p_i$.
- Inter-arrival times are exponentially distributed with mean λ.

A problem instance consists of 500 jobs. In order to circumvent distortion effects we discard jobs 1 to 100 as well as jobs 401 to 500 from being evaluated (Bierwirth & Mattfeld 1999). Consequently, the empirical results presented in this paper are calculated as $\overline{T}$ observed for job 101 to job 400 averaged over 30 different problem instances.

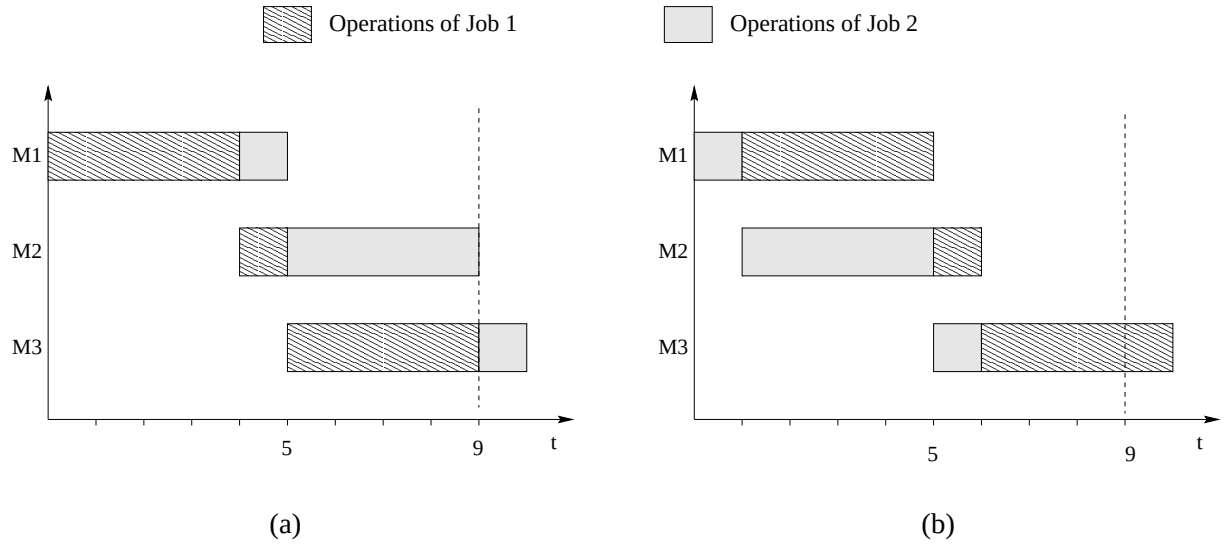
Evolutionary Algorithms for Dynamic Scheduling

Most EA approaches to scheduling problems (and also the one we are using in this paper) encode solutions by priorities for operations (Cheng, Gen, & Tsujimura 1999). In order to determine the fitness of an encoded solution, a schedule builder constructs a schedule by consecutively inserting operations along the time axis. Whenever two or more operations compete for a machine, the conflicts are resolved by the encoded priority scheme.

A schedule builder operates by iteratively considering machine M' with the earliest possible starting time t' of an operation. For *non-delay* schedules one of the operations queued in front of M' is picked for dispatching which can start at $t = t'$, i.e. a machine is never kept idle when there is an operation that might be started. To produce *active* schedules, an operation is determined on M' with a minimal possible completion time c''. Here, an operation is dispatched on M', which can start in the interval $t' \leq t < c''$. Furthermore we can think of *hybrid* schedules, simply by considering the interval $t' \leq t < t' + (c'' - t')\delta$ with $\delta \in [0, 1]$ defining a bound on the time span a machine is allowed to remain idle (Storer, Wu, & Vaccari 1992).

Active schedules allow for additional machine idle-time and therefore the set of active schedules is larger compared to the set of non-delay schedules. It is possible to scale the size of the set of hybrid schedules by means of δ. On one hand, the additional possibility of inserting idle times in active schedules may be beneficial, e.g. when a machine is deliberately kept idle for a short time to wait for an urgent job. On the other hand, for regular measures of performance, additional idle time often reduces the quality of the schedule, and there is evidence that non-delay schedules perform much better on average (Norman & Bean 1997; Bierwirth & Mattfeld 1999). Concerning the optimal solution, it is known that there is at least one optimal active schedule, while there is not necessarily an optimal non-delay schedule.

As default we use a schedule builder that generates active schedules ($\delta = 1.0$) while the effect of hybrid schedule builders is examined in Section . Besides the above described components, standard EA settings are used in this research. Selection is based on inverse proportional fitness, the recombination probability is 0.6 and the mutation probability is set to 0.1. We use generational reproduction with an



(a) (b)

Figure 1: Assuming the performance of both schedules being equal, the schedule depicted in Gantt-chart (b) may be favored because it preserves idle-time of machine M2 longer.

elitist strategy which ensures that the overall superior solution always survives to the next generation. An EA run consists of 100 generations with a population size of $N = 100$.

To apply an EA, the problem is decomposed into a series of static sub-problems on the basis of a rolling time horizon with event-triggered rescheduling. When new jobs arrive at time t, operations already scheduled to start before t are considered as implemented and are consequently discarded from further consideration. Since non-preemptive scheduling is assumed, the set of already implemented operations includes the ones currently processed.

For the new problem to be constructed, release times of jobs r_i are re-set to t or, in case of a currently processed operation, to its prospective completion time. Additionally, the time of the earliest availability of machines a_j ($1 \leq j \leq m$) is re-set to t, or if it is currently busy, to the completion time of the associated operation. Finally the newly arrived jobs are added to the problem.

A Flexibility Measure for Job Shop Problems

In this section, we describe the implementation of an anticipation term. According to the conjecture made in the introduction, schedules derived in a rolling time fashion should not only obey a prescribed performance criterion but should additionally take into account a flexibility measure. The so generated flexible schedules should be easier to adapt after the arrival new jobs, resulting in an improved performance over time (Branke & Mattfeld 2000).

Consider the simple example depicted in Figure 1: Let the due date for both jobs be at $t = 9$, i.e. the two alternative schedules (a) and (b) yield the same tardiness of 1 time unit. Nevertheless, the schedules differ in their distribution of machine utilization. When a new job arrives, it is likely that schedule (b) will be able to integrate it more easily, as machine M2 becomes available earlier than in schedule (a). Thus, by taking into account the times when the operations are processed, the system's flexibility may be increased.

With this introductory example in mind, we are going to reason what measurable aspects of a manufacturing system's state can have an impact on the system's flexibility, before we are going to integrate these aspects by modifying the objective function of the optimization algorithm.

Let us first review and comment on some aspects discussed with respect to the relationship between static and dynamic vehicle routing. Among other topics (Psaraftis 1988) argues that in the dynamic case (1) time dimension is essential, (2) resequencing decisions may be warranted, (3) near-term events are more important, and finally, (4) queuing considerations become important. In the following we discuss (1)–(4) in the context of dynamic scheduling in order to derive methodological implications.

1. We can identify a subtle difference between static and dynamic scheduling with respect to the consideration of time. In the static case, the problem is fully defined and the solution quality is fully described by the performance measure under consideration. In other words, machine idle-times are inserted as desired in order to obtain a schedule of optimal performance. In the dynamic case, the problem and thus the perception of what constitutes a good solution changes over time. In fact, there is no warranty that a sequence of optimal static schedules yields optimality also for the overall dynamic problem, therefore measures leading to an improved solution quality of the static problem will not necessarily be beneficial from a global perspective. The insertion of idle-times may lead to a waste of machine capacity, potentially causing future bottlenecks.
2. Whenever new jobs arrive in the dynamic case, the system benefits from rescheduling as it integrates new information as early as possible. We construct the next static prob-

lem from the the newly arrived jobs and from the backlog present at the current state of the system. The backlog is actually re-scheduled, although its dedicated machine idle-times may have already been partially implemented. Thus rescheduling might make the original reason for inserting idle-times obsolete, leading to a further potential waste of machine capacity.

3. In a non-deterministic shop flbor environment the operation of a certain job on a machine may be re-scheduled many times, but events scheduled early have a higher probability of being actually implemented before the next change, and are thus more certain than events scheduled later. Whenever machine idle-times are implemented, they are in danger to become a waste of machine capacity. Therefore, early machine idle-times should receive particular attention.
4. Because of the missing time horizon in static scheduling, a priori no machine utilization rates or queue lengths are considered. In dynamic scheduling the arrival process of jobs determines the long-term machine utilization, approximated by the ratio of processing times of jobs arrived and the time of machine availability in the arrival span considered. The already discussed waste of machine capacity due to the implementation of machine idle-times increases work in process which in turn decreases flexibility.

The above discussion reveals important differences between static and dynamic scheduling: Idle times in general, but early idle times in particular, may have a substantially negative effect on a system's flexibility. Clearly, when there is a choice between two schedules with equal quality according to the static goal, but unequal distribution of idle times, the schedule with less early idle time should be preferred.

We take a rather pragmatic approach to implement the above ideas: in addition to the standard quality measure (in this case mean tardiness of jobs $\overline{T}$), we introduce an idle time penalty S. Since we expect "early" idle-times to be of particular importance for dynamic scheduling, the penalty decreases linearly with the time t of its occurrence in the schedule, i.e.

$$w_j(t) = \begin{cases} \max\{0, 1 - t/(\beta - a_j)\} & : \quad \beta > a_j \\ 0 & : \quad \textit{otherwise} \end{cases}$$

evaluates idle-time on machine j at time $t = a_j$ with weight 1.0 and idle-time at time $t \geq \beta$ with weight 0.0, where β is a user-defined parameter denoting the length of the interval considered. We calculate S by summing up weighted idle-times over the machines $1 \leq j \leq m$.

An appropriate scaling of $\overline{T}$ against S can hardly be derived analytically. Fortunately, the evolutionary algorithm used for optimization always maintains a population of solutions, which can be used to normalize both terms independently to the interval $[0,1]$ on the basis of minima and maxima observed in the population. As the evolutionary algorithm's population changes over time, the normalization adapts accordingly. The fitness f_k of schedule/individual k $(1 \leq k \leq N)$ is constructed from a convex combination of both normalized terms with the parameter α being the weighting factor:

$$f_k = (1 - \alpha)\hat{T} + \alpha\hat{S}$$

with

$$\hat{T} = \begin{cases} \frac{\overline{T}_k - \min_l\{\overline{T}_l\}}{\max_l\{\overline{T}_l\} - \min_l\{\overline{T}_l\}} & : \quad \max_l\{\overline{T}_l\} > \min_l\{\overline{T}_l\} \\ 0 & : \quad \textit{otherwise} \end{cases}$$

$$\hat{S} = \begin{cases} \frac{S_k - \min_l\{S_l\}}{\max_l\{S_l\} - \min_l\{S_l\}} & : \quad \max_l\{S_l\} > \min_l\{S_l\} \\ 0 & : \quad \textit{otherwise} \end{cases}$$

With $\alpha = 0.0$ the schedule's idle-time is not considered at all, whereas with $\alpha = 1.0$ the tardiness does not contribute to f_k.

In this section we have argued that the insertion of machine idle-times can decrease flexibility in many cases. In Section we have pointed out that the insertion of idle-times also can increase efficiency. The remainder of this paper is devoted to a computational investigation on the role of machine idle-times in dynamic scheduling. We investigate whether an increase of flexibility without loss of efficiency is possible and furthermore whether an increase of flexibility can even improve scheduling efficiency.

Maintaining Flexibility by Anticipatory Scheduling

In this section, we evaluate the benefits obtainable by the suggested anticipation term. We examine its robustness with respect to several parameters. Furthermore we show how modifications effect the work in process, i.e. the average number of operations in the production system.

The anticipation term suggested is parameterized by the length of the time interval considered (β), and the emphasis on idle time (α). In order to examine the interaction between α and β, as well as their impact on the system performance under different load conditions, we run experiments for three different load conditions $U \in \{0.7, 0.8, 0.9\}$, in each case varying $\alpha \in \{0.000, 0.125, \ldots, 0.750\}$ and $\beta \in \{10, 30, \ldots, 130\}$. The largest β value of 130 approximately corresponds to the mean time of 10 job arrivals for $U \geq 0.7$, which seems to be a reasonable upper bound. For each combination of U, α and β the average over 30 different problem instances is reported. Together, a total of $3 \cdot 7 \cdot 7 \cdot 30 = 4410$ runs are performed.

The results observed are given in Figure 2. In order to focus on the mean tardiness while varying α and β, the percentage of improvement over $\overline{T}$ observed for $\alpha = 0.000$ is reported, i.e. for the reference parameterization idle-times are not considered at all and consequently the setting of β is of no matter. The plots on the left side of Figure 2 show iso-lines in steps of 10% for $U = 0.7$ and steps of 5% for $U = 0.8$ and $U = 0.9$. Since β is of no concern for $\alpha = 0.000$, all solutions along the y-axis represent the reference setting with zero improvement.

The shape of the iso-line plots resemble each other for the different U considered. The solution quality of the most

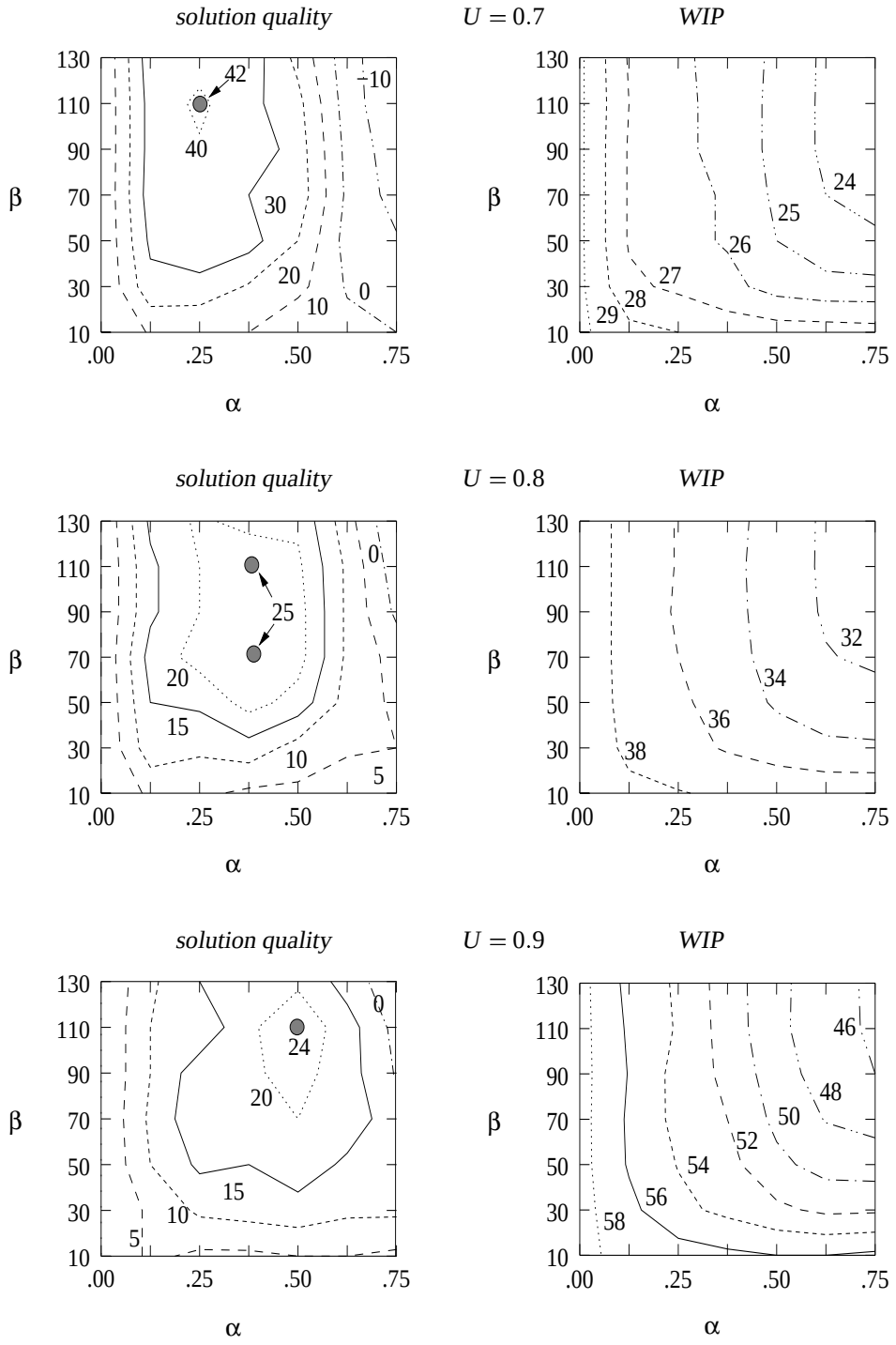


Figure 2: The effect of α and β on the improvement of the solution quality $\overline{T}$ in percent over the result obtained for $\alpha = 0.0$ (left), and the work in process (WIP) depicted as the average number of operations involved in deterministic sub-problems (right). Idle-times occurring in $[a_j, \beta]$ are penalized with linear decreasing weight.

Table 1: Percentage of $\overline{T}$ improvement over EA with $\alpha = 0.000$ and $\delta = 1.00$.

U	δ	α						
		0.000	0.125	0.250	0.375	0.500	0.625	0.750
	0.00	10	16	21	22	7	-27	-31
	0.25	11	25	32	31	11	-24	-29
0.7	0.50	15	39	40	41	19	-19	-28
	0.75	16	37	38	43	21	-13	-24
	1.00	0	34	42	33	23	-5	-19
	0.00	9	15	18	20	17	-5	-12
	0.25	13	21	22	27	21	-2	-11
0.8	0.50	12	21	26	30	25	3	-6
	0.75	5	19	26	30	23	5	-5
	1.00	0	14	20	25	21	9	-3
	0.00	14	17	23	27	26	17	6
	0.25	12	20	25	26	26	15	7
0.9	0.50	8	16	22	23	28	15	8
	0.75	6	14	21	22	27	16	7
	1.00	0	10	11	19	24	17	9

improving parameter setting is indicated by a gray dot. The improvements that can be obtained by penalizing early idle times are impressive and range from 24% for $U = 0.9$, over 25% for $U = 0.8$, to 42% for $U = 0.7$.

The area of significant improvements is rather broad, meaning that the approach is quite robust with respect to setting the parameters α and β. Only extremely high values of α and β (which basically means to completely ignore tardiness) lead to decreasing solution qualities. Of course, extremely low values of either α or β also eliminate the effect of anticipation.

Parameter α should be increased with increasing U, because the more often the schedule is turned over by newly arriving jobs, and the more congested the shop fbor system gets, the more important the avoidance of idle-times becomes. As a simple guideline, α may be set to 0.25 in cases where machine capacity is readily available and it should not exceed 0.5 even in a congested shop fbor.

Generally an appropriate length of the β interval depends on the degree of distortion caused by the arrival of new jobs. With regard to the problem at hand β can be fixed at 110 for all U considered. The relatively long time span of β approximates 8 to 12 job arrivals (depending on U). This indicates that a long term consideration of idle-times is important, even if the corresponding part of the schedule will overturn for several times until its eventual implementation.

The right hand side of Figure 2 shows the effect of the parameter variation on the average number of operations per sub-problem. A large U imposes a small interarrival-time which in turn leads to a large number of operations per sub-problem. It can be observed, that an increasing α clearly decreases the problem size. Here, the idle-time penalty leads to a more effective utilization of machine capacity which in turn results in shorter job fbw times and finally in a smaller number of operations per sub-problem.

However, although the problem size continues to shrink with increasing α, beyond a certain α value the solution quality deteriorates. That means that the observed improvements are not due to the smaller problem sizes, but rather that the reduction in problem size is a side-effect of the focus on increasing flexibility. Summarizing, anticipatory scheduling is a robust and easily implemented method, yielding remarkable improvements in solution quality.

The Effect of the Schedule Builder

Schedule builders are distinguished by generating either active or non-delay schedules. Although optimality conditions with respect to regular measures of performance suggest to search the set of active schedules, heuristics may benefit from a confinement towards non-delay scheduling

The concept of anticipatory scheduling suggests an additional viewpoint: Non-delay schedule builders implicitly avoid machine idle times whenever possible. Moreover "early" idle-times are specifically avoided at the expense of later idle-time insertions occurring due to the existence of precedence constraints. Hence for dynamic scheduling a non-delay schedule builder might be particularly advantageous because it implicitly follows the proposed idea of anticipatory scheduling. It might even be possible that non-delay scheduling renders an explicit consideration of flexibility in the objective function unnecessary.

To examine the interdependencies between the schedule builder and the anticipation term we conduct another set of experiments and vary the schedule builder's tendency to build active schedules by means of δ. Experiments are carried out for $U \in \{0.7, 0.8, 0.9\}$ with all combinations of $\alpha \in \{0.000, 0.125, \ldots, 0.750\}$ and $\delta \in \{0.00, 0.25, \ldots, 1.00\}$. Parameter β is set to 110, which has already been identified as appropriate for all U considered. Table 1 lists the percentage of improvement in solution quality ($\overline{T}$) over the EA parameterized with $\alpha = 0.000$ and $\delta = 1.00$ (no anticipation and active scheduling).

As can be observed in the first column ($\alpha = 0$), δ alone has a noticeable effect on the resulting tardiness values. However, the benefits that can be obtained from tuning δ are significantly smaller than those that can be obtained from tun-

ing α, i.e. non-delay scheduling is certainly no alternative to introducing a flexibility term into the objective function. But although not as powerful as a variation of α, δ can definitely lead to a further improvement of peak performance. In accordance with the results reported by (Bierwirth & Mattfeld 1999), the higher the utilization is, the smaller are the appropriate δ values. Except for $U = 0.7$, where δ close to 1.00 seems to be a reasonable setting anyway, additional improvements in solution quality of 5% for $U = 0.8$ and 4% for $U = 0.9$ are obtainable.

For a closer examination of the interaction effects of α and δ, we run an analysis of variance (ANOVA) on the data set with $\overline{T}$ as dependent variable and α and δ as independent variables. Since U has an obvious impact on the fitness obtained, U is considered as covariate in the statistical experiment. As one might expect, there is a strong significance of the F-Test for α. For δ we still observe a weak significance of 0.057. For the interaction of α and δ no significance at all can be observed.

The above results confirm that α has a major influence on $\overline{T}$ obtained. The role of δ is less significant, but ANOVA confirms our observation that $\overline{T}$ partially depends on δ regardless of other parameter settings. Analyzing Table 1 more closely we observe a strong δ-dependency for small α values, whereas for larger α values the dependency vanishes with increasing U. For example, for a congested shop fbor with $U = 0.9$ and $\alpha \geq 0.5$ we no longer recognize an influence of δ, which explains the weak significance reported by ANOVA.

The insignificant interaction between α and δ imply that the parameters can be tuned almost independently of each other. However, $\overline{T}$ improvements with regard to α and δ are not additive. This indicates that α and δ can substitute each other to a certain extent.

Unfortunately, the impact on the EA of searching a smaller search space and the supposed implicit flexibility due to non-delay scheduling can hardly be separated from each other. For instance, in case of a congested shop fbor the EA may benefit from a decreasing δ because of the resulting smaller search space. At the same time the decreasing δ may lead to an increased flexibility by avoiding early machine idle-times. Both effects would positively contribute to the overall solution quality. Therefore, we can conclude that the potential benefit of increased flexibility through non-delay scheduling is even smaller than the figures in Table 1 suggest. Overall, it is obvious that non-delay scheduling is no substitute for introducing a flexibility term into the objective function.

Conclusion

Many real-world optimization problems change over time and require repeated optimization as new information becomes available. A standard approach to deal with these dynamics is to use a rolling time-horizon. The problem is decomposed into a series of static sub-problems, which are then solved independently.

In this paper, we have argued that, if the decisions at one stage influence the problems encountered in the future, future changes need to be anticipated by searching not only for good solutions, but for solutions that additionally influence the state of the problem in a favorable way. We have identified these solutions as being *flexible*, i.e. easily adjustable to changes in the environment.

To validate this conjecture, we have considered a dynamic summed tardiness job shop scheduling problem, with jobs arriving non-deterministically over time. For this particular application, we concluded that the avoidance of early idle times can support the flexibility of a solution.

As a consequence, we suggested to extend the goal function by an anticipation term which penalizes early idle times. This algorithm showed impressive improvements in solution quality of up to 58% in comparison to an advanced priority rule based approach. Short-term efficiency-losses due to a flexibility term added to the scheduling algorithm lead to significant long-term gains.

References

Adam, N., and Surkis, J. 1980. Priority update intervals and anomalies in dynamic ratio type job shop scheduling rules. *Management Science* 26:1227–1237.

Bierwirth, C., and Mattfeld, D. C. 1999. Production scheduling and rescheduling with genetic algorithms. *Evolutionary Computation* 7(1):1–18.

Branke, J., and Mattfeld, D. 2000. Anticipation in dynamic optimization: The scheduling case. In Schoenauer, M.; Deb, K.; Rudolph, G.; Yao, X.; Lutton, E.; Merelo, J. J.; and Schwefel, H.-P., eds., *Parallel Problem Solving from Nature (PPSN VI)*, volume 1917 of *LNCS*, 253–262. Springer.

Branke, J. 2001. *Evolutionary Optimization in Dynamic Envionments*. Kluwer.

Cheng, R.; Gen, M.; and Tsujimura, Y. 1999. A tutorial survey of job-shop scheduling problem using genetic algorithms-II: Hybrid genetic search strategies. *Computers and Industrial Engineering* 36:343–364.

Church, L., and Uzsoy, R. 1992. Analysis of periodic and event-driven rescheduling policies in dynamic shops. *International Journal of Computer Integrated Manufacturing* 5:153–163.

Fang, J., and Xi, Y. 1997. A rolling horizon job shop rescheduling strategy in the dynamic environment. *Int. Journal of Advanced Manufacturing Technology* 13(3):227–232.

Farn, C.-K., and Muhlemann, A. 1979. The dynamic aspects of a production scheduling problem. *International Journal of Production Research* 17:15–21.

Jensen, M. T. 2001. Improving robustness and flexibility of tardiness and total flow-time job shops using robustness measures. *Applied Soft Computing* 4:1–18.

Kutanoglu, E., and Wu, S. D. 1998. Improving schedule robustness via stochastic analysis and dynamic adaptation. Technical Report 98T-001, Lehigh University.

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

Mehta, S., and Uzsoy, R. 1998. Predictable scheduling of a job -shop subject to breakdowns. *IEEE Transactions on Robotics and Automation* 14(3):365 –378.

Morton, T. E., and Pentico, D. W. 1993. *Heuristic Scheduling Systems*. Wiley Publishers.

Muhlemann, A.; Lockett, A.; and Farn, C.-K. 1982. Job shop scheduing heuristics and frequencies of scheduling. *Internaltional Journal of Production Research* 20:227–241.

Norman, B., and Bean, J. 1997. A random keys genetic algorithm for job shop scheduling problems. *Engineering Design and Automation Journal* 3:145–156.

Panwalkar, S., and Iskander, W. 1977. A survey of scheduling rules. *Operations Research* 25(1):45–61.

Psaraftis, H. N. 1988. Dynamic vehicle routing problems. In Golden, B. L., and Assad, A. A., eds., *Vehicle Routing: Methods and Studies*, volume 16 of *Studies in Management Science and Systems*. North-Holland. 223–248.

Raman, N., and Talbot, F. B. 1993. The job shop tardiness problem: a decomposition approach. *European Journal of Operational Research* 69:187–199.

Snoek, M. 2001. Anticipation optimization in dynamic job shops. In *GECCO Workshop on Evolutionary Algorithms for Dynamic Optimization Problems*, 43–46.

Storer, R.; Wu, D.; and Vaccari, R. 1992. New search spaces for sequencing problems with application to job shop scheduling. *Manufacturing Science* 38:1495–1509.

Yamamoto, M., and Nof, S. 1985. Scheduling/rescheduling in the manufacturing operating system environment. *International Journal of Production Research* 23:705–722.

The Challenge of Planning and Execution for Spacecraft Mobile Robots

Gregory A. Dorais and Yuri Gawdiak

NASA Ames Research Center
MS 269-2
Moffett Field, California 94035
gdorais@arc.nasa.gov, ygawdiak@arc.nsa.gov

Abstract

The need for spacecraft mobile robots continues to grow. These robots offer the potential to increase the capability, productivity, and duration of space missions while decreasing mission risk and cost. Spacecraft Mobile Robots (SMRs) can serve a number of functions inside and outside of spacecraft from simpler tasks, such as performing visual diagnostics and crew support, to more complex tasks, such as performing maintenance and in-situ construction. One of the predominant challenges to deploying SMRs is to reduce the need for direct operator interaction. Teleoperation is often not practical due to the communication latencies incurred because of the distances involved and in many cases a crewmember would directly perform a task rather than teleoperate a robot to do it. By integrating a mixed-initiative constraint-based planner with an executive that supports adjustably autonomous control, we intend to demonstrate the feasibility of autonomous SMRs by deploying one inside the International Space Station (ISS) and demonstrate in simulation one that operates outside of the ISS. This paper discusses the progress made at NASA towards this end, the challenges ahead, and concludes with an invitation to the research community to participate.

Introduction

In order to robustly achieve increasingly ambitious mission goals for longer periods with less ground support than traditionally required, we expect future space flight projects to increasingly require advanced onboard autonomy to support both manned and unmanned missions. Moreover, autonomously-controlled mobile sensors and manipulators (that can be encapsulated in a SMR) can provide additional capabilities and productivity that would otherwise require greater mission cost or risk.

Sensing Tasks

Generally, sensing tasks are viewed as more readily achievable than tasks that require sensing and manipulation. As such, the systems that we are initially developing are spacecraft robots restricted to mobile sensing and this paper is restricted to discussing planning and execution of such robots. Consider a SMR (a mobile robot with a variety of sensors) that can operate within a spacecraft such as the International Space Station (ISS).

Figure 1: International Space Station Illustration

Such a robot could potentially perform a number of tasks such as:

- Measuring and localizing toxic gases. In former Russian MIR space station, there was concern that batteries might leak sulfur dioxide. During a fire on MIR, toxic gases were released. In both cases it would have been helpful to have a SMR measure and, if necessary, localize the source of such gases.
- Measuring changes in pressure and ratios of nominal gases, e.g., oxygen and carbon dioxide. The first crew of the Salyut 1 station all tragically died by suffocation when a valve failed on its return vehicle. Fixed sensors can also fail or not be available during a crisis such as happened on MIR when a collision caused a loss of cabin pressure down to ~600mb, far below the safety level. A SMR can provide early warning of anomalies and be a redundant, portable system during a crisis.
- Validate fixed environmental sensing systems. In event that an anomaly is detected by the ISS life support system, there exists the possibility that the problem is a fixed sensor and not the environment. A SMR can be autonomously deployed or controlled by Mission Control to validate if the sensor is defective or not. If the sensor is defective, the SMR can act as a virtual sensor until the

fixed sensor is replaced. If not, the SMR help isolate the source of the anomaly.

- Visually validate regions of the spacecraft. Multi-spectral cameras on a SMR can provide crew members, Mission Control, and scientists a visual record of anything from a piece of equipment, to a crew activity, to a science experiment, without tying up a crew member to perform the task.
- Perform time-consuming special monitoring tasks. Specially-equipped SMRs can be deployed to specific tasks such as measure or localize certain sounds. Detecting unusual sounds is a method often used by people to diagnose a failing piece of equipment. Also, small leaks can be detected by the sound they emit. An autonomous SMR can isolate and localize particular sounds that human ears cannot detect.

An example of a task for a SMR operating outside a spacecraft is:

- Detecting external spacecraft damage. Astronaut EVAs are risky and time consuming. As a result, monitoring tasks such as checking the Shuttle for tile damage prior to reentry and looking for micrometeorite damage on ISS are not routinely performed. Once extended to remote spacecraft, failure assessment alone is of enormous value. Extraordinary effort is made to determine failure causes often with low confidence due to lack of data.

SMR and Terrestrial Mobile Robot Comparison

Although there are many similarities between SMRs that operate in engineered dynamic environments that may include people, and mobile robots that operate in natural terrains other than Earth, there are also striking differences that present challenges for SMRs including:

- Operates in close range in complex, dynamic, structured environment in 3 dimensions.
- Recognizes, and in some cases manipulates, many engineered objects
- Observes nominal and diagnoses off-nominal situations
- Interacts with people in a number of ways:
  - People are commanders (at various levels of authority to command at various levels of autonomy)
  - People are agents instructed by robot to achieve goal
  - People are dynamic obstacles to avoid
  - People are dynamic objects to track
  - People are peers to collaborate on achieving joint goals

These tasks and the operational environment levy a number of requirements on the planner(s) used to achieve such tasks over an extended period:

- Mixed-initiative task planner/scheduler
- Mixed-initiative path planner
- Local obstacle avoidance path planning
- Resource management
  - Power & Energy
  - Momentum
  - Thermal power management
  - Battery-life
- Multi-agent state estimation and control (people, SMRs, in-situ systems)
- Reactive planning and adjustably autonomous control
- Real-time planning and execution

Spacecraft Mobile Robots at NASA

NASA has begun to address the need for SMRs and the above challenges. Currently, two spacecraft mobile robots in particular are under development at NASA, the Personal Satellite Assistant (PSA), and the Sprint AERCam. This paper will focus on the PSA all many of the issues and technologies are relevant to both.

Personal Satellite Assistant (PSA)

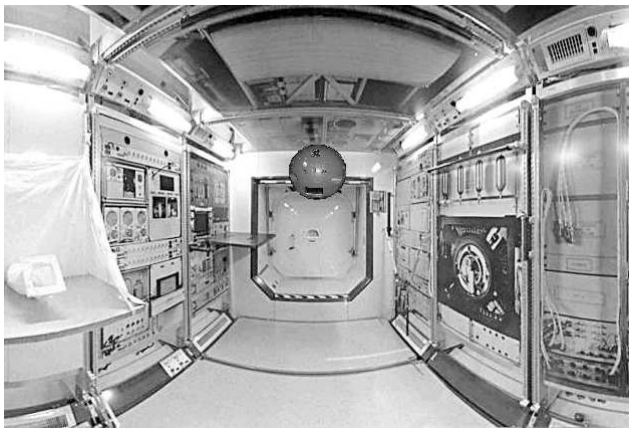


Figure 2: PSA Prototype depicted in ISS Node Mockup

The PSA is being designed as a softball-sized flying robot that operates autonomously onboard manned and unmanned spacecraft in micro-gravity, pressurized environments, and in particular onboard ISS. PSA's hardware architecture is being designed to accommodate a wide range of components that enable a broad set of mission support scenarios. Environmental sensors for gas, temperature, and pressure provide the ability for the PSA to monitor spacecraft for abnormal conditions, e.g., overheating equipment, payload and crew conditions. Video and audio interfaces will provide support for navigation, remote monitoring and video-conferencing. A radio frequency identification tag reader/writer and/or bar-code reader on the PSA will enable it to recognize specific objects and update their location in an inventory control system. Ducted fans/blowers will provide propulsion and batteries will provide portable power. An auto-docking locker will enable the PSA to autonomously recharge its

batteries and provide a secure storage location when not in flight. The PSA will be connected by a wireless network to a laptop computer that will provide a user interface with the crew and to a server for additional information processing capacity, primarily for PSA planning. A speech interface and dialogue management system for the PSA will permit spoken language commanding and data queries of the PSA and databases that the PSA has access to via its wireless network. A long-range goal for the PSA is to connect it via the wireless network to the spacecraft's avionics data, payload networks, and uplink/downlink communications.

The main benefit PSA is expected to provide is for it to act as a crew work-force multiplier by performing intra-vehicular activities on behalf of the crew. Current spacecraft are constrained in terms of crew size, power, volume, and computing resources. Crew time on the International Space Station is one of the most constrained resources and is projected to cost hundreds of dollars per minute per astronaut. The crew will have to maintain complex critical ISS systems, perform dozens of major simultaneous payload experiments, and perform general housekeeping. Enhancing the crew's ability to perform their duties is critical for successful, productive, and safe space-based operations. Moreover, PSA can enhance crew safety by performing monitoring tasks that might endanger a crewmember or not otherwise be performed.

The PSA's autonomy capabilities are expected to significantly improve productivity by directly supporting flight crews, ground controllers, and the principle investigators of science experiments. The biggest benefits to those users will come from its ability to monitor the environment, e.g., detect abnormal concentrations of CO^2, act as a mobile camera/camcorder/data terminal, and track inventory using advanced inventory micro-tags. For example, when the PSA detects a sharp pressure drop while performing an inventory audit, it would then notify the crew of the abnormal condition and attempt to localize it. If however, a fixed sensor on ISS detected a pressure drop, the PSA could be used to validate the reading. If the sensor is diagnosed as defective, the PSA could act as a temporary replacement sensor. We expect this entire activity could be conducted without the need for human intervention or be initiated by the ground operators, onboard crew, or the spacecraft itself.

The PSA will provide an additional side-benefit by acting as an autonomy and mobile robot testbed for researching intra-vehicular robots that eventually will be used for long-term missions, e.g., operating onboard a crew return vehicle orbiting Mars for two years while the crew explores the surface.

PSA Operational Requirements

In order to support the development of suitable autonomous control system for the PSA, the following subset of operational requirements were defined:

1. Achieve set of 10 commands in an optimal sequence where each command is to take a picture and environmental sensor reading at a global <x, y, z, yaw, pitch, roll> specified immediately prior to execution. Perform in each of the following ISS Node environments:
Environment A: uncluttered, static
Environment B: known clutter, static
Environment C: unknown clutter, static
Environment D: unknown clutter, dynamic

2. Validate two environment fixed-sensors. For example, go to the location of a fixed sensor indicating high temperature and measure environment. If the fixed sensor is accurate, localize the source of the heat. If the fixed sensor is not accurate, station-keep at the fixed sensor location transmitting temperature readings until the fixed sensor readings are accurate then return to base locker.

3. Demonstrate mixed-initiative planning for both path and deliberative planning. This shall include:
a. Adding temporary constraints to change an existing plan
b. Adding goals to an existing plan
c. Rejecting goals in an existing plan
d. Rejecting goals from a plan that fails to converge

4. Demonstrate mixed-initiative execution. Includes allowing human interrupts and command additions, retractions, & modifications as well as asking humans or other agents for assistance during execution. Levels of autonomy to be demonstrated:
a. High-level teleoperation
b. Guarded & guided teleoperation
c. Dynamic commanding of PSA by human
d. Dynamic commanding of PSA by another agent
e. Dynamic commanding of human by PSA
f. Dynamic querying and modification of plan currently being executed
g. Executing and modifying generated plan due to environment uncertainty

5. Demonstrate teleconferencing. Includes face-tracking.

6. Demonstrate crew following. Includes body-tracking.

7. Demonstrate energy resource management including dynamic auto-recharging.

8. Demonstrate leak isolation using acoustics and a leak isolation expert agent.

9. Demonstrate spoken language commanding and status reporting.

10. Demonstrate inventory sensing and location tracking.

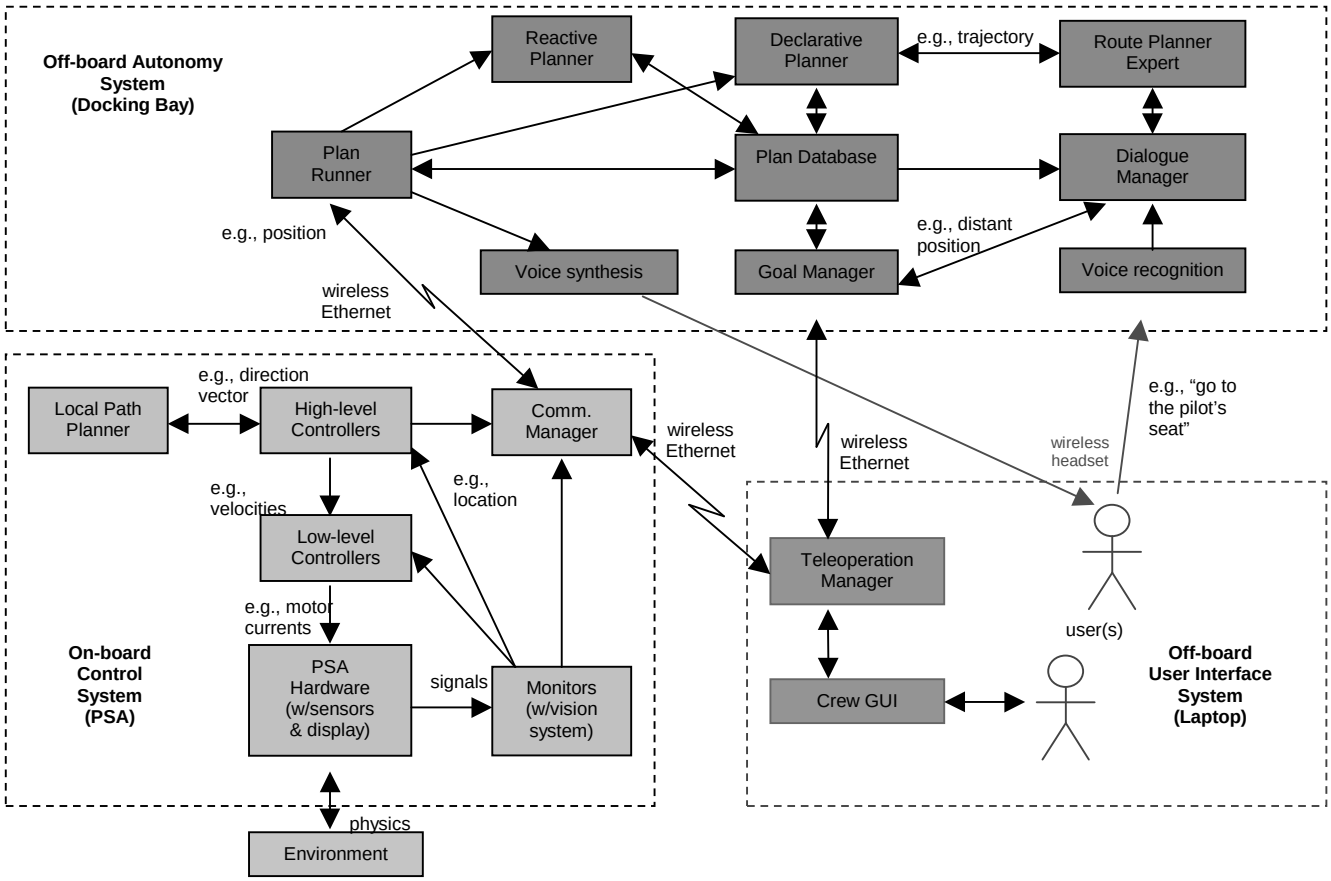


Figure 3: PSA Top-level Autonomy Architecture

PSA Autonomy Control Architecture

A prototype autonomy control architecture, illustrated in figure 3, has been developed to address the operational requirements. The architecture implementation was distributed over three processors as depicted by the dashed boxes:

- Onboard flight processor for sensing and real-time control. Software for localization to a global map, object recognition, and obstacle avoidance using stereo vision and other proximity and inertia sensors is executed here.
- User-interface laptop for commanding and displaying information. This includes interfaces for interactively creating and modifying the plan and teleoperation. Our intent is for this interface to support operation at various autonomy levels that can be dynamically changed and range from teleoperation to high-level autonomous control.
- Off-board docking bay processor for high-level autonomous control including planning, scheduling, command sequencing, and human and other agent communication and coordination.

The high-level autonomous control system, depicted by the top dashed box in figure 3, is a planning and execution system in its own right based on the unified agent framework described in [Muscettola et al. 2000]. This agent is composed of the following subsystems:

- **Plan Database**

This is a temporal, constraint-based network of tokens that defines the past, the present, and flexibly-defined future states and actions of the system. Each token represents the "state" of a state variable for a period of time. The token data structure is a tuple that specifies the state variable, the procedure and its arguments that is invocated when the token is "executed," and the token start and end time bounds. The plan database supports multiple timelines with constraints on and between tokens. If none of the constraints are violated for a given instantiation of the plan database, the database is defined to be consistent. The current implementation uses a next-generation plan database of the Remote Agent plan database described in [Jonsson et al. 2000], which was part of the Remote Agent control system demonstrated on the Deep Space One spacecraft in 1999 [Bernard et al. 1998].

- **Plan Runner** (command sequencer)

The plan runner is a process responsible for "executing" tokens in the plan database at the appropriate time. Executing a token involves calling the procedure with its arguments defined by the token, updating the plan database with the token return values when the procedure terminates, constraining the plan database so that planners only have limited ability to change the past, and calling planners, described below, as needed to update the plan database. The plan runner implemented is described in more depth in [Muscettola et al. 2000].

- **Planners**

This architecture support the integrated use of a number of planners so that planners can be specialized for various functions depending on the domain requirements. For the purposes of this paper, with the exception of the plan runner, a planner is any process that modifies the plan database or provide information to be added to the plan database at the request of a planner. The planners in this implementation include:

1. Declarative Planner

The declarative planner is based on the Remote Agent Planner/Scheduler described in [Jonsson et al. 2000]. It is responsible for generating a consistent, flexible plan in the plan database given a start and end horizon time bound, an initial state of the timelines at the start time, and a set of goals. A flexible plan is loosely defined as a set of timelines, each consisting of tokens on each timeline, token order constraints that prevent overlapping tokens on the same timeline, and token procedure variable constraints. Plan flexibility is characterized by the set of decisions yet to be made in a plan database that is consistent. The declarative planner is called to initialize the plan database and also is called during plan execution as specified by the plan being executed. It is typically called to plan for a period of significant duration sufficiently in the future such that the deliberative planner will complete prior to the start time of this period, but not so far in the future that the initial state at the future start horizon is not known with high confidence.

2. Reactive Planner

The reactive planner is also based on the Remote Agent Planner/Scheduler described in [Jonsson et al. 2000], but typically uses different heuristics. It is regularly called by the plan runner to insure that the plan database is consistent after token return values are posted to the database (repairing the plan as necessary), to insure the database contains a token on each timeline being executed or to immediately start executing, and to remove any ambiguity in whether a token is ready to execute and what its procedural arguments are.

3. Goal Manager

The goal manager essentially acts as a meta-planner for the declarative planner. As stated above, the declarative planner requires a start and end horizon time bounds, an initial state of the timelines at the start time, and a set of goals. The goal manager interacts with the user to determine this information. This may include negotiation of goals when all goals are not achievable or supporting mixed-initiative planning for hypothetical situations.

4. Route Planner Expert

The route planner expert is called by any one of the above planners to determine the time, route, and energy required to move between two points in the environment or to cover a certain space. It has access to a global map that can be updated with sensed obstacles. A route plan request is typically made by the deliberative planner as part of developing the initial plan, but may also be called by the reactive planner to develop an alternate route if necessary, e.g., the route is blocked or there is insufficient energy to complete the current plan. In addition, a user may initiate a request to answer a hypothetical question about a particular goal.

- **Spoken Language Interaction**

A simplified abstraction of the spoken language interaction system can be viewed as consisting of the following three subsystems:

1. Dialogue Manager

The dialogue manager is responsible for acting as an intelligent interface between a person speaking a restricted natural language and the planner modules along with the plan database. New goals can be inserted or removed in the plan database, and queries can be made by spoken commands.

2. Voice Recognition

The voice recognition subsystem essentially converts an audio signal into a parsed text stream. In the past, we have used commercial products to accomplish this. We anticipate that we can continue to use such products, upgrading them as improvements are made. However, it may be necessary to filter the audio signal for noise.

3. Voice synthesis

Conversely, the voice synthesis subsystem essentially converts text to speech. Similarly, we use a commercial product for this purpose.

Current State of the PSA Project

The PSA project began in 1998 and according to the current project schedule, the PSA begins flight operation in 2006. At this time, an oversized version of the flight model has been developed and is being tested on a granite table and is supported by a test stand with a compressor that enables the prototype to float on a thin cushion of air. On this test facility, we have demonstrated visual-servoing to various locations as well as vision-based localization to a global map. A 3D test facility that will house a full-size station node mockup is nearing completion. With the aid of a crane-like support mechanism and gimble, the PSA prototype will be able to move in 6 degrees-of-freedom (DOF), i.e., (X, Y, Z, yaw, pitch, roll) as if it were in a micro-gravity environment. The facility will also enable crewmembers to interact with the PSA in this environment while being suspended by a sling. A next-generation

version of the prototype is also under development and is scheduled for testing in 2003.

In addition to the physical hardware for testing, a simulator has been developed. The simulator primarily reads the force commands generated by the controller and moves the PSA in an ISS module accordingly. It also provides simulated PSA sensors signals, e.g., vision, temperature, at various fidelities depending of the required tests. Although, the simulator is typically operated in force mode, it can also be operated in velocity or position modes when it is desirable to interact directly with high-level control systems. The PSA motion along with dynamic obstacles and in-situ crewmembers are rendered in 3D. The simulator also supports multiple PSAs. In addition, the simulator supports scripted environmental events, such as a fire.

An initial version of the spoken language interaction system has been developed and tested with a simplified PSA simulation. The system has also been integrated with the plan database such that the database can be queried and modified in simple ways in response to spoken commands.

An initial version of the autonomous control system has also been developed and deployed, although certain modules, namely the goal manager and the route planner expert have been stubbed at this time. Although currently the reactive planner has been integrated and used by the system to accomplish simple scenarios, scenarios involving plan repair are not scheduled until later this year.

Sprint AERCam

In contrast to the PSA, the AERCam is being designed to operate in unpressurized regions, essentially outside spacecraft, primarily the ISS. However, in many other respects the planning and execution challenges are similar to those faced by the PSA.

Figure 4: Sprint AERCam during 1999 Flight Test

The Sprint AERCam is a teleoperated, free-flying spherical robot. It weighed about 35lbs and was 14” in diameter. It had 12 nitrogen-gas thrusters, each producing about 0.08lbs of thrust, for propulsion and attitude control. It was designed to operate for about 7 hours outside of and near spacecraft at low velocities relative to the spacecraft, less than 30cm/s. Its primary mission sensors are two color video cameras. Its primary function is to provide video supporting a crew extra-vehicular activity (EVA) or perform reconnaissance in lieu of an EVA. The Sprint was successfully flight-tested for about 30 minutes on the STS87 space shuttle flight in 1999.

Two limitations of AERCam are its size and the teleoperation requirement. In order to address these limitations, a mini AERCam is being developed and efforts have begun to develop an autonomous control system that will enable it to be autonomously controlled at levels varying from entirely teleoperated to entirely autonomously controlled.

The PSA and AERCam projects are coordinated so that they can leverage each others technologies, but it remains to be seen the extent that the autonomy architectures will be similar due to different operational requirements.

Challenge: Spacecraft Mobile Robot Scenarios

In order to measure the system capabilities with reference to the operation requirements and to identify the challenging problems, several scenarios have been developed. These scenarios were designed to be executed both in simulation as well as with the prototype hardware in the test facilities. The current scenarios that the system is being designed to address are:

Scenario A: Robust generation of an ISS node environment map

Description:
PSA will create an environment map of the ISS node by traversing the space in a serpentine path recording the environment sensor readings along the way. During this activity, its path will be blocked by static obstacles (some of which are known of ahead of time) and moving obstacles. At one point the PSA will be interrupted to be teleoperated and then perform a station-keeping task at a location specified by an ISS Rack Locker name, after which it will complete its original environment-mapping task.

Purpose:
- Demonstrate navigation to several waypoints in an environment that has static and dynamic obstacles.
- Demonstrate mixed-initiative execution including autonomous task interruption and resumption, guarded teleoperation, and visual servoing by command.
- Demonstrate generation of a near-optimal 6-DOF route plans
- Demonstrate obstacle detection and avoidance
- Demonstrate stereo vision-based 6-DOF localization and map registration

Scenario B: Participate in the diagnosis and recovery of an ISS node fault

Description:
A fixed sensor in the ISS node signals a high temperature to the Environmental Control Life Support System (ECLSS). However, it is not known whether the sensor is defective or the source or the heat. PSA is given a command by ECLSS to go the fixed sensor location and verify the temperature at that location. If PSA confirms the fixed sensor is correct, PSA is to locate the heat source and signal the source to ECLSS, will then power down the locker at that location. Once PSA verifies that the temperature has returned to normal, it returns to its docking bay. If the fixed sensor is not correct, PSA is to stay at that location until the fixed sensor is made operational. Once PSA verifies the sensor, PSA returns to its docking bay.

Variation Summary:
1. Perform with faulty fixed sensor
2. Perform with overheating locker

Purpose:
- Demonstrate IVHM
- Demonstrate cooperative multi-agent planning and execution
- Demonstrate generation of a near-optimal 6-DOF route plans
- Demonstrate stereo vision-based 6-DOF localization and map registration

Scenario C: Fault Detection and Cooperative diagnosis of an ISS node atmosphere leak

Description:
PSA is commanded to perform a routine task to monitor an ISS locker. While en route, PSA detects a drop in pressure in the node. It interrupts its current task and performs a set of directional microphone sensor readings to determine the cause is a leak to space and then PSA isolates the general location of the leak. PSA reports this information to ECLSS, which then dispatches and external SMR (AERCam) to the general location outside station where it images the region of the leak to get visual confirmation.

Purpose:
- Demonstrate autonomous IVHM
- Demonstrate dynamically changing plan to respond to fault detected in the environment
- Demonstrate multi-agent cooperative diagnosis

Scenario D: Cooperative Data Collection and Crew Instruction for Performing Interactive Mission Science Experiments

Description:
Crewmember commands PSA to follow the crewmember to an ISS rack where the crewmember will perform an experiment. When the crewmember arrives, he/she commands PSA to point at the locker where the crewmember will work. The crewmember commands PSA to start recording the video and audio. The crewmember then commands PSA to brief him/her on experiment X then instruct him/her on the first step of the experiment. Once the crewmember completes that step, he/she requests the next step and so on until all steps of the experiment are completed. The crewmember then commands the PSA to visually servo to his/her face to record a summary of the experiment while the crewmember is moving. The crewmember then instructs PSA to stop recording and return to its docking bay, which it does.

Purpose:
- Demonstrate automated data collection
- Demonstrate human – autonomous system collaboration
- Demonstrate autonomous teleconferencing with face-tracking
- Demonstrate person following
- Demonstrate automated task instruction
- Demonstrate spoken language commanding and reporting

Scenario E: Long-term mixed-initiative planning and optimization including inventory tracking

Description:
PSA is given a list of visual servoing goals with time constraints and is requested to generate a near-optimal plan to achieve the goals. The goals will be such that it will be necessary to schedule multiple battery recharges in order to achieve them. The operator will dynamically change the plan prior to its execution. During the execution, PSA will monitor the location of inventory items it senses as it passes by. PSA will encounter static and dynamic obstacles in the environment. Due to an inaccurate battery model, PSA will have to replan to prevent running out of power prior to recharging at the docking bay. Once PSA has completed the goal list, it given a list of inventory items to locate, some of which it passed by. PSA responds with the locations of the items it senses and then generates a plan to explore the areas of the ISS node it did not previously explore in order to locate the other items.

Purpose:
- Demonstrate near-optimal path plan generation
- Demonstrate resource planning
- Demonstrate static and dynamic obstacle avoidance
- Demonstrate mixed-initiative plan generation
- Demonstrate spoken language commanding and reporting
- Demonstrate inventory item sensing and location tracking

Invitation to the Research Community

As previously discussed, as part of the development process for the PSA, a simulation has been developed that supports operating multiple PSAs in the ISS and interacting with in-situ crew members and dynamic obstacles in 3D. If there is sufficient interest by the research community in exploring this domain, a version of this simulation, including the simulated hardware and environment but without the autonomy, control and spoken language software, may be made available for distribution to the research community in order to encourage research in this area. Please email gdorais@arc.nasa.gov if you would be interested in such a simulation and to signify support for its release.

Summary

Spacecraft Mobile Robots, such as PSA and AERCam provide a challenging domain for a number of planning and execution problems. By developing a modular software architecture and realistic simulator, a wide number of planning and execution approaches can be analyzed. Moreover, the overall system can be incrementally improved as new planning technologies are developed. Making the simulator, scenario definitions, and operation requirements available to the research community is viewed as one way to encourage the development of such technologies that operate in a real-world environment.

Acknowledgements

The authors acknowledge the outstanding contributions of the entire PSA team and others from the NASA Ames Autonomy and Robotics Area and the Systems Engineering Division. We also acknowledge the financial support for this project by the NASA Cross-Enterprise Technology Development program and the NASA Engineering Complex Systems program. Images are courtesy of NASA JSC.

References

D. E. Bernard, G. A. Dorais, C. Fry, E. B. Gamble Jr. B. Kanefsky, J. Kurien, W. Millar, N. Muscettola, P. P. Nayak, B. Pell, K. Rajan, N. Rouquette, B. Smith, B. C. Williams, 1998. "Design of the remote agent experiment for spacecraft autonomy." In Proc. of the IEEE Aerospace Conference.

A.K. Jonsson, P. Morris, N. Muscettola, K. Rajan, B. Smith 2000. "Planning in interplanetary space: theory and practice", in Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems (AIPS '00), Breckenridge, Colorado.

N. Muscettola, G. Dorais, C. Fry, R. Levinson, C. Plaunt, 2000. "A unified approach to model-based planning and execution," in Proceedings of the Sixth International Conference on Intelligent Autonomous Systems, Venice, Italy.

Predictable Multiprocessor Scheduling in Manufacturing Systems underlying Hard Real-Time Constraints

Dania A. El-Kebbe
Paderborn University / Heinz Nixdorf Institute
Fürstenallee 11, 33102 Paderborn, Germany
Tel.:+49-5251-606494, Fax.:+49-5251-606502
E-mail: elkebbe@uni-paderborn.de

Abstract – **This paper deals with predictable aperiodic scheduling of aperiodic tasks upon multiprocessor production stages in a manufacturing system underlying hard real-time constraints. The uniform multiprocessor scheduling algorithm presented is analyzed by considering its performance when it is allowed to run on faster machines. The predictability of the system is proved through schedulability analysis techniques and illustrated by an example.**

Index Terms – Real-time Systems scheduling, Production Planning and Control.

1 INTRODUCTION

The use of state-of-the-art real-time techniques in manufacturing planning and control is still rare. The lack of competence in real-time theory among production engineers and the lack of commercially available tools are the major reasons for this.

Real-time manufacturing systems must be able to handle not only periodic tasks, but also aperiodic tasks. Periodic tasks are used to implement off-line pre-planned requests. While periodic tasks in real-time manufacturing systems have hard deadlines, aperiodic tasks may have soft, hard or no deadlines at all. When aperiodic tasks have hard deadlines, the goal of the system is to allow the production of aperiodic tasks without jeopardizing the schedulability of hard periodic tasks. This problem is illustrated in Figure 1. Figure 1 presents a production line consisting of four production stages (Broaching, Machining, Galvanic, and Assembly) designed according to the MFERT-model presented by 12). This figure indicates a production stage to show that not only pre-planned (periodic) tasks are schedulable upon a uniform multiprocessor platform (using a production planning and control tool like OOPUS-DPS[1]) but also aperiodic order requests.

[1]OOPUS-DPS is an object-oriented planning tool developed by the workgroup of Prof. Dr. habil. W. Dangelmaier at the Heinz-Nixdorf Institute in Paderborn. For further information, please visit the web-page: http://wwwhni.uni-paderborn.de/cim/projekte/oopus-dps.php3

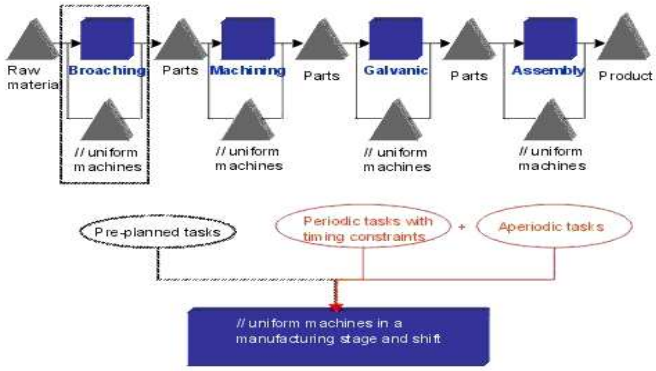


Figure 1: The on-line scheduling problem of a production planning and control system

To cope with various unexpected events in production planning and control, production engineers adopt a rescheduling policy (10). Such rescheduling policies yields to the following drawbacks:

1. Rescheduling policies are feasible for small-sized and simple manufacturing systems. As manufacturing systems grow in size and complexity, a rescheduling policy becomes impracticable.

2. Additionally, no rescheduling policy is made on-line, in the sense that rescheduling policies are unfortunately executed at the end of a production shift.

3. Furthermore, no prediction can be made concerning unexpected arriving requests.

2 State-of-the-Art

The scheduling of real-time systems has been much studied, particularly upon uniprocessor platforms. In multiprocessor platforms, there are several processors available upon which jobs may execute. Recently steps have been taken towards obtaining a better understanding of real-time scheduling on identical multiprocessors (11), (8), (1), and (2). However, not much is known about real-time scheduling on uniform or unrelated processors.

Furthermore, task scheduling in hard real-time systems can be *static* or *dynamic*. A static approach calculates schedules for tasks off-line and it requires the complete prior knowledge of tasks' characteristics. Although static approaches have low run-time cost, they are inflexible and cannot adapt to a changing environment or to an environment whose behavior is not completely predictable. Several uniprocessor on-line algorithms, such as the Earliest Deadline First algorithm (7) and the Least Laxity algorithm (9) are known to be optimal in the sense that if a set of jobs that can be scheduled such that all jobs complete by their deadlines, then these algorithms will also schedule these sets of jobs to meet all deadlines. However, no on-line scheduling algorithm in multiprocessor systems can be optimal: this was shown for the simplest (identical) multiprocessor model by (5) and the result from (5) can be directly extended to the more general (uniform or unrelated) machine models. (11) explored the use of *resource augmentation* techniques[2] for the on-line scheduling of real-time tasks. They considered two problems in dynamic scheduling: scheduling to meet deadlines in a preemptive identical multiprocessor setting, and scheduling to provide good response time in a number of scheduling environments. Using the resource augmentation approach, they established that several well-known on-line algorithms, that prove poor performance from an absolute worst-case perspective, are optimal for the problems in question when allowed moderately more resources. (4) extended this method to be applied upon uniform parallel machines. However, results derived from their work are applied to periodic task systems.

The idea of this paper is based on competitive analysis theory, introduced by (6), in which the on-line algorithm is allowed more resources than the optimal off-line algorithm to which it is compared. Please note that in addition to changeover time cost considerations, the results of this paper are applied to hard periodic and hard aperiodic task systems.

[2] A method of analysis introduced by (6) for uniprocessor scheduling, comparing the performance of an on-line algorithm to the performance of an optimal off-line algorithm when the on-line algorithm is given extra resources.

3 The On-line Parallel Model

Based on the following assumptions and terminology, the scheduling of hard real-time systems upon a uniform multiprocessor platform comprised of $m \in \mathbb{N}^*$ machines (there is at least one machine) is considered.

- $\pi = \{s_1, s_2, ..., s_m\} \mid m \in \mathbb{N}^*$ represents the m-machine uniform multiprocessor platform in which the machines have *speeds* or *production capacities* $s_1, s_2, ..., s_m$ respectively; without loss of generality, it is assumed that these speeds have positive values and they are indexed in a decreasing manner: $s_j \geq s_{j+1}$ for all $j, 1 \leq j < m$.

- $\tau = \{\tau_i \mid i \in \mathbb{N}\}$ A set of periodic tasks with hard deadlines.

- $J = \{J_j \mid j \in \mathbb{N}\}$ A set of active aperiodic tasks ordered by increasing deadline, J_1 being the task with the shortest absolute deadline.

- Each Job τ_i or J_j is characterized by an arrival time r_i, a production time c_i and a deadline d_i, respectively r_j, c_j, d_j. Whereas the arrival times, production times and deadlines of the periodic jobs τ_i are known in advance, it is assumed that, for the aperiodic job set J_j, these relevant information about the jobs are known when a job arrives.

- The preemptive multiprocessor scheduling model is considered. In the preemptive scheduling model presented in this paper, a job may be interrupted and subsequently resumed *with* a penalty.
  $O = \{O_{j,m} \mid j, m \in \mathbb{N}^*\}$ represents the switch time or changeover time caused by the arrival of the part from type j at the machine m. Changeover time is derived by a static analysis on the machine.

- $u_i = c_i/p_i$ The utilization u_i of a task is the ratio of its execution requirement to its period. Without loss of generality, the tasks in τ and J are indexed according to a decreasing utilization: $u_i \geq u_i + 1$ for all i, $1 \leq i < n$.

- In the context of uniform multiprocessor scheduling, a work-conserving scheduling algorithm is defined to be one that satisfies the following conditions (4): i) no machine is idled while there are active jobs awaiting execution and, ii) if at some instant there are fewer than m (the number of processors in the uniform multiprocessor platform) active jobs awaiting execution then the active jobs are executed upon the fastest machines. More formally, at any instant t and for all $k > j$, if the j'th-slowest processor is idled by the work-conserving scheduling algorithm, then the k'th-slowest processor is also idled at instant t.

- **Job preemption is permitted.** That is, a job executing on a machine may be preempted, prior to completing execution, and its execution may be resumed later. Unfortunately, state-of-the-art real-time multiprocessor scheduling techniques assume that there is no penalty associated with such preemption. It is obvious that disregarding this assumption is inappropriate for manufacturing systems where changeover time overhead may have a considerable time value.
- **Job migration is permitted.** That is, a job that has been preempted on a particular machine may resume execution on the same or different processor. The penalty associated with such migration is unfortunately not accounted for in the literature. Manufacturing system applications necessitate that transport costs of a part or product from one machine to another are regarded.
- **Job parallelism is forbidden.** That is each job may execute at most one processor at any given instant in time.

4 Total Bandwidth Server on Uniform Multiprocessors

Recall that the TBS server technique is used to schedule jointly hard periodic and hard aperiodic tasks under dynamic priority systems upon uniprocessor platforms. One main benefit of this technique is that it guarantees both periodic and aperiodic task sets. An extension of the TBS technique to include changeover time costs is developed in (3). Each aperiodic request receives a deadline

$$\overline{d}_j = \overline{r}_j + \frac{\overline{C}_j + 2\overline{O}_{j,m}}{U_s}$$

where

$$\overline{r}_j = max\ (r_j, \overline{d}_{j-1}, f_{j-1})$$

A TBS algorithm to be implemented upon uniform multiprocessor systems according to the following rules is defined as follows:

- No machine is idled while there is an active job awaiting execution.
- When fewer than m jobs are active, they are acquired to execute upon the fastest machines while the slowest are idled.
- Higher priority jobs are executed on faster processors. More formally, if the j'th-slowest processor is executing job J_g at time t under the TBS implementation, it must be the case that the deadline of J_g is not greater than the deadlines of jobs (if any) executing on the $(j+1)$'th-, $(j+2)$'th-, $(j+3)$'th-, ..., m'th-slowest machines.
- Whenever the j-th aperiodic task arrives at time $t = r_j$, it receives a deadline

$$\overline{d}_j = \overline{r}_j + \frac{\overline{C}_j + \overline{O}_{j,m}}{U_s}$$

where

$$\overline{r}_j = max\ (r_j, \overline{d}_{j-1}, f_{j-1})$$

The utilization of the server U_s will de defined later.
Unavoidably, some additional notations are given in the following.

Definition 1 (W(A,π,I,t)). *Let I denote any set of jobs, and π any uniform multiprocessor platform. For any algorithm A and time instant $t \geq 0$, let W(A,π,I,t) denote the amount of work done by algorithm A on jobs of I over the interval [0,t), while executing on π.*

Definition 2 (S_j). *Let π denote an m-processor uniform multiprocessor platform with processor capacities $s_1, s_2, ..., s_m, s_j \geq s_{j+1}$ for all j, $1 \leq j < m$. S_j is defined as follows:*

$$S_j = \sum_{l=1}^{j} s_l \ for\ all\ j,\ 1 \leq\ j \leq m$$

Definition 3 (λ_π). *(4) Let π denote an m-processor uniform multiprocessor platform with processor capacities $s_1, s_2, ..., s_m, s_j \geq s_{j+1}$ for all j, $1 \leq j < m$. λ_π is defined as follows:*

$$\lambda_\pi = \max_{j=1}^{n} \left\{ \frac{\sum_{k=j+1}^{m} s_k}{s_j} \right\}$$

The parameter λ_π measures the "degree" by which π differs from an identical multiprocessor platform. Consequently λ_π becomes progressively smaller as the speeds of the processor differ from each other by greater amounts.

Lemma 1. (4) *Let π denote an m-processor uniform multiprocessor platform with processor capacities $s_1, s_2, ..., s_m, s_j \geq s_{j+1}$, for all j, $1 \leq j < m$. Let π' denote an m-processor uniform multiprocessor platform with processor capacities $s'_1, s'_2, ..., s'_m, s'_j \geq s'_{j+1}$, for all j, $1 \leq j < m$. Let A denote any m-processor uniform multiprocessor algorithm, and A' any work-conserving m-processor uniform multiprocessor algorithm. If the following condition is satisfied by π and π':*

$$S'_m \geq \lambda_{\pi'} . s_1 + S_m$$

then for any set of jobs I and at any time-instant $t \geq 0$

$$W(A', \pi', I, t) \geq W(A, \pi, I, t)$$

Lemma 1 specifies a condition under which any work-conserving algorithm A' (such as TBS) executing on π' is guaranteed to complete at least as much work as any other algorithm A (including an optimal algorithm) executing on π, when both algorithms are executing on any set of jobs I. This condition expresses the additional computing capacity needed by π' in terms of the $\lambda_{\pi'}$ parameter, and the speed of the fastest processor in π. The smaller the value of $\lambda_{\pi'}$ (the more π' deviates from being an identical multiprocessor), the smaller the amount of this excess processing capacity needed.

The processing of aperiodic tasks can be integrated into a periodic environment by introducing one or more periodic tasks to execute the aperiodic tasks. Therefore, we may deal with aperiodic tasks in a similar way with periodic tasks. As a result, the following theorem (4) uses Lemma 1 to deduce whether a work-conserving algorithm can feasibly schedule a task set: it states that any collection of jobs I that is feasible on a uniform multiprocessor platform π will be scheduled to meet all deadlines by algorithm TBS on any platform π' satisifying the condition of lemma 1.

Theorem 1 *Let I denote an instance of jobs that is feasible on m-processor uniform multiprocessor platform π. Let π' denote another m-processor uniform multiprocessor platform. Let the parameter $\lambda_{\pi'}$ of π' be as defined in Definition 3:*

$$\lambda_{\pi'} = \max_{j=1}^{n} \left\{ \frac{\sum_{k=j+1}^{m} s'_k}{s'_j} \right\}$$

If the condition of Lemma 1 is satisfied by platforms π and π':

$$S'_m \geq \lambda_{\pi'} . s_1 + S_m$$

then I will meet all deadlines when scheduled using TBS algorithm executing on π'.

Thus, Theorem 1 characterizes a uniform multiprocessor platform π' according to its parameter "$\lambda_{\pi'}$" (as defined in Definition 3), and relates the TBS-feasibility of a system, known to be feasible on some platform π, to the cumulative capacities of π and π', the speed of the fastest processor in π, and this parameter $\lambda_{\pi'}$ of platform π'.

As an immediate result of Theorem 1, the results of (11) concerning TBS-scheduling on identical multiprocessor platforms are obtained:

Corollary 1. *TBS is a preemptive, $(2 - \frac{1}{m})$-speed algorithm for hard real-time scheduling on parallel machines.*

Theorem 2 *Given a set of n periodic tasks with machine utilization U_p and a Total Bandwidth server with machine utilization U_s, the whole set is feasibly scheduled if and only if*

$$U_p + U_s \leq S_m$$

where

$$U_p = U_{p_1} + U_{p_2} + ... + U_{p_m}$$

U_{p_1}, U_{p_2}, ..., U_{p_m} are the periodic utilization of the 1st, 2nd, ..., m-th machine respectively , and

$$U_s = U_{s_1} + U_{s_2} + ... + U_{s_m}$$

U_{s_1}, U_{s_2}, ..., U_{s_m} are the total bandwidth utilization of the 1st, 2nd, ..., m-th machine respectively.

5 Schedulability Analysis of Hybrid Task Systems on Uniform Multiprocessors

In this section, the theory developed in Section 4 is applied to study the deadline-based scheduling of hybrid (hard periodic and hard aperiodic tasks) task systems on uniform multiprocessor platforms. Although scheduling hybrid task systems is partly an "on-line" problem in the sense that the periodic task parameters are assumed known beforehand, the results in section 4 concerning hybrid tasks nevertheless turn out to be useful towards developing a framework for scheduling hybrid task systems on uniform multiprocessors.

The method of analysis developed in this section proceeds as follows:

1. an exact test for determining whether a given hybrid task system is feasible on a particular uniform multiprocessor platform is developed and
2. this exact feasibility test along with the results obtained in section 4 are used, to design a schedulability analysis for determining whether a given hybrid task system will be successfully scheduled by TBS on a specified uniform multiprocessor platform.

(4) identified a uniform multiprocessor platform upon which a given periodic task system τ is schedulable. They determine a sufficient condition for τ to be successfully scheduled by EDF on any given multiprocessor platform π' (Theorem 3).

Theorem 3 (4) *Let $\pi' = [s'_1, s'_2, ..., s'_m]$ denote any m-processor multiprocessor platform, and let $\lambda_{\pi'}$ be as defined in Definition 3:*

$$\lambda_{\pi} = \max_{j=1}^{n} \left\{ \frac{\sum_{k=j+1}^{m} s_k}{s_j} \right\}$$

Periodic task system τ will meet all deadlines when scheduled on π' using EDF if the following condition holds:

$$S'_m \geq \lambda_{\pi'} * max\left\{u_1, \frac{U_n}{m}\right\} + U_n$$

Below we show how we can transform the problem of scheduling periodic tasks on uniform multiprocessors to the scheduling of periodic and aperiodic tasks. Whereas EDF is a scheduling policy trying to schedule up to the whole capacity of the multiprocessor platform, the TBS algorithm upon multiprocessor platforms aims at using the whole capacity of the system, while assuring that a fraction of this capacity is dedicated to aperiodic requests. This necessitates the computation of the server utilization of the multiprocessor platform given by Theorem 4.

Theorem 4 *Let $\pi' = [s'_1, s'_2, ..., s'_m]$ denote any m-processor multiprocessor platform, and let $\lambda_{\pi'}$ be as defined in Definition 3:*

$$\lambda'_\pi = \max_{j=1}^{n}\left\{\frac{\sum_{k=j+1}^{m} s_k}{s_j}\right\}$$

The aperiodic task system J has a utilization

$$S'_m = \lambda_{\pi,} * u_1 + U_n + U_s$$
$$U_s = S'_m - \lambda_{\pi,} * u_1 - U_n$$

As a result, deadlines of aperiodic jobs may be computed as defined in the following Corollary.

Corollary 3. *The aperiodic jobs $J(r_j, c_j)$ of a m-processor uniform multiprocessor platform are scheduled using TBS and a total bandwidth as defined in Theorem 4 with a deadline*

$$d_j = max(r_j, d_{j-1}) + \frac{c_j}{U_s}$$

Furthermore, the Corollary 4 follows directly from the results of (3) involving the changeover time costs in the TBS algorithm and allowing resource reclaiming.

Corollary 4 *Aperiodic jobs $J(r_j, c_j)$ of a m-processor uniform multiprocessor platform are scheduled using TBS and a total bandwidth as defined in Theorem 4 with a deadline*

$$\overline{d}_j = \overline{r}_j + \frac{\overline{C}_j + 2\overline{O}_{j,m}}{U_s}$$

where

$$\overline{r}_j = max\left(r_j, \overline{d}_{j-1}, f_{j-1}\right)$$

Theorem 4 is now illustrated by an example.

Example. *Consider a task system τ comprised of five periodic tasks (c_i, p_i)*

$$\tau = \{(15, 10), (4, 5), (14, 20), (6, 15), (2, 10)\}$$

and an aperiodic task (r_i, c_i)

$$J = \{(5, 7)\}$$

for this system, $u_1 = 1.5$, $u_2 = 0.8$, $u_3 = 0.7$, $u_4 = 0.4$, $u_5 = 0.2$. Suppose that τ and J are to be TBS-scheduled upon the uniform multiprocessor platform $\pi' = [3, 1, 0.5]$ - will all deadlines be met?

By Definition 3, the value of $\lambda_{\pi'}$ for the uniform multiprocessor platform π' is

$$\lambda_{\pi'} = max\left\{\frac{1 + 0.5}{3}, \frac{0.5}{1}\right\} = \frac{1}{2}$$

and the total computing capacity is

$$3 + 1 + 0.5 = 4.5$$

The total computing capacity needed for τ is

$$1.5 + 0.8 + 0.7 + 0.4 + 0.2 = 3.6$$

and with the fastest processor having a computing capacity

$$s_1 = 1.5$$

By theorem 4, the aperiodic requests J are feasible on the 3-processor uniform multiprocessor platform with a total bandwidth of

$$4.5 = 0.5 * 1.5 + 3.6 + U_s$$
$$4.5 = 4.35 + U_s$$
$$U_s = 0.15$$

The aperiodic job $J = \{(5, 3)\}$ is to be scheduled on the 3-processor multiprocessor platform with a deadline equal to:

$$d_j = max\{5, 0\} + \frac{3}{0.15} = 25$$

and τ and J can consequently be scheduled by TBS to meet all deadlines on π' with $U_s = 0.15$.

6 Conclusions

One of the most important properties, that differentiates real-time systems from other conventional systems, is predictability. The system must be able to predict the consequences of any scheduling decisison. If some task cannot

be guaranteed within its timing constraints, the system must notify this fact in advance, so that alternative actions can be planned in time to cope with the event. This paper deals with the problem of predictable aperiodic scheduling in manufacturing systems underlying real-time constraints. A predictable scheduling, based on the TBS algorithm, is adapted upon a multiprocessor production shift. It is extended to include the novel feature of accounting for changeover time overheads. Schedulability analysis techniques are presented, in order to assure the predictability of the system.

References

[1] T.F. Abdelzaher and K.G. Shin. Period-Based Load Partitioning and Assignment for Large Real-Time Applications. *IEEE Transactions on Computers*, 49(1):81–87, January 2000.

[2] H. Aydin, R. Melhem, S. Mossé, and P. Mejia-Alvarez. Optimal Reward-Based Scheduling of Periodic Real-Time Tasks. In *Proc. of the Real-Time Systems Symposium*, Phoenix, Arizona, December 1999. IEEE Computer Society Press.

[3] D. A. El-Kebbe. Aperiodic Scheduling in a Dynamic Real-Time Manufacturing System. In *Proc. of the IEEE/EE Real-Time Embedded Systems Workshop (Satellite of the IEEE Real-Time Systems Symposium)*, London, December 2001.

[4] S. Funk, J. Goossens, and S. Baruah. On-line Scheduling on Uniform Multiprocessors. In *Proceedings of the Real-Time Systems Symposium*, pages 183–192, London, December 2001.

[5] K. Hong and J. Leung. On-line Scheduling of real-time tasks. In *Proc. of the Real-Time Systems Symposium*, pages 244–250, Huntsville, Alabama, December 1988. IEEE Computer Society Press.

[6] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. In *36th Annual Symposium on Foundations of Computer Science*, pages 214–223, Los Alamitos, Oktober 1995. IEEE Computer Society Press.

[7] C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *J. of the ACM*, 20(1):46–61, January 1973.

[8] M. Moir and S. Ramamurthy. Pfair Scheduling of Fixed and Migrating Periodic Tasks on Multiple Resources. In *Proc. of the Real-Time Systems Symposium*, Phoenix, Arizona, December 1999. IEEE Computer Society Press.

[9] A.K. Mok. *Fundamental Design Problems of Distributed Systems for the Hard Real-Time Environment*. PhD thesis, Massachusetts Institute of Technology, 1983.

[10] Tatsushi Nishi, Akihiro Sakata, Shinji Hasebe, and Iori Hashimoto. Autonomous Decentralized Scheduling System for Just-in-Time Production. In *Proc. of the 7th International Symposium on Process System Engineering*, pages 345–351, 2000.

[11] C. A. Philips, C. Stein, E. Torng, and J. Wein. Optimal Time-Critical Scheduling via Resource Augmentation. In *Proc. of the 29th Annual ACM Symposium on Theory of Computing*, pages 140–149, El Paso, Texas, May 1997.

[12] U. Schneider. *Ein formales Modell und eine Klassifikation für die Fertigungssteuerung*. PhD thesis, Heinz-Nixdorf Institut / Universität Paderborn, 1996.

CLEaR: A Framework for Balancing Deliberative and Reactive Control

Forest Fisher, Daniel M. Gaines, Tara Estlin, Steve Schaffer, Caroline Chouinard

Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Dr.
Pasadena, CA 91109
{firstname.lastname}@jpl.nasa.gov

Abstract

A major challenge in developing robotic applications for real-world problems is that many domains include tight resource and temporal constraints coupled with uncertainty in how much resource and time will be required to perform a task. We have developed the CLEaR framework to address this challenge. CLEaR unifies the planning and execution processes to increase the responsiveness of a robotic agent operating in these types of environments. This unified approach is realized by extending the traditional three-tier robotic control architecture with an Execution-Time Plan Manager, an Atomic Resource Manager (ARM) and an Execution-Time Query (ETQ) capability. Through the interaction of these components, CLEaR is able to (1) reduce the need to replan, (2) detect the need to replan earlier, and (3) replan before entering a failed state.

Introduction

Robotic agents performing under hard resource and time constraints in uncertain environments require careful balancing of both deliberative and reactive reasoning [Knight, et al, 2001]. As in most domains with uncertainty, a task may fail or produce unexpected results leading to plan failures. If the robot is also under hard time deadlines and resource constraints, a task requiring a different time or resource allocation than planned could cause failure at future points in the plan. In some cases, the robot may be able to retry a failed task, use more time or take up more of a resource without causing a problem.

Consider for example, a Mars exploration rover that must pick up a rock. If it fails on its first attempt, it may want to try again. However, doing so could lead to other problems at later stages of the plan. If the rover spends too much time trying to complete this task, it may miss another deadline, such as taking an image while the sun is in a particular position in the sky. Or, it may use up too much of some resource, such as energy, resulting in the inability to perform other critical tasks. The challenge is to determine whether or not a change in time or resource usage will cause a problem so that the rover can take appropriate action, and to identify and fix the conflicts in the plan without preventing the rover from meeting other deadlines. In this example, a deliberator is used to project current resource and time usage into the future, detect problems and make repairs. An executive uses more reactive reasoning to deal with unexpected events and perform low-level control. The rover needs both of these capabilities to successfully operate in this environment.

Most of the robotic applications in the literature have not been confined by hard resource constraints and strict time deadlines; consequently little work has been done in this area. However, there has been a growing awareness of these issues in recent years. At NASA, almost all the robotic space exploration domains involve uncertain environments with deadlines and tight resource constraints.

In pursuit of developing high-level control software capable of addressing these issues, we have developed the CLEaR (Closed Loop Execution and Recovery) control software/framework. CLEaR provides a unified framework for performing planning, scheduling and execution by balancing both deliberative and reactive reasoning. In most related approaches to robotic control, the planning and execution components are treated as black box functions that do not interact in real-time. Our approach differs in that both the planning and execution functionalities share the responsibility for decision-making and resource management.

In our system the unified planning and execution responsibilities are realized through three means of increased interaction and information sharing between the deliberative and reactive functions:

1. The executive provides soft-real-time state, resource and time updates enabling the deliberator to anticipate problems and replan if necessary.
2. The deliberator provides rapid response to queries about time and resource usage variations, thus enabling the executive to manage a task that is behaving unexpectedly.
3. The executive uses execution time resource knowledge combined with projected usages while managing tasks.

By enabling the long-term deliberation and the short-term reactive execution functionalities to share information on a more frequent basis, the system can: (1) reduce the need to replan, (2) detect the need to replan earlier and (3) replan while continuing to execute valid portions of the plan without entering a failed state. In other words, the system can circumvent as many failure situations as possible without impacting plan execution. By achieving these capabilities we are able to produce a robotic agent control system capable of goal-based commanding in an uncertain environment while adhering to hard resource and time constraints.

Our framework for balancing deliberation and reaction has been motivated by several NASA space exploration

domains. The most significant influence has been Mars surface exploration with autonomous rovers, especially the proposed Mars Smart Lander mission. In the next section we will describe this mission and illustrate how the mission provides challenging time and resource constraints for an autonomous robot. We will describe how we have designed CLEaR to deal with these types of challenges and then present a case study illustrating how CLEaR will enable a rover to successfully deal with these challenges.

2009 Mars Smart Lander Rover scenario

In 1997, JPL successfully completed the first mission to explore Mars' surface with a mobile robotic platform (Sojourner rover). During the mission, human ground teams performed nearly all deliberative decision-making including the determination of resource bounds. While the mission was a landmark in space exploration and provided valuable science data, it required intensive human interaction and explored a very small region of terrain.

In 2009, JPL plans to send another mobile robotic platform to Mars to perform numerous geological surface experiments. This mission is currently called the Mars Smart Lander mission and represents a significant increase in scale with respect to mission duration, science return and terrain covered. Figure 1 provides an overview of the mission. The mission objectives are to explore the landing site and make long-range traverses to two additional geological science locations where the robot will perform more science data gathering. The rover will have limited resources, such as power and RAM, to complete these goals. It will also be under tight time constraints in order to complete the ambitious objectives and meet mission requirements, such as ground communication windows.

There will be communication with Earth at the beginning and the end of each Martian day. In the morning session, the goals for the day are uploaded to the rover and additional data will be down-linked. In the evening, the day's data is down-linked. This data includes panoramic images used in selecting future goals.

This scenario has two modes of operation.. The first being the geological science location operations, and the second being the long-range traverses between those locations. During the first mode, the role of high-level autonomy software will primarily involve resource management (mainly power, memory and time) and robust execution.

During the second mode, the rover is expected to make long-range traverses averaging 600m/day. This distance is well beyond the "line of sight" of the ground operations team based on images down-linked from the previous day. Therefore the traverse will require significant onboard autonomy. Further motivating the need for high-level goal-based autonomy is that the rover should perform as much opportunistic traverse science as possible without impacting the progress of the 3km long-range traverses.

Unified Planning and Execution Framework

Current practice for rover operations, as used on the Pathfinder mission [Mishkin, et al, 1998] and planned for the upcoming 2003 Mars Exploration Rover (MER) mission, is to perform nearly all decision making remotely

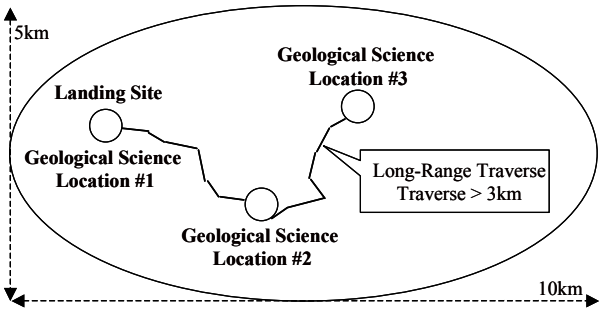


Figure 1: Mars Smart Lander Scenario

from earth. When the rover encounters a situation that deviates from its uploaded command sequence, the fault protection software will attempt to resolve the problem. Failing that, the rover enters *safe-mode* and must: wait for a communications opportunity, transmit the state of the rover and imagery of the environment back to Earth, and wait for a new command sequence. Depending on when the next communication window is scheduled, this can waste considerable time. Further, to date these rovers have been solar powered and can only perform major functions for a few hours per day (typically 4-6 hours). Placing the rover in *safe-mode* can easily cause the loss of a full day of operations. Because the mission cannot be extended, falling behind schedule due to execution failures results in reduced science return.

While this style of operations reduces development cost and simplifies testing of flight control software, it adds to the time and cost of mission operations. This, in turn, severely limits the rover's in-situ capabilities.

From an automation standpoint part of what limits rover operations performance is that the decision-making process has traditionally been separated from the execution process. To address this several systems have colocated the deliberative-planning and execution capabilities, to dramatically increase the rover's responsiveness and reduce the need for the rover to be put into *safe-mode*.

Most of these systems can be classified as three-tiered control architectures [Gat 1998]. Under a three-tiered system the deliberative planning and reactive execution components are colocated but tend to function independently typically in a black-box integration. These architectures get their name from a stack-like partitioning of the system into three functional components. The top tier provides deliberative function, the middle tier performs reactive execution, and the bottom interfaces to the hardware controllers. Generally, the higher up in the stack, the greater the level of abstraction at which the components functions and the longer it takes to perform. The top tier is usually reserved for search algorithms. In the event of an execution failure, when compared to Earth-based deliberation, this approach can reduce the time the rover waits for ground intervention by facilitating replanning onboard.

While some systems will plan for future phases of a mission during the execution of the current phase. One drawback of many traditional three-tiered[1] approaches is

[1] Not all three-tiered architectures are limited by Sense-Plan-Act(SPA), for instance ATLANTIS [Gat 1992] plans and executes asynchronously.

that they do not instigate replanning prior to an execution failure of the mission phase currently being executed. In order to replan and thus preempt execution failures, it is necessary to provide the deliberator with frequent state, resource and temporal updates. These can then be propagated through the plan to predict future inconsistencies. If the deliberator is able to incrementally resolve these conflicts[2] while executing valid portions of the plan, then the robotic agent will be more responsive to unexpected events. We refer to this capability as continuous planning.

In our implementation of this framework, we use CASPER (Continuous Activity Scheduling Planning Execution and Replanning) as the continuous planner [Chien, et al, 2000a, 2000b]. CASPER provides the Deliberator and Execution-Time Plan Manager components depicted in Figure 2. The Executive component is provided by TDL (Task Description Language), a robust task level execution framework [Simmons, Apfelbaum 1998].

The CLEaR framework is distinct from other three-tier architectures because it provides increased interaction and information sharing between the executive and the deliberator [Gat 1998, 1992; Bonasso, et al, 1997]. This is partly realized by the use of a continuous planner combined with frequent updates from the executive to the deliberator.

Two other areas of increased interaction and information sharing are provided by the executive's ability to: (1) make decisions on how to execute a task by querying the deliberator to determine if a given execution will cause a plan failure and (2) consider execution time resource knowledge in deciding on task expansions. These last two capabilities are provided by the Execution Time Query (ETQ) manager and the Atomic Resource Manager (ARM), also depicted in Figure 2. In the following two sub-sections, we describe these components.

ARM: Atomic Resource Management

Motivations and Design Goals

There are certain types of activities that require a resource intermittently during their execution. For example, while a rover is navigating, it will occasionally take images to detect and avoid obstacles in its path. Although navigation requires the camera, it does not use it continually. In fact, after each image, it can make a rough estimate of when it needs the camera again. This can be done because, given an image, the navigation activity determines how far it can safely travel before taking another image. As a result, the camera will become available at different times throughout the navigation, and it would be nice if other activities could take advantage of this.

In fact, within the context of the Mars Smart Lander mission, there is a need for such a capability to enable opportunistic traverse science during a long-range traverse. Traverse science uses the camera to take images at different times during the traverse to look for items of interest. Like navigation, traverse science does not need the camera continually throughout the traverse and could use the camera when not in use by navigation.

[2] Our incremental conflict resolution is performed by an iterative repair algorithm [Zweben, et al 1994; Minton, Johnston 1998].

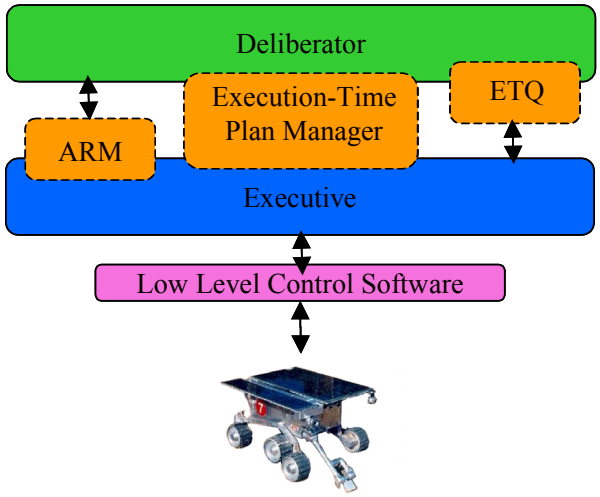


Figure 2: CLEaR Framework Diagram

In general, we may have several activities that each make intermittent use of a particular resource. If we knew ahead of time when each requires the resource and for how long we could use deliberative scheduling techniques to create a plan to avoid resource conflicts while executing these activities. Unfortunately, for some activities, such as navigation, we cannot accurately predict when the resource will be needed. Furthermore, the accuracy of our prediction will decrease as we attempt to predict uses further in the future. As a result, an activity may use the resource earlier or later than expected, and once the activity has the resource, it may require it for a duration different from what it had originally anticipated.

Given these conditions, scheduling such activities is challenging. Previous approaches for dealing with these issues include the following. First, a planner could avoid the problem by refusing to schedule activities concurrently if they require the same resource, regardless of whether or not they only require the resource intermittently. The downside to this is that you are limiting the robot's capabilities, and in some applications concurrent activities are required to complete a goal. A second approach would be to form a deliberative schedule for these activities based on rough estimates on the frequency and duration that each activity will use the resource. The disadvantage here is that it is very unlikely that during execution the activities will use the resources as predicted. This could be handled by performing rescheduling within the planner as it gets new updates on actual resource usage or by allowing the executive to preempt lower priority tasks whenever there is contention for the resource. The former approach is likely to result in thrashing within the plan as information changes. Both approaches are likely to lead to a large number of preempted activities. A third approach is to create special executive task managers for each combination of activities that may need to run in parallel. Each such manager would be designed to arbitrate resource usage among these particular activities.

Instead, we have chosen to deal with these challenges by developing a resource manager for use within the executive. Because the executive needs to be responsive to unexpected changes, our primary design goal is to keep the resource manager fast so that it can quickly respond to requests.

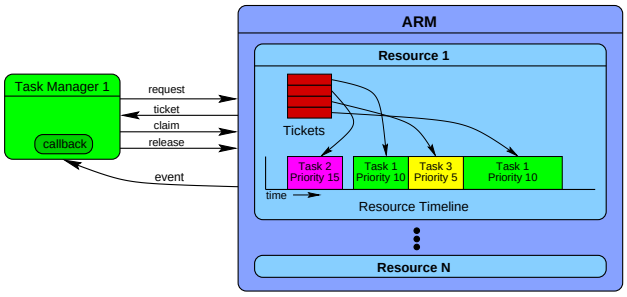


Figure 3: Design of ARM

Therefore, we will favor simpler designs and algorithms to reduce computational complexity.

Because the predictions on when and how long a resource will be used is uncertain, the resource manager must be able to quickly react so that high priority activities can have access to the resource when it is needed. However, the resource manager should make use of predicted information when available to try and reduce the number of times it must preempt another activity. Thus, our secondary goal is to balance the use of deliberation and reaction, where deliberation takes advantage of predicted information and reaction to deal with unexpected changes in resource usage.

Design of ARM

For our first implementation of ARM, we decided to address only atomic resources. An atomic resource can be used by at most one task at a time and is either available or not available. For example, a camera can be used by a single activity and, therefore, is considered an atomic resource. In contrast, aggregate resources, such as solar array power, can be used by several tasks at a time and each task can use a different amount of solar power. This makes it more difficult to represent and search for reservations. As this reasoning is needed, we rely on CASPER's deliberative planning and scheduling capabilities to perform reasoning about aggregate resource usage. These decisions are generally based on near worst-case estimates of usages. In the next sub-section we describe a method for enabling the executive to make reactive decisions about aggregate resources with the assistance of the deliberator.

Figure 3 shows the design of ARM. For each resource, ARM maintains a timeline that keeps track of when the resource is in use, along with the task and the priority of the task that is using it. For each reservation on this timeline, ARM keeps a ticket, which can be used by tasks to access their reservations.

Before a task can use a resource it must first make a request to ARM indicating its priority, the time interval within which it would like to start using the resource and the duration that the resource will be used. If ARM can find room, it will place a reservation on the timeline and return a ticket to the requesting task. Before using the resource the task must hold a valid ticket and claim the resource. When the task is finished with the resource or otherwise no longer needs the reservation, it sends a release to ARM, which will clear out the reservation.

Although this is the nominal behavior of the system, it is unlikely that things will go so smoothly during execution. Therefore, ARM is designed to deal with unexpected situations. Unexpected events include: a task requiring a resource sooner or later than it anticipated, a task using a

```
Request (priority, startTime, endTime, duration)

T = resource timeline
T' = working copy of T between startTime
     and (endTime + duration)
p = -INFINITY
While p < priority
  Remove from T' all reservations with priority p
  i = earliest free interval in T' with size >=
duration
  If i exists:
    Discard from T any reservations during interval i
    Create new reservation for interval i
    Return i
  p = lowest priority in T'
Return failure
```

Figure 4: ARM Reservation Request Algorithm

resource for a shorter or longer duration than it expected and a task making a reservation during a time interval in which another task already has a reservation. All of these cases are handled by ARM. The following subsections provide more detail on how these issues are resolved. In general, our approach is to associate a task priority with each reservation. Whenever there is a conflict for a resource, the task with the higher priority wins. If the tasks have equal priority, advantage is given to the task that came earlier.

Requesting a Ticket

In keeping with our goal of avoiding preemption due to resource conflicts, the resource manager will do some amount of look ahead when processing requests from tasks. Look ahead is facilitated by requiring each task to request a ticket before using a resource. A ticket represents a promise between ARM and a task. When a task requests a ticket, it informs ARM of the time in which it would like to start using the resource and the duration of that usage. If ARM can find a slot for the request, it will issue a ticket, giving the task the right to claim the resource during the specified interval.

However, given uncertainty during execution, the manager cannot strictly follow these reservations and must accommodate deviations in the actual timing requirements of the activities. The resource manager will have to modify the plan, which may involve dropping lower priority reservations or preempting the current resource holder. The resource manager will attempt to give notice to the affected activities so that they can take appropriate action.

Figure 4 shows the algorithm used for making reservations. When an activity makes a request, it provides its priority, the time interval in which it would like to begin using the resource and the duration indicating how long it intends to use the resource. Note that the duration is independent of the time interval in which it would like to start using the resource. ARM first tries to find an existing slot during the requested time interval. If none are found, it will begin removing lower priority reservations until enough space is freed or until all the remaining reservations have equal or higher priority than the requesting task.

The algorithm runs in time $O(n^2)$ where n is the number of currently open reservations (i.e. reservations not in the past). At each priority level, the algorithm must search through the reservations at that level and higher looking for open space. In the worst case, each reservation is at a different level, requiring n iterations through the loop

The reservation algorithm reveals tradeoffs that were made when designing ARM. Our objective was to provide fast response to the requesting task without disturbing existing reservations. The quickest algorithm would be to first remove all reservations with a lower priority than the requesting task and then find a free space. While fast, this could also result in the unnecessary removal of reservations. The algorithm we are using is more computationally complex. It iterates through the reservation priority levels in an attempt to remove lower priority reservations first. Although of higher complexity, this algorithm better enforces graduated priority levels and only iterates a few times in practice.

We recognize that many further enhancements could be made to the scheduling algorithm such as: instead of removing reservations when there is no room for a new request, it might be possible to relocate them. Alternatively a single higher priority reservation may be removed to preserve numerous lower priority reservations. However, we did not incorporate these techniques because (a) they would have involved computationally expensive search and (b) the benefit would be reduced if a task did not perform as predicted, thus forcing repeated changes to the plan.

Our approach does not consider multiple reservations simultaneously. Some tasks may require the use of several resources at the same time, requiring concurrent free intervals to be found for each resource. More complicated situations could arise if tasks require resources at temporal offset from each other. This type of scheduling is dealt by our deliberator and not ARM.

Claiming a Resource

A task may claim the resource at any time during its reservation. This addresses the uncertainty a task may have about when it needs the resource. If it is late, it can still claim the resource. However, if it needs the resource earlier, it must request a new reservation.

If another task is still holding the resource (after its reservation period), then ARM checks the priority of the tasks. The higher priority task always wins, and ties are broken in favor of the current resource owner. This avoids a possible preemption.

Releasing a Ticket

A task can release a resource at any time, providing extra free space on the timeline for new requests. However, if the task requires the resource for longer than allotted, it may keep it until a higher priority task makes a claim.

ETQ: Execution-Time Query

Even with execution-time atomic resource management, situations will arise where a task requires a different amount of a resource or time than was scheduled. For example, adverse soil conditions may make it more difficult for a rover to dig, thus using more energy and time to complete the task. To enable reactive reasoning about aggregate resources and time, we have developed an Execution–Time Query (ETQ) mechanism to enable the executive to safely deviate from the constraints laid out in the plan by the deliberator.

One approach to dealing with this problem is to allow the task to continue operation and use more of the resource. As resource and time updates are made, the deliberator will detect problems that this extra resource use will have on the plan. For some types of resources, this approach will be fine. If an imaging task uses extra RAM, and the scheduling functionality detects that this will cause a problem, it can decide to discard some of the collected data.

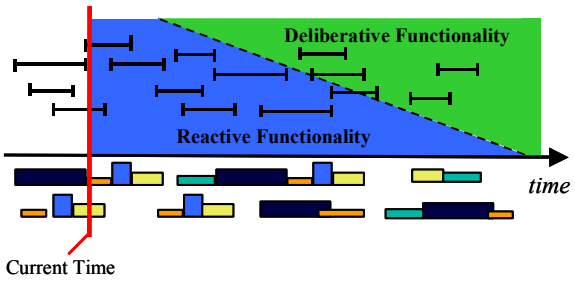


Figure 5: CLEaR Concept Diagram

Unfortunately, other types of resources cannot be so easily replenished, and this approach could lead to catastrophic failures. If the rover uses extra energy, the scheduling functionality detects a problem too late and the energy is already gone. This could prevent the rover from completing a mission critical task such as communicating with Earth. *Sometimes it is better to ask for permission than it is to ask for forgiveness*.

To deal with this challenge, our framework supports a query system that enables the executive to ask for permission before exceeding a resource limitation. This capability provides global consideration of resource and time usage during execution. When a monitor detects that the resource will be over-subscribed, instead of just completing or failing the task, it can query the deliberator. The executing task queries the deliberator indicating how much more of the resource or time the task would like to use. The deliberator then does a quick check to determine if the new resource usage would cause any conflicts, by placing this new expected use into the plan and propagating it forward. This is done similarly to how execution updates are handled. If no conflicts result, then the query request, *permission* is granted and the task can continue to execute within the confines of this new resource or time constraint/restriction. If this query propagation creates any conflicts then the projected update is backed out of the plan and the request is rejected. As we will see in the scenario, there are situations when exceeding the resource allotment is the desired behavior. For that reason, the framework does not require that execution-time resource query be used. Instead it is left to the knowledge engineer to decide which tasks should “ask for permission” or “ask for forgiveness”.

Similar to the design of the execution-time resource management functionality, there are alternative designs for this query capability that would provide more functionality at higher computationally expense. The deliberator could check if the conflicts resulting from a changed resource usage could be adequately repaired, and if so, give permission to the task.

Current Status

In our current framework CASPER creates abstract command sequences and executes those sequences by translating the CASPER planning activities into TDL task-tree goal nodes, which are then further expanded by TDL. In Figure 5 we graphically depict levels of responsibility between deliberative and reactive decision-making as a

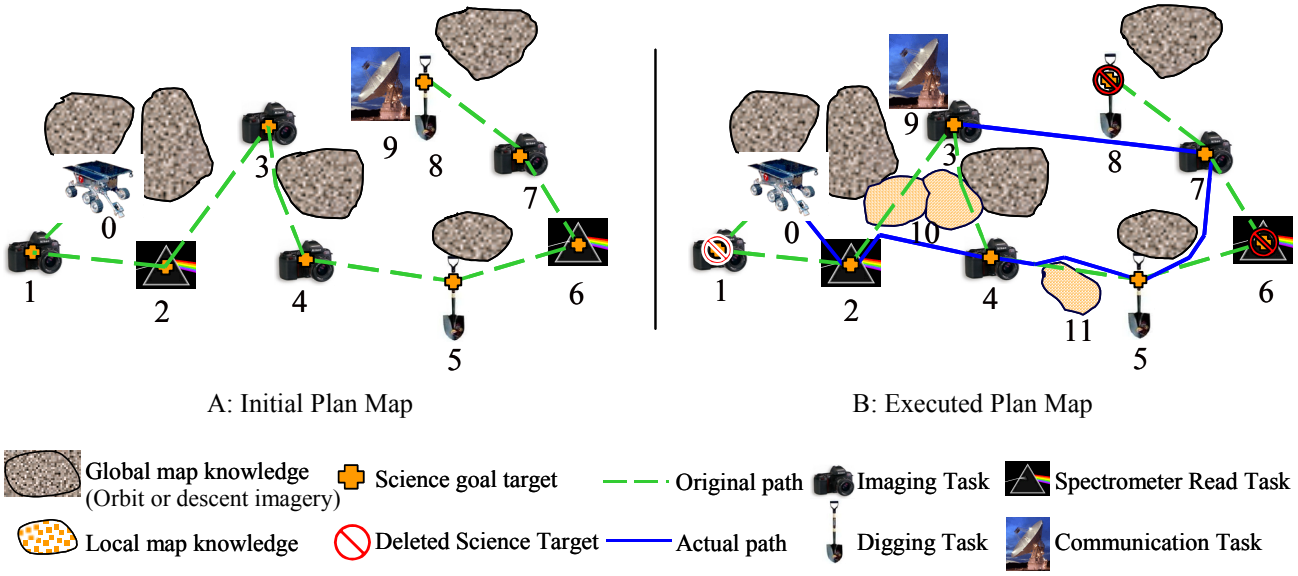


Figure 6: Scenario Maps for a Geological Science Location

function of time. At the *current time*, all decision-making with respect to the executing tasks are performed reactively. As the plan is projected forward, the deliberator takes on an increasing role in decision-making.

By enabling the long-term deliberator and the short-term reactive executive to share information on a more frequent basis, the system can: (1) reduce the need to replan, (2) detect the need to replan earlier and (3) when necessary replan before entering a failed state while continuing to execute other valid portions of the plan.

Scenario Examples:

We are continuing to develop both our concept of unified planning and execution along with the implementation of that concept within the CLEaR system. To assist in this process, we are developing rover mission scenarios consistent with the proposed Mars Smart Lander mission, described in the *Mars Smart Lander Rover scenario* section, for use in testing and validating our system. We are performing tests in simulation and on the Rocky7 and Rocky8 research rovers in the JPL Mars Yard.

Figure 1 provides a high level view of the complete scenario, which includes two long-range traverses to three geological science locations and several science data gathering goals at those locations. Figure 6 contains a blow-up view of one potential geological science location. The ground operations team provides the rover with eight science targets within this site. These targets consist of: four images, two spectrometer readings, and two digs each at different locations. The ground team assigns a priority to each target, which is used in the science return optimization algorithm of the deliberator and ARM [Rabideau, et al, 2000].

Our description of the scenario will begin with events that occur while the rover is completing tasks at the geological science location. We begin with this portion of the scenario as these techniques are a logical extension to those of previous work in integrated planning and execution [Gat 1998, 1992; Bonasso, et al, 1997]. We will then move on to events that occur during the long-range traverse between science locations. During this section we will describe how these new capabilities, namely ARM and ETQ, increase the rover's ability to deal uncertain events.

We decided to turn the execution-time query facility off while the rover was in the geological science site. This was done because each of the goals in this part of the scenario is part of the rover's primary mission. If it requires extra resources to complete a task, it should do so, and the planner will have to repair the plan as best it can to achieve future goals. During the long-range traverse, in the second part of the scenario, we will use execution-time resource queries to prevent opportunistic science from interfering with the rover's primary goals.

Part 1: Within the Geological Science Location

The system begins by employing a generic Traveling Salesman Problem solver to identify an initial sequence (tour) for visiting each of the science targets. The sequence is then expanded to include all of the planner level activities required to carry out that tour. During the generation of the command sequence, all of the resource constraints are maintained. For our current scenario this means that the rover's energy and memory resource profiles must be maintained within the operations constraints. For energy this requires that the projected and actual used energy level must not drop below the prescribed margin levels. In part this is to ensure that there is enough energy available for the communications activities at the end of each day and also to ensure that there is enough energy stored in the batteries for overnight operations. For memory the system must balance the memory buffer capacity to maximize science return and ensure the availability of memory storage space for future higher priority science observations.

The dashed line in Figure 6-A indicates the initial planned sequence that the rover will take to visit the science targets. However, things will not go as planned during execution and the plan will have to be modified, as shown by the solid line in Figure 6-B. The following section highlights some of the unexpected events that occurred during execution and

the challenges these events posed when coupled with the time and resource constraints imposed by the mission.

Deliberator

The first problem with the plan is detected before execution begins. The rover has been asked to collect more science data than it has room to store in memory. The deliberative scheduling functionality is able to detect this problem and discards low priority science targets until enough space is available for the remaining targets. In the example, image target 1 from Figure 6-A is thrown out, and a new path for visiting the remaining targets is generated.

Execution-Time Plan Manager

During execution, other resource usage issues arise. One of the challenges in execution monitoring for a system under time and resource constraints is that it is not enough to detect whether or not an action resulted in success. One must also monitor how the activity affected the rover's resources and how much time it took. For example, in Figure 6-B, the image task at target 4 and the dig at target 5 were successful in that the main objective of the task was completed. However, they also resulted in the use of more resource than was anticipated. The image task required an excessive amount of memory and the dig used up too much energy.

The Execution-Time Plan Manager (Figure 2) enables the rover to deal with these problems. The Executive continuously provides updates on the state of each resource. After each task is completed, the continuous planner notices that there will be a deficiency in one or more resources. For example, after the image is taken, the system realizes that there will be insufficient memory to complete the other science goals. In each case, the deliberator looks for low priority tasks to drop, just as it did during initial plan generation.

Similar behavior occurs when a task requires an unexpected amount of time. Like the resource constraints, tight time constraints require that the rover keep track of how much time a task is taking so that it can avoid missing future deadlines. For example, as the rover moves from target 2 to target 3, its obstacle detection behavior must avoid unexpected rocks that did not show up in the initial map the rover was given. If the rover spends too much time trying to reach this target, it may miss other deadlines, such as the communication opportunity with Earth.

Again, the continuous scheduling functionality of our framework addresses this challenge. Just as each task includes monitors on resource usage, some tasks also include monitors to track the rover's progress over time. In this example, the monitor realizes that, given the rover's position, it will not be able to complete the task in the allotted time. At this point the continuous scheduling functionality takes into account the latest information about obstacles in the area and modifies the plan accordingly. As in the previous cases, it might be necessary to drop certain tasks to make up time. However, in this case, it turns out that the rover can visit the targets in a different sequence and still have enough time to make the communication deadline.

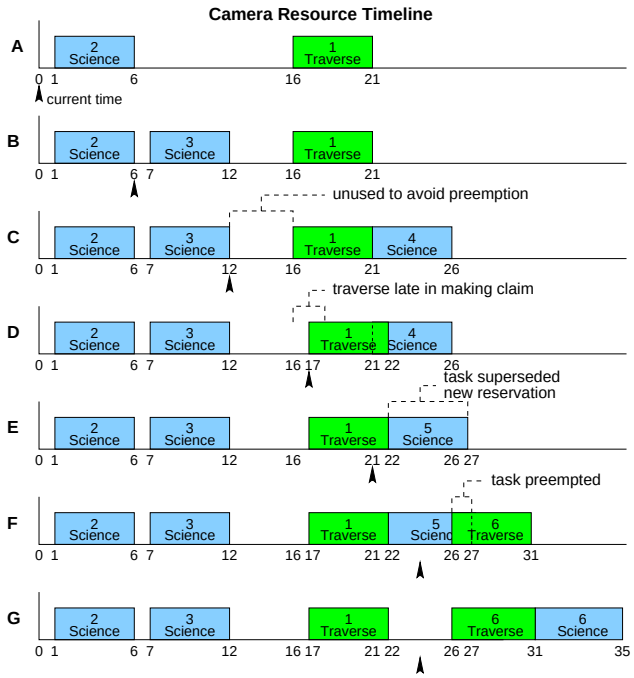


Figure 7: Example of ARM in Action

Part 2: Long-range Traverse Between Geological Science Locations

After the rover completes the tasks in Figure 6, it must proceed to the next geological science location in Figure 1. This portion of the scenario will highlight benefits of performing execution-time resource management to schedule concurrent activities that make intermittent use of the same resource. We will also show how the execution-time query facility can be used to prevent a task from interfering with a plan when it requires more of a resource.

ARM: Atomic Resource Manager

The benefits of execution-time resource management are highlighted during long-range traverses. Recall from the Mars Smart Lander (MSL) reference mission that we would like to perform opportunistic science during these traverses. Although both the traverse and traverse science tasks require the use of the camera, neither requires it continuously. These tasks can be scheduled concurrently. Due to uncertainty in execution, however, it is difficult to predict when and for how long each task will require use of the camera.

To test our execution-time resource scheduling capability, we created a simulation to model the camera usage behavior of Gestalt, the navigation software that will be used on the 2003 Mars Exploration Rover (MER) mission. Whenever Gestalt takes an image, it determines how far the rover can safely travel before it must take the next image. With an estimate of the rover's velocity, we make a prediction of when the traverse will require the camera again. A corresponding request is made to the resource manager for the interval that the camera will be needed.

Meanwhile, our simulation of opportunistic science tries to take images as often as it can. Before using the camera, it must first make a request of ARM specifying how long it will need the camera. As stated in the MSL reference mission description, opportunistic science should not

interfere with other rover activities; thus, we give opportunistic science a lower priority than traverse tasks.

Figure 7 illustrates the events that occur during a typical run. The figure depicts the reservations that are placed on the camera resource timeline during the execution of the traverse and opportunistic science tasks. Each reservation is numbered to indicate the order in which it was placed on the timeline. The time units in the x-axis are in seconds and mark the times for the various reservations. The upward arrow denotes the current point in time.

At the start of the scenario, Figure 7 (A), the traverse task has made a reservation that will begin at time 16 and last for 5 seconds, until time 21. Next, opportunistic science makes a request for the camera for sometime between time point 1 and time point 31. The executive resource manager finds space for the reservation starting at time 1. As time elapses, the science task completes its first use of the camera, places another request and uses the camera for a second time. At time point 12 in (C), opportunistic science requests the camera, however it cannot be given the earliest slot because its duration would conflict with traverse's reservation. Therefore, it is given a reservation that begins at time 21. In the absence of execution-time resource management, opportunistic science would take the earlier slot and later be preempted by traverse. This step demonstrates how ARM protects against preemption. Instead, opportunistic science is scheduled at a time when the resource is predicted to be free.

In (D) things do not go as planned: traverse has taken longer than expected to claim the resource. The behavior of the system at this point depends on how traverse interacts with the resource manager. If traverse releases its current reservation and makes a new one to start immediately, the resource manager will notify the following science task that it has been superseded.

However, in the example scenario, the traverse task does not release the resource and instead claims it and then holds on to it past its scheduled reservation. At time point 21, opportunistic science attempts to claim the resource but is denied in favor of the higher priority traverse. Science then makes a new request and is given a reservation starting at time slot 22. Here, no preemption was necessary to resolve the conflict, as the science task was not started.

There will be cases when preemption cannot be avoided. An example occurs in (F) when the science task has been initiated, but then the traverse requires the use of the camera beginning at time 26. Because traverse has higher priority, the resource manager gives it a reservation and preempts the opportunistic science task. (G) Shows the final state of the timeline after opportunistic science has been given a new reservation.

ETQ: Execution-Time Query

As stated earlier, opportunistic science should never interfere with other rover activities. Therefore, for the traverse portion of the scenario, we employ the execution-time query (ETQ) capability to enable the task manager for opportunistic science to ask permission before using more of a resource than it was prescribed by the plan.

In our scenario, the deliberator allocates a certain amount of memory for use by opportunistic science based on a rough estimate of how many images it will take and how much RAM the images will require. If, during execution, opportunistic science is able to take more images than predicted, or the images require more memory than anticipated, the task manager will detect that the task has used up the memory it has allocated. At this point, if it would like to take another image, it will use ETQ to see if it can use more memory without disrupting the plan. In its query it states the amount of additional memory it would like to use. If the scheduling facility determines that this extra usage will not cause conflicts, it will give opportunistic science the permission to take the image. However, opportunistic science will have to check again if it needs extra memory beyond this new allotment. If the additional use would lead to conflicts, opportunistic science would be denied and would have to stop taking images.

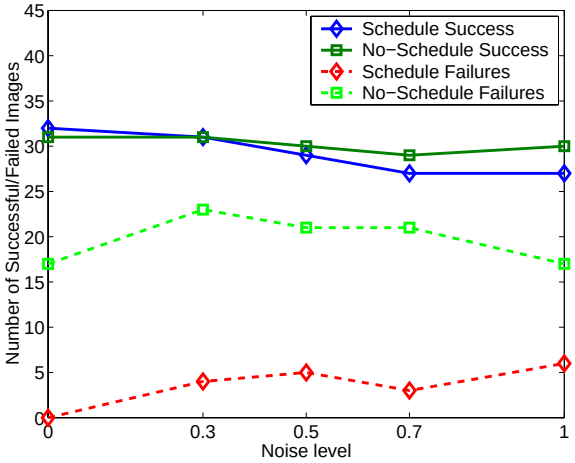


Figure 8: Successful and failed science image attempts with scheduling and non-scheduling versions of ARM

Note that ARM is unable to perform this function because it does not have the long-term picture of the plan that the deliberator has. Further because ARM currently only addresses atomic resources, it is unaware of future memory requirements and, thus, does not know whether or not a task using extra memory now will cause problems in the future.

Evaluation of ARM

We ran a series of tests to evaluate the impact ARM has on the execution of concurrent activities that make intermittent use of a shared resource. Our main objective with the evaluation was to determine if there is any benefit to using the scheduling capability of ARM. Our intention in designing this capability was to allow ARM to use predictions about resource usage to do simple form of scheduling in an attempt to avoid preempting tasks. The study looks at the impact of ARM's scheduling on task preemption.

Methodology:

Our evaluation scenario is based on the opportunistic traverse science task described earlier. In our example, the objective of traverse science is to take as many images as it can during a long range traverse. Before taking an image, it must first request the resource from ARM stating when it would like to use the camera and for how long. When making the request, the science task tries to get the resource as soon as possible but is willing to accept it any time until the end of the traverse. We used a duration of 4 seconds.

The traverse task uses the camera at different intervals to look for obstacles in the path. Based on the images it will plan a path that is typically 35 centimeters in length. It does not use the camera while following that path and thus it can be made available to other tasks, in this case the science task. After each image is taken, the traverse task makes a prediction for when it will need the camera again, based on how fast it moves and the length of the planned path and requests the resource from ARM for that predicted time. For the purpose of evaluation we picked nominal values of 14 seconds between camera uses and 4 seconds for using the camera.

To test the performance of ARM under uncertainty we included noise with these numbers. For each run we select a different probability *p*. With probability 1-*p*, the predicted and actual resource use will be the times stated above. With probability *p*, the predicted duration and time before the next camera use will be drawn randomly from 2-6 seconds for the duration of use and 9-19 seconds for the time before the next use. Because the predicted camera use will not always match the actual use (e.g. it may take longer to travel the planned distance) we also vary the actual use. Again, with probability 1-*p*, the actual use will match the predicted use. With probability *p* we randomly pick the duration and time before the next camera use from the same intervals used for the predicted use.

For priorities, we gave the traverse task a higher priority than the science task.

We ran two versions of the system Schedule and No-Schedule. The Schedule version works as described above using predicted resource usage information to avoid preemption. The No-Schedule version does not take resource reservations. Instead it simply give the resource to the higher priority tasks whenever it requests it.

Results:

Figure 8 contains the results from running each system at 5 different noise levels. Each entry in the table indicates the number of successful traverse science images taken along with the number of failed science images. In each case, a failed science image represents a preempted science task. Note that because the traverse task had a higher priority, it was never preempted.

Discussion:

Overall, these results show that the scheduling capability of ARM is effective in avoiding the preemption of tasks. Without any noise, the schedule version worked perfectly and did not preempt a single task. At higher levels of noise, a few tasks were preempted but much fewer than the No-Schedule version at the same noise levels. Because the Schedule version is a bit more conservative than the No-Schedule version, there were slightly fewer successful images taken. However, considering that a failed image corresponds to wasted power, the cost of a small number of missed images is likely to be much smaller than the cost of wasted power.

Related work

There have been many techniques for combining deliberative and reactive reasoning into hybrid architectures for robotic applications. These architectures have been successfully applied to many dynamic and uncertain real-world domains including manufacturing [Lyons and Hendriks, 1995], military operations [Arkin, 1997; Myers, 1998] and space exploration [Gat, 1992; Washington, et al, 1999; Pell, et al, 1997].

[Arkin 1998] and [Knight, et al, 2001] contain surveys of many hybrid architectures. Only a few of these architectures were designed with resource constraints and tight deadlines in mind. Consequently, there has been little work in addressing these issues in dynamic, uncertain environments. Without some facility for reasoning about resources and deadlines, there is a danger that the robot will not detect problems in the plan until it is too late to do anything about it.

However, there are some architectures that are capable of reasoning about resources and deadlines. CIRCA (Musliner et al. 1993) contains a scheduler that enforces hard real-time constraints for a mobile robot navigation domains. However, rather than repair the schedule, it returns failure if it cannot meet the hard real-time constraints. CPEF [Myers, 1998] uses the SIPE-2 [Wilkins, 1988] planning system which is capable of resource management. CPEF is unique in its ability to perform indirect commanding, in which the system supervises a collection of entities executing the plan along with its ability to accept user advice for plan development. [Washington, et al, 1999] present a system that can perform resource management that is also applied to the Mars exploration domain. To deal with uncertain resource and time usage, their system precompiles resource envelopes to provide task management flexibility to the executive. The system also performs contingency planning to deal with the set of most probable plan deviations.

CLEaR extends the capabilities of the previous systems by providing execution-time atomic resource management as part of the reactive reasoning. The CLEaR framework also provides the reactive reasoning components with limited access to the global view of the plan through the execution-time query facility.

With respect to individual components of CLEaR, ARM shares some similarities with the resource manager in [Gat 2000]. In particular, both components represent execution-time resource managers. However, Gat was concerned with providing hard-real-time guarantees, and enforcing resource reservations through gate keeping access to the resources. In contrast, we settled for a soft-real-time system that, through the use of limited search, can do a small amount of look-ahead to avoid task preemption. Thus ARM supports execution-time decision-making, while [Gat 2000] provides execution-time resource safety.

Future work

Our future work will involve the use of different types of execution-time resource management (including aggregate resources), means of better utilizing path-planning algorithms in conjunction with planning and execution, means of performing quick local plan repairs while minimizing global plan risk, and finding other ways of applying our unified planning and execution framework to improve mission operations, increase science return and enable more efficient long-range traverses.

We are also applying the CLEaR system to ground station automation for NASA's Deep Space Network (DSN) [DSN 1994; Fisher, et al, 1998, 2000]. In this domain area CLEaR is used in a similar fashion to generate command sequences for commanding the ground station communications subsystems to communicate with assets in deep space, whether that be Earth orbiters, spacecraft in orbit around Mars, on the surface of Mars, or as far out as Voyager I & II now beyond the edge of our solar system. The CLEaR software has also been licensed by Lockheed Martin Skunk Works division for use in automating the pilot functionality of Unmanned Air Vehicles (UAVs).

Conclusions

Resource constraints and tight deadlines pose challenges beyond those found in other uncertain robotic environments. In these applications, a task may require a different amount of time or resource than anticipated, potentially leading to execution failure at future points in the plan. We have developed the CLEaR framework to address these challenges. CLEaR extends previous work in hybrid deliberative/reactive architectures in three ways. A continuous planning and scheduling system allows the robot to identify and repair problems before they occur, while continuing to perform other tasks. ARM provides execution-time atomic resource management enabling the scheduling of concurrent tasks that require intermittent use of the same resource, while avoiding the need for task preemption. Finally, ETQ provides the executive with access to the global plan perspective needed to prevent tasks from deviating from time and resource allocations in situations when doing so will lead to conflicts. This framework will increase the effectiveness of robots in many real-world applications, including the space exploration mission presented in our case study.

Acknowledgement

This work was performed by the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration. This work has been partially supported by: the Inter-Planetary Network Information Systems Directorate (IPN-ISD) Technology Program's Mission Planning and Execution (MP&E), the Intelligent Systems' Automated Reasoning program, and the Mars Technology Program's CLARAty architecture effort. This work has also leveraged a previous large body of work including JPL's ASPEN/CASPER planning and scheduling system and CMU's Task Description Language (TDL). We would like to thank all of those involved. We would like to recognize past CLEaR team members Darren Mutz and Barbara Engelhardt. We also thank Brad Clement for his editorial comments.

References

Arkin, R., and Balch, T., 1997, AuRA: Principles and Practice in Review, *Journal of Experimental and Theoretical Artificial Intelligence,* 9(2)

Arkin, R., 1998, *Behavior-Based Robotics*, Cambridge, Massachusetts, MIT Press

Bonasso, R.P., Firby, R.J., Gat, E., Kortenkamp, D., Miller, D.P., and Slack., M.G., 1997, Experiences with an Architecture for Intelligent, Reactive Agents, In *Journal of Experimental and Theoretical Artificial Intelligence*

Chien, S., Knight, R., Stechert, A., Sherwood, R., and Rabideau, G., 2000a, Using Iterative Repair to Improve the Responsiveness of Planning and Scheduling, In *Proceedings of the 5th International Conference on Artificial Intelligence Planning and Scheduling*, Breckenridge, CO

Chien, S., Rabideau, G., Knight, R., Sherwood, R., Engelhardt, B., Mutz, D., Estlin, T., Smith, B., Fisher, F., Barrett, T., Stebbins, G., and Tran, D. 2000b. ASPEN - Automating Space Mission Operations using Automated Planning and Scheduling, In *Proceedings of the SpaceOps 2000 Conference*, Toulouse, France.

(DSN, 1994) Deep Space Network, Jet Propulsion Laboratory Publication 400-517, April 1994.

Fisher, F., Knight, R., Engelhardt, B., Chien, S., and Alejandre, N., 2000, A Planning Approach to Monitor and Control for Deep Space Communications", In *Proceedings of the IEEE Aerospace Conference*

Fisher, F., Estlin, T., Mutz, D., and Chien, S., 1998, Using Artificial Intelligence Planning to Generate Antenna Tracking Plans, In *Proceedings of the Conference on Innovative Applications of Artificial Intelligence*, Orlando, Florida

Gat, E., 2000, Hard-real-time Resource Management for Autonomous Spacecraft, In *Proceedings of the 2000 IEEE Aerospace Conference*. Big Sky, MT.

Gat, E., 1998, On Three-Layer Architectures, In D. Kortenkamp, R. P. Bonnasso, and R. Murphy, eds. *Artificial Intelligence and Mobile Robots*. MIT/AAAI Press. Cambridge, MA, 1998.

Gat, E., 1992, Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots. In *Proceedings of the National Conference on Artificial Intelligence*

Knight, R., Fisher, F., Estlin, T., Engelhardt, B., and Chien, S., 2001 Balancing Deliberation and Reaction, Planning and Execution for Space Robotic Applications, In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, Maui, Hawaii,

Lyons, D. and Hendriks, A., 1995 Planning as incremental adaptation of a reactive system, *Robotics and Autonomous Systems*, 14(4), 255-288

Minton, S., and Johnston, M. 1988. Minimizing Conflicts: A Heuristic Repair Method for Constraint Satisfaction and Scheduling Problems." *Artificial Intelligence*, 58:161-205.

Mishkin, A., Morrison, J., Nguyen, T., Stone, H. Cooper, B., Wilcox, B., 1998, Experiences with Operations and Autonomy of the Mars Pathfinder Microrover, *Proceedings of IEEE Aerospace Conference*, Snowmass at Aspen

Musliner, D.J., Durfee, E., and Shin, K., 1993, CIRCA: A cooperative, intelligent, real-time control architecture, *IEEE Transactions on Systems, Man and Cybernetics*, 23(6)

Myers, K. L. 1998, Towards a Framework for Continuous Planning and Execution, In Proceedings of the AAAI Fall Symposium on Distributed Continual Planning

Pell, B., Gat, E., Keesing, R., Muscettola, R., and Smith, B., 1997b. Robust periodic planning and execution for autonomous spacecraft, In *Proceedings of the Int'l Joint Conference on Artificial Intelligence*

Rabideau, G., Engelhardt, B., and Chien, S. 2000. Using Generic Preferences to Incrementally Improve Plan Quality. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling*, Breckenridge, CO.

Simmons, R. and Apfelbaum, D., 1998 A Task Description Language for Robot Control, In *Proceedings of Conference on Intelligent Robotics and Systems*, Vancouver Canada

Washington, R., Golden, K., Bresina, J., Smith, D.E., Anderson, C., and Smith, T. 1999. Autonomous Rovers for Mars Exploration. In *Proceedings of the 1999 IEEE Aerospace Conference*. Aspen, CO.

Wilkins, D. E., 1988, Causal Reasoning in Planning, *Computational Intelligence*, vol. 4, no. 4, pp. 373--380

Zweben, M., Daun, B., Davis, E., and Deale, M. 1994. Scheduling and Rescheduling with Iterative Repair, In *Intelligent Scheduling*, Morgan Kaufmann, San Francisco, CA. 241-256.

Programming Hierarchical Task Networks in the Situation Calculus

Alfredo Gabaldon
Department of Computer Science
University of Toronto
alfredo@cs.toronto.edu

Introduction

Hierarchical Task Network (HTN) planning (Sacerdoti 1974) is an approach to planning where problem-specific knowledge is used to remedy the computational intractability of classical planning. This knowledge is in the form of task decomposition directives, i.e. the planner is given a set of *methods* that tell it how a high-level task can be decomposed into lower-level tasks. The HTN planning problem consists in computing a sequence of primitive tasks that corresponds to performing the initial set of high-level tasks.

Our purpose in this paper is 1) to give an account of HTN-planning as high-level programming in the situation calculus (McCarthy 1963) based languages Golog/ConGolog (Levesque *et al.* 1997; Giacomo, Lesperance, & Levesque 2000) and 2) to illustrate our approach with a ConGolog encoding of a logistics domain HTN-planning problem. The Golog/ConGolog languages have been extended to deal with explicit time, sensing actions, exogenous events, execution monitoring, incomplete knowledge of the initial state, stochastic actions and others. Thus the range of problems that can be tackled with this approach is potentially much larger. As an example, we modified the logistics domain encoding to execute on-line and deal with run-time exogenous delivery requests.

Preliminaries

The Situation Calculus

The situation calculus (McCarthy 1963) is a logical language for axiomatizing dynamic worlds. Intuitively, it has three basic components: *actions*: responsible for all the changes in the world; *situations*: sequences of actions which represent possible histories of the world; and *fluents*: relations and functions which represent properties of the world and whose values change from situation to situation.

We will use the definition of the situation calculus and the axiomatization of situations as it appears in (Levesque, Pirri, & Reiter 1998; Reiter 2001). The language of the situation calculus includes function symbols for actions, for example, $loadTrk(obj, trk)$ could stand for the action of loading obj into truck trk. It includes a special constant S_0 that denotes the *initial situation* and a function symbol $do(\alpha, s)$ that denotes the situation that results from doing action α in situation s. For example, the situation term

$$do(driveTrk(Trk_1, Loc_1, Loc_2), do(loadTrk(A, Trk_1), S_0))$$

denotes the history of the world consisting of the sequence of actions

$$[loadTrk(A, Trk_1), driveTrk(Trk_1, Loc_1, Loc_2)].$$

Relational fluents and *functional fluents* are relations and functions, resp., whose last argument is a situation. Examples of these are a relation $atTruck(Trk_1, Loc_1, S_0)$ meaning that Trk_1 is at Loc_1 in the initial situation, and function $temperature(Room_1, s)$ denoting the temperature value of $Room_1$ in situation s.

A situation calculus axiomatization of a domain includes[1]:

1. Action precondition axioms: For each action function $A(\vec{x})$ an axiom of the form $Poss(A(\vec{x}), s) \equiv \Pi_A(\vec{x}, s)$ where $\Pi_A(\vec{x}, s)$ is a formula with free variables among $\vec{x}, s$ and s is its only situation term. These axioms characterize the (situation dependent) preconditions for the execution of primitive actions.
2. Successor state axioms: For each relational fluent $F(\vec{x}, s)$ an axiom of the form $F(\vec{x}, do(a, s)) \equiv \Phi_F(\vec{x}, a, s)$ where $\Phi_F(\vec{x}, a, s)$ has free variables among $\vec{x}, a, s$ and s is its only situation term. Similar axioms are included for functional fluents. These axioms characterize the value of fluents after executing a primitive action a in situation s. These axioms embody Reiter's solution to the frame problem for deterministic actions (Reiter 1991).
3. Unique names axioms for actions.
4. Axioms describing the initial situation: A finite set of sentences whose only situation term is S_0 and which describe what is initially true, before any actions have occurred.

Example 1 Our main example through out this paper will be a logistics domain problem. There are objects that are to be moved between locations by truck or plane. Cities contain different locations some of which are airports. Primitive actions include loading/unloading an object onto a truck or

[1] Arguments in predicates and formulas starting with a lowercase letter denote variables. Free variables are implicitly universally quantified.

plane, driving a truck and flying a plane. The following is an axiomatization of this domain:
Action Precondition Axioms:

$$Poss(loadTruck(o,tr),s) \equiv atTruck(tr,l,s) \wedge atObj(o,l,s).$$
$$Poss(unloadTruck(o,tr),s) \equiv inTruck(o,tr,s).$$
$$Poss(loadAirplane(o,p),s) \equiv atObj(o,l,s) \wedge atAirplane(p,l,s).$$
$$Poss(unloadAirplane(o,p),s) \equiv inAirplane(o,p,s).$$
$$Poss(driveTruck(tr,orig,dest),s) \equiv atTruck(tr,orig,s) \wedge inCity(orig,city) \wedge inCity(dest,city).$$
$$Poss(fly(p,orig,dest),s) \equiv atAirplane(p,orig,s) \wedge airport(dest).$$

Successor State Axioms:

$$atObj(o,l,do(a,s)) \equiv a = unloadTruck(o,tr) \wedge atTruck(tr,l,s) \vee a = unloadAirplane(o,p) \wedge atAirplane(p,l,s) \vee atObj(o,l,s) \wedge a \neq loadTruck(o,tr) \wedge a \neq loadAirplane(o,p).$$
$$atTruck(tr,l,do(a,s)) \equiv a = driveTruck(tr,o,l) \vee atTruck(tr,l,s) \wedge (a \neq driveTruck(tr,o,d) \vee d = l).$$
$$atAirplane(p,apt,do(a,s)) \equiv a = fly(p,oapt,apt) \vee atAirplane(p,apt,s) \wedge (a \neq fly(p,oapt,dapt) \vee dapt = apt).$$
$$inTruck(o,tr,do(a,s)) \equiv a = loadTruck(o,tr) \vee inTruck(o,tr,s) \wedge a \neq unloadTruck(o,tr).$$
$$inAirplane(o,p,do(a,s)) \equiv a = loadAirplane(o,p) \vee inAirplane(o,p,s) \wedge a \neq unloadAirplane(o,p).$$

Unique names axioms for actions:

$$loadTruck(o,tr) \neq unloadTruck(o,tr),$$

$$loadTruck(o,tr) \neq loadAirplane(o,p), \text{ etc.}$$

Initial situation:

$$atAirplane(p,l,S_0) \equiv p = Plane_1 \wedge l = Loc_{5,1} \vee p = Plane_2 \wedge l = Loc_{2,1}.$$
$$atTruck(t,l,S_0) \equiv t = Truck_{1,1} \wedge l = Loc_{1,1} \vee t = Truck_{2,1} \wedge l = Loc_{2,1} \vee \ldots$$
$$airport(loc) \equiv loc = Loc_{1,1} \vee loc = Loc_{2,1} \vee loc = Loc_{3,1} \vee loc = Loc_{4,1} \vee loc = Loc_{5,1}.$$
$$inCity(l,c) \equiv l = Loc_{1,1} \wedge c = City_1 \vee l = Loc_{2,1} \wedge c = City_2 \vee \ldots$$
$$atObj(p,l,S_0) \equiv p = Package_1 \wedge l = Loc_{3,3} \vee p = Package_2 \wedge l = Loc_{3,1} \vee \ldots$$

The above set of axioms forms a complete situation calculus primitive action theory for our logistics domain example.

Golog and ConGolog

The situation calculus based programming languages Golog (Levesque *et al.* 1997) and ConGolog (Giacomo, Lesperance, & Levesque 2000) allow us to define complex actions in terms of the actions in a primitive action theory. The constructs of Golog are the following:

- Test condition: $\phi?$. Test whether ϕ is true in the current situation.
- Sequence: $\delta_1 ; \delta_2$. Execute δ_1 followed by δ_2.
- Nondeterministic action choice: $\delta_1 | \delta_2$. Execute δ_1 or δ_2.
- Nondeterministic choice of arguments: $(\pi x)\delta$. Choose a value for x and execute δ for that value.
- Nondeterministic iteration: δ^*. Execute δ zero or more times.
- Procedure definitions: **proc** $P(\vec{x})$ δ **endProc**. $P(\vec{x})$ is the name of the procedure, $\vec{x}$ its parameters, and δ is the body.

ConGolog has the above constructs plus the following:

- synchronized conditional: **if** ϕ **then** δ_1 **else** δ_2.
- synchronized loop: **while** ϕ **do** δ.
- concurrent execution: $\delta_1 \parallel \delta_2$.
- prioritized concurrency: $\delta_1 \rangle\rangle \delta_2$. Execute δ_1 and δ_2 concurrently but δ_2 executes only when δ_1 is blocked or done.
- concurrent iteration: $\delta^{\parallel}$. Execute δ zero or more times in parallel.
- Interrupt: $\phi \rightarrow \delta$. Execute δ whenever condition ϕ is true.

Example 2 The following is a procedure definition for the logistics domain:

proc $moveObj(o,loc)$
 $(\pi\ oloc, ocity).$
 if $atObj(o,oloc) \wedge inCity(oloc,ocity)$ **then**
  %% if obj. is to be moved within the same city
  if $inCity(loc,ocity)$ **then** $inCityDeliver(o,oloc,loc)$
  else %% else must go by air to destination city
   $(\pi\ dcity).$
   if $inCity(loc,dcity) \wedge dcity \neq ocity$ **then**
    $(\pi\ oaprt, daprt).$
    $(inCity(oaprt,ocity) \wedge inCity(daprt,dcity))?$;
    $inCityDeliver(o,oloc,oaprt)$;
    $airDeliver(o,oaprt,daprt)$;
    $inCityDeliver(o,daprt,loc)$
   else $False?$
 else $False?$
endProc

The formal semantics of ConGolog is defined in terms of relations $Trans(\delta,s,\delta',s')$ and $Final(\delta,s)$.[2] Intuitively, $Trans(\delta,s,\delta',s')$ holds if after executing a single step of program δ in situation s, δ' is what remains of δ to be executed and s' is the resulting situation. $Final(\delta,s)$ means that δ can be considered in a terminating state in situation s.

[2] For the original, simpler semantics of Golog see (Levesque *et al.* 1997; Reiter 2001).

These are some of the axioms for $Trans$ and $Final$ from (Giacomo, Lesperance, & Levesque 2000):

$$
\begin{array}{l}
Trans(nil, s, \delta', s') \equiv False. \\
Trans(a, s, \delta', s') \equiv \\
\quad Poss(a, s) \wedge \delta' = nil \wedge s' = do(a, s). \\
Trans(\phi?, s, \delta', s') \equiv \\
\quad \phi[s] \wedge \delta' = nil \wedge s' = s. \\
Trans(\delta_1; \delta_2, s, \delta', s') \equiv \\
\quad (\exists \gamma)\delta' = (\gamma; \delta_2) \wedge Trans(\delta_1, s, \gamma, s') \vee \\
\quad Final(\delta_1, s) \wedge Trans(\delta_2, s, \delta', s'). \\
Trans((\pi x)\delta, s, \delta', s') \equiv \\
\quad (\exists x) Trans(\delta, s, \delta', s'). \\
Trans(\textbf{if } \phi \textbf{ then } \delta_1 \textbf{ else } \delta_2, s, \delta', s') \equiv \\
\quad \phi[s] \wedge Trans(\delta_1, s, \delta', s') \vee \\
\quad \neg\phi[s] \wedge Trans(\delta_2, s, \delta', s') \\
Trans(\textbf{while } \phi \textbf{ do } \delta, s, \delta', s') \equiv \\
\quad (\exists \gamma).(\delta' = \gamma\,;\ \textbf{while } \phi \textbf{ do } \delta) \wedge \\
\qquad \phi[s] \wedge Trans(\delta, s, \gamma, s'). \\
Trans(\delta_1 \parallel \delta_2, s, \delta', s') \equiv \\
\quad (\exists \gamma)[\delta' = (\gamma \parallel \delta_2) \wedge Trans(\delta_1, s, \gamma, s')] \vee \\
\quad (\exists \gamma)[\delta' = (\delta_1 \parallel \gamma) \wedge Trans(\delta_2, s, \gamma, s')].
\end{array}
$$

$$
\begin{array}{l}
Final(nil, s) \equiv True. \\
Final(a, s) \equiv False. \\
Final(\phi?, s) \equiv False. \\
Final(\delta_1; \delta_2, s) \equiv Final(\delta_1, s) \wedge Final(\delta_2, s). \\
Final((\pi x)\delta, s) \equiv (\exists x) Final(\delta, s). \\
Final(\textbf{if } \phi \textbf{ then } \delta_1 \textbf{ else } \delta_2, s) \equiv \\
\quad \phi[s] \wedge Final(\delta_1, s) \vee \neg\phi[s] \wedge Final(\delta_2, s). \\
Final(\textbf{while } \phi \textbf{ do } \delta, s) \equiv \neg\phi[s] \vee Final(\delta, s). \\
Final(\delta_1 \parallel \delta_2, s) \equiv Final(\delta_1, s) \wedge Final(\delta_2, s).
\end{array}
$$

An abbreviation $Do(\delta, s, s')$, meaning that executing δ in situation s is possible and it legally terminates in situation s', can be defined in terms of the transitive closure of $Trans$ and predicate $Final$:

$$Do(\delta, s, s') \stackrel{\text{def}}{=} (\exists \delta').Trans^*(\delta, s, \delta', s') \wedge Final(\delta', s').$$

$Trans^*$ is defined by a second order situation calculus formula. For details see (Giacomo, Lesperance, & Levesque 2000).

A Prolog interpreter for ConGolog can be obtained almost directly from these axioms and a primitive action theory (Giacomo, Lesperance, & Levesque 2000).

HTN Planning

In this section we briefly review HTN-planning. Our discussion is based on the definitions of HTN-planning from (Erol, Hendler, & Nau 1996). For the primitive tasks, however, we will use situation calculus notation, i.e. we use primitive actions instead of STRIPS-style *HTN operators*. Moreover, we use situations instead of states.

A *primitive task* is an action term $A(\vec{x})$. A *compound task* is a term of the form $tname(\vec{x})$. A *task network* is a pair (T, ϕ) where T is a list of tasks and ϕ a boolean formula of constraints of the forms $(t \prec t')$, (t, l), (l, t), (t, l, t'), $(v = v')$ and $(v = c)$ where t, t' are tasks from T, l is a fluent literal, v, v' are variables and c is a constant. An HTN *method* is a pair (h, d) where h is a compound task and d is a task network. Methods are the HTN construct for building complex tasks from primitive ones.

An HTN *planning problem* is a tuple (d, s, D) where d is a task network, s is a situation, and D is a *planning domain* consisting of a primitive action theory plus a set of methods. A *plan* is a sequence of ground primitive tasks.

Let d be a primitive task network, s be a situation, and D a planning domain. A sequence of primitive tasks σ is a *completion* of d in s, denoted by $\sigma \in comp(d, s, D)$, if σ is a total ordering of a ground instance of the primitive task network d and is executable in s.

Let d be a task network that contains a compound task t and $m = (h, d')$ be a method such that θ is a most general unifier of t and h. Define $reduce(d, t, m)$ to be the task network obtained from $d\theta$ by replacing $t\theta$ with $d'\theta$ and incorporating (see (Erol, Hendler, & Nau 1996) for details) the constraints in d' with those in d. Define $red(d, s, D)$ as the set of all reductions of d by methods of D.

A solution $sol(d, S_0, D)$ to a planning problem (d, S_0, D) is the set of all plans that can be computed in a finite number of reduction steps:

$$
\begin{array}{l}
sol_1(d, S_0, D) = comp(d, S_0, D) \\
sol_{n+1}(d, S_0, D) = \\
\qquad sol_n(d, S_0, D) \cup \bigcup_{d' \in red(d, S_0, D)} sol_n(d', S_0, D) \\
sol(d, S_0, D) = \bigcup_{n < \omega} sol_n(d, S_0, D)
\end{array}
$$

Example 3 The following are methods for moving an object in the logistics domain that correspond to the Golog procedure example above. The first method works for moving an object within the same city. The second is for moving an object between cities.

$$
\begin{array}{l}
(moveObj(o, loc) \\
\quad [t = inCityDeliver(o, oloc, loc)] \\
\qquad (atObj(o, oloc), t) \wedge \\
\qquad (inCity(oloc, ocity), t) \wedge \\
\qquad (inCity(loc, ocity), t) \\
)
\end{array}
$$

$$
\begin{array}{l}
(moveObj(o, loc) \\
\quad [t_1 = inCityDeliver(o, oloc, oaprt), \\
\quad\ t_2 = airDeliver(o, oaprt, daprt), \\
\quad\ t_3 = inCityDeliver(o, daprt, loc)] \\
\qquad (atObj(o, oloc), t_1) \wedge (inCity(oloc, ocity), t_1) \wedge \\
\qquad (inCity(loc, dcity), t_1) \wedge (ocity \neq dcity, t_1) \wedge \\
\qquad (inCity(oaprt, ocity), t_1) \wedge \\
\qquad (inCity(daprt, dcity), t_1) \wedge \\
\qquad (t_1 \prec t_2) \wedge (t_2 \prec t_3) \\
)
\end{array}
$$

Programming HTNs in Golog/ConGolog

In this section we show how HTN-planning problems can be encoded in Golog/ConGolog. Let us first consider task networks which are totally ordered and with a constraint formula ϕ that is a conjunction of constraints of the form (l, t). This is the type of task networks the HTN-planning system SHOP (Nau *et al.* 1999) is designed to solve.

Totally ordered task networks can be encoded in Golog since there is no concurrency among the tasks.

Totally ordered task networks

Consider an HTN-planning problem $P = (d, S_0, D)$. We encode the methods $(h, d_1), (h, d_2), \ldots, (h, d_k)$ of each compound task h as a Golog procedure as follows:

proc h
$(L_{1,1})?\ ;\ t_{1,1}\ ;\ldots;\ (L_{1,i_1})?\ ;\ t_{1,i_1}\ |$
$(L_{2,1})?\ ;\ t_{2,1}\ ;\ldots;\ (L_{2,i_2})?\ ;\ t_{2,i_2}\ |$
$\ldots$
$(L_{k,1})?\ ;\ t_{k,1}\ ;\ldots;\ (L_{k,i_k})?\ ;\ t_{k,i_k}$
endProc

where $t_{i,j}$ is the jth task in d_i and $L_{i,j}$ is a conjunction of the literals l such that $(l, t_{i,j})$ is a constraint in d_i.

Let Δ_P denote the resulting set of Golog procedures. To complete the encoding of the HTN planning problem P we include a Golog program δ_T obtained from the task network d. This program has the same form as that of a single method: $(L_1)?\ ;\ t_1\ ;\ldots;\ (L_l)?\ ;\ t_l$.

The HTN planning problem can now be reformulated in terms of the logical semantics of Golog:

$$\mathcal{D}_P \models (\exists s)\, Do(\Delta_P\ ;\ \delta_T, S_0, s)$$

Here, $\mathcal{D}_P$ is the primitive action theory of P plus the axioms of Golog.

The procedure in Example 2 is an encoding of the methods in Example 3, except that instead of using nondeterministic choice of actions, i.e. operator $|$, we used if-statements since the conditions before the first tasks are mutually exclusive.

Partially ordered task networks

Before we move on to partially ordered task networks, let us comment on enforcing constraints of the values of literals, i.e. constraints of the forms (l, t), (t, l) and (t, l, t') and their boolean combination. Intuitively, one way to think about these constraints is that their purpose is for eliminating or "pruning" some of the plan candidates. Their purpose is similar to that of the temporal constraints used by Bacchus and Kabanza (1995; 2000) for controlling search in a forward chaining classical planner. Reiter uses this technique in a Golog implementation of several classical planners (Reiter 2001). The idea is to use a predicate $badSituation(s)$ to encode constraints and check them before adding a primitive action to the plan being computed. So in the remainder of the paper, we will assume that these constraints have been suitably encoded by means of a $badSituation$ predicate.

Furthermore, we will assume that the partial order boolean formula is a conjunction of atoms $(t \prec t')$. This is not a limitation since an unrestricted formula can also be enforced through the $badSituation$ predicate. However, if the partial order formula is a conjunction, it is computationally better to enforce it imperatively in the program.

Let us now consider encoding partial order HTN planning problems in ConGolog. As before, for each method there will be a procedure, but we also need to introduce two fluents and two actions which are used to enforce the partial ordering among tasks: fluent $executing(p(\vec{x}), s)$ meaning that the ConGolog procedure p is executing in situation s, fluent $terminated(p(\vec{x}), s)$ meaning that the basic action or procedure p has executed and terminated in situation s, action $start(p(\vec{x}))$ which causes $executing(p(\vec{x}), s)$ to become true, and $end(p(\vec{x}))$ which causes $executing(p(\vec{x}), s)$ to become false and $terminated(p(\vec{x}), s)$ to become true. Both fluents are initially false for all procedures and actions and the two actions are the only ones that change these fluents' truth value. Formally, the successor state axioms for these fluents are the following:

$$\begin{aligned} executing(p(\vec{x}), do(a, s)) \equiv\ & a = start(p(\vec{x})) \vee \\ & executing(p(\vec{x}), s) \wedge a \neq end(p(\vec{x})). \end{aligned}$$

$$\begin{aligned} terminated(p(\vec{x}), do(a, s)) \equiv\ & a = p(\vec{x}) \vee a = end(p(\vec{x})) \vee \\ & terminated(p(\vec{x}), s). \end{aligned}$$

Let d be a task network and t one of its tasks. Let $nexec(t)$ stand for $\neg executing(t) \wedge \neg terminated(t)$. Let $pred(t, d)$ stand for the conjunction:

$$\bigwedge_{\{t':(t' \prec t) \in d\}} terminated(t)$$

If there is no constraint $(t \prec t_i)$ in d then $pred(t, d) = True$.

The ConGolog procedure that encodes the methods $(h, d_1), (h, d_2), \ldots, (h, d_k)$ for a compound task h is:

proc $h\ \delta_1 | \delta_2 | \ldots | \delta_k$ **endProc**

where

$$\delta_i \stackrel{\text{def}}{=} \begin{array}{l} pred(t_{i,1}) \wedge nexec(t_{i,1}) \rightarrow t_{i,1}\ || \\ pred(t_{i,2}) \wedge nexec(t_{i,2}) \rightarrow t_{i,2}\ || \\ \ldots \\ pred(t_{i,k_i}) \wedge nexec(t_{i,k_i}) \rightarrow t_{i,k_i} \end{array}$$

The $t_{i,j}$s are the tasks in d_i. The δ_is consist of a set of interrupts one for each subtask. As soon as the predecessors of a task that has not yet executed terminate, the interrupt fires and the task executes.

Example 4 This is a simple blocks world example method for moving a block v_1 from a block v_2 onto a block v_3:

$$\begin{array}{l} (move(v_1, v_2, v_3) \\ \quad [clear(v_1),\ clear(v_3),\ unstack(v_1, v_2), stack(v_1, v_3)] \\ \quad\quad (clear(v_1) \prec unstack(v_1, v_2)) \wedge \\ \quad\quad (clear(v_3) \prec unstack(v_1, v_2)) \wedge \\ \quad\quad (unstack(v_1, v_2) \prec stack(v_1, v_3)) \\ ) \end{array}$$

The encoding as a ConGolog procedure is the following:

proc $move(v_1, v_2, v_3)$
$nexec(clear(v_1)) \rightarrow clear(v_1)\ ||$
$nexec(clear(v_3)) \rightarrow clear(v_3)\ ||$
$nexec(unstack(v_1, v_2)) \wedge terminated(clear(v_1)) \wedge$
$\quad terminated(clear(v_3)) \rightarrow unstack(v_1, v_3)\ ||$
$nexec(stack(v_1, v_3)) \wedge terminated(unstack(v_1, v_2))$
$\quad \rightarrow stack(v_1, v_3)$
endProc

It is not always possible but in many cases the partial ordering of tasks can be captured without introducing extra fluents. For instance, the procedure for $move(v_1, v_2, v_3)$ can clearly be written in the following simpler way:

proc $move(v_1, v_2, v_3)$
$(clear(v_1) \mid\mid clear(v_3))$;
$unstack(v_1, v_2)$; $stack(v_1, v_3)$
endProc

On-line Execution with Exogenous Actions

The situation calculus and Golog/ConGolog are very powerful languages which allow one to solve problems well beyond the capabilities of today's HTN-planners. In this section we present an encoding of the logistics domain of the previous examples for execution on-line and handling of exogenous delivery requests at run-time. We also show some sample runs using a ConGolog interpreter in Prolog.

On-line execution of a ConGolog program means that once the first primitive action is determined according to the control structure of the program, which due to nondeterminism may involve randomly choosing one, this action is actually executed in the world. This means that our ConGolog interpreter should not backtrack after choosing such an action. Luckily, this behaviour is very easy to realize in Prolog using a cut. The off-line interpreter includes the rule:

```
offline(Prog,S0,Sf):-
          final(Prog,S0), S0=Sf ;
          trans(Prog,S0,Prog1,S1),
          offline(Prog1,S1,Sf).
```

To prevent the interpreter from backtracking on primitive actions, including exogenous ones, we simply add a cut after a one step transition is chosen:

```
online(Prog,S0,Sf):-
          final(Prog,S0), S0=Sf ;
          trans(Prog,S0,Prog1,S1), !,
          online(Prog1,S1,Sf).
```

This is a *brave* online interpreter. A *cautious* one may, for instance, check offline that the remainder of the program successfully terminates before committing to an action:

```
online(Prog,S0,Sf):-
          final(Prog,S0), S0=Sf ;
          trans(Prog,S0,Prog1,S1),
          offline(Prog1,S1,Soff), !,
          online(Prog1,S1,Sf).
```

These issues are further discussed in (Giacomo, Reiter, & Soutchanski 1998; Reiter 2001).

Let us now turn to exogenous actions. Although an agent, or in our case the logistics program, does not have control over when exogenous actions occur, its background theory includes axioms informing it what exogenous actions can occur and what their effects are. In our logistics example, we only consider one exogenous action: $requestDelivery(obj, loc)$ meaning that a request to deliver obj to loc has been issued. Exogenous actions will be generated by having the interpreter ask the user to input them.

Following (Giacomo, Lesperance, & Levesque 2000), we will model exogenous actions by defining a special procedure which will execute in parallel with the logistics main procedure:

proc $exoProg$
$(\pi e)(exoActionOccurred(e) \rightarrow e)$
endProc

The condition $exoActionOccurred(e)$ always succeeds when evaluated and it comes back with a user supplied value for e which can be an exogenous action, nil which means no exogenous action occurred, or $endSim$ which is just as nil but tells the interpreter to stop asking the user for exogenous actions. We could alternatively have had them generated randomly without complication.

Now, the main logistics procedure is a program that reacts to the occurrence of exogenous actions $requestDelivery(obj, loc)$ by triggering the execution of a $moveObj(obj, loc)$ task:

proc $deliveryDaemon$
$(\pi pck, loc)\ deliveryReq(pck, loc) \rightarrow$
$startDelivery(pck, loc)$;
$[(moveObj(pck, loc)$;
$endDelivery(pck, loc))$ $\mid\mid$
$deliveryDaemon]$
endProc

The main ConGolog program is the parallel execution of the logistics procedure and the exogenous actions procedure: $exoProg \mid\mid deliveryDaemon$.

Here is a sample run in Eclipse Prolog:

```
[eclipse 2]: runSim.
startSim
  Enter an exogenous action:
        requestDelivery(package1, loc5-1).
requestDelivery(package1, loc5-1)
startDelivery(package1, loc5-1)
  Enter an exogenous action:  nil.
driveTruck(truck3-1, loc3-1, loc3-3)
  Enter an exogenous action:  nil.
loadTruck(package1, truck3-1)
  Enter an exogenous action:  nil.
driveTruck(truck3-1, loc3-3, loc3-1)
unloadTruck(package1, truck3-1)
  Enter an exogenous action:  nil.
fly(plane1, loc5-1, loc3-1)
  Enter an exogenous action:
  requestDelivery(package2, loc3-2).
requestDelivery(package2, loc3-2)
loadAirplane(package1, plane1)
fly(plane1, loc3-1, loc5-1)
unloadAirplane(package1, plane1)
startDelivery(package2, loc3-2)
  Enter an exogenous action:  nil.
endDelivery(package1, loc5-1)
loadTruck(package2, truck3-1)
driveTruck(truck3-1, loc3-1, loc3-2)
```

```
unloadTruck(package2, truck3-1)
  Enter an exogenous action:
  requestDelivery(package3, loc1-3).
requestDelivery(package3, loc1-3)
  Enter an exogenous action:  nil.
startDelivery(package3, loc1-3)
endDelivery(package2, loc3-2)
driveTruck(truck2-1, loc2-1, loc2-3)
  Enter an exogenous action:  nil.
loadTruck(package3, truck2-1)
driveTruck(truck2-1, loc2-3, loc2-1)
unloadTruck(package3, truck2-1)
  Enter an exogenous action:  nil.
loadAirplane(package3, plane2)
  Enter an exogenous action:  nil.
fly(plane2, loc2-1, loc1-1)
  Enter an exogenous action:  nil.
  Enter an exogenous action:  endSim.
endSim
unloadAirplane(package3, plane2)
loadTruck(package3, truck1-1)
driveTruck(truck1-1, loc1-1, loc1-3)
unloadTruck(package3, truck1-1)
endDelivery(package3, loc1-3)

Plan length: 32 More?  n.
```

The non-indented lines are primitive tasks appearing in the order they occur. The user is prompted for an exogenous action every time the condition $exoActionOccurred(e)$ is evaluated. This happens every time the interpreter computes a transition for the $exoProg$ procedure.

Conclusion

Our purpose was two-fold. On one hand we have argued that HTN-planning can be thought of as a special case of high-level programming in the sense of Golog/ConGolog. We have done this by showing an encoding of HTN-planning problems in these languages. In doing this, we only took advantage of a few of their constructs and of the techniques which have been developed for the many problems that have arisen in cognitive robotics research. These techniques are obviously relevant to planning given that both problems involve modeling dynamic worlds. The work by the Cognitive Robotics group at the U. of Toronto includes formalizations for robotic control that account for explicit time of action occurrence, sensing and knowledge, execution monitoring, stochastic actions, action choice based on decision theory, and others.[3] Our second goal was to actually show a generalization of HTN-planning, after taking this programming perspective, by taking a classic HTN-planning problem, a logistics domain problem, and encoding it in ConGolog for online execution and run-time exogenous actions.

We were not the first to point out a connection between HTN-planning and high-level languages Golog and ConGolog. Baral and Son (1999) extended ConGolog with an HTN construct. In the extended language, a program may include an HTN-planning problem as a statement. However, the new construct is limited: the tasks appearing in it cannot be ConGolog programs. One has to separately define methods for the compound tasks mentioned in an HTN-statement.

[3] Much of this work can be found at http://www.cs.toronto.edu/cogrobo

Acknowledgments

We are thankful to Ray Reiter and Fahiem Bacchus for helpful discussions on the subject of this paper.

References

Bacchus, F., and Kabanza, F. 1995. Using temporal logic to control search in a forward chaining planner. In *Proceedings of the Third European Workshop on Planning*.

Bacchus, F., and Kabanza, F. 2000. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence* 16:123–191.

Baral, C., and Son, T. C. 1999. Extending ConGolog to allow partial ordering. In *Proc. of the Sixth International Workshop on Agent Theories, Architectures, and Languages (ATAL-99)*, volume 1757 of *LNCS*, 188–204.

Erol, K.; Hendler, J. A.; and Nau, D. S. 1996. Complexity results for hierarchical task-network planning. *Annals of Mathematics and ArtificialIntelligence* 18:69–93.

Giacomo, G. D.; Lesperance, Y.; and Levesque, H. J. 2000. ConGolog, a concurrent programming language based on the situation calculus. *ArtificialIntelligence* 121:109–169.

Giacomo, G. D.; Reiter, R.; and Soutchanski, M. 1998. Execution monitoring of high-level robot programs. In *Proceedings of the 6th International Conference on Principles of Knowledge Representation and Reasoning (KR'98)*.

Levesque, H. J.; Reiter, R.; Lespérance, Y.; Lin, F.; and Scherl, R. B. 1997. Golog: A logic programming language for dynamic domains. *Journal of Logic Programming* 31(1–3):59–83.

Levesque, H.; Pirri, F.; and Reiter, R. 1998. Foundations for the situation calculus. *Linköping Electronic Articles in Computer and Information Science* 3(18). http://www.ep.liu.se/ea/cis/1998/018/.

McCarthy, J. 1963. Situations, actions and causal laws. Technical report, Stanford University. Reprinted in Semantic Information Processing (M. Minsky ed.), MIT Press, Cambridge, Mass., 1968, pp. 410–417.

Nau, D. S.; Cao, Y.; Lotem, A.; and Munoz-Avila, H. 1999. SHOP: Simple hierarchical ordered planner. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, 968–975.

Reiter, R. 1991. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In Lifschitz, V., ed., *Artificial Intelligence and Mathematical Theory of Computation*. Academic Press. 359–380.

Reiter, R. 2001. *Knowledge in Action: Logical Foundations for Describing and Implementing Dynamical Systems.* Cambridge, MA: MIT Press.

Sacerdoti, E. 1974. Planning in a hierarchy of abstraction spaces. *ArtificialIntelligence* 5:115–135.

Dynamic task sequencing in temporal problems with uncertainty

M.-J. Huguet
INSA and LAAS–CNRS
7, avenue du Colonel Roche
F-31077 Toulouse cdex
huguet@laas.fr

P. Lopez
LAAS–CNRS
7, avenue du Colonel Roche
F-31077 Toulouse cdex
lopez@laas.fr

T. Vidal
ENIT/LGP/PA
47, venue Azereix – BP 1629
F-65016 Tarbes cdex
thierry@enit.fr

Abstract

In classical planning and scheduling approaches, a task schedule is first designed off line, then executed on line, simply releasing tasks at times compatible with temporal domains of their starting time-points. When durations of tasks are uncertain, one wishes to keep as much flexibility on line as possible so as to release each task according to effective durations taken by previous ones. One also wishes to ensure off line that the on-line schedule will be feasible whatever the uncertain durations will be, which has been called temporal controllability of the plan. Such proactive reasoning both leads to predictice schedules that are more robust when executed on line, and also to easier and more effective on-line rescheduling when needed. Going further, the notion of sequentiability has been defined with respect to resource constraints. It means here the ability to decide on line the sequencing of tasks that use the same discrete resource, according to effective durations taken by previous ones. In the non-uncertain framework, algorithms exist to prune the search and detect so-called forbidden precedences among tasks. In this paper we show how these techniques can be extended in temporal problems with uncertainty.

Introduction

Temporal Constraint Satisfaction Problems, and particularly Simple Temporal Problems (STPs) (Dechter, Meiri, & Pearl 1991; Schwalb & Dechter 1997) are frequently used in planning and scheduling applications that involve quantitative time constraints (e.g. (Laborie & Ghallab 1995; Morris, Muscettola, & Tsamardinos 1998)), as they allow fast checking of temporal consistency. A duration between two time-points or the temporal domain of a time-point (set of possible times of occurrence) are represented through intervals of possible values. However this formalism does not adequately address an important aspect of real execution domains: the time of occurrence of some events may not be under the complete control of the execution agent. For example, on a building site, a task might wait for a supply truck, which arrival time is dependent on the traffic, while the task duration itself might depend on the weather conditions. In such cases, the execution agent does not have freedom to select the precise delay between events. Instead, the value is selected by Nature independently of the agent's choices. This can lead to constraint violations during execution even if the STP appeared consistent at plan generation time.

The problem of constraint satisfaction for STPs with Uncertainty was addressed formally in (Vidal & Fargier 1999). Uncertainty means here the effective duration of a task or the effective delay between two particular times (start or end of tasks) still lie within allowed bounds but cannot be decided, and will hence be observed on line during execution. In this setting, the question of temporal feasibility goes beyond mere consistency to encompass the main issue of *Dynamic Controllability*. Essentially, a network is dynamically controllable if there is a strategy for executing on line the time-points under the agent's control that satisfy all requirements. Such a property must consider that the agent will apply the strategy in a chronological way: to decide when to execute next task, she might take advantage of observations made on the occurrence of past uncontrolled events, but she must decide without knowing the effective durations of tasks still to come. Dynamic controllability was proven to be tractable (Morris, Muscettola, & Vidal 2001), through the application of a mere local consistency checking algorithm.

Actually the Dynamic Controllability (checked off line) means the ability to postpone effective timing of tasks until executing them on line, but resting assured that no constraint will ever be violated, whatever the observations are. This can be viewed as a least-committment approach adding flexibility to the planning and execution loop, still ensuring the plan safe execution. In other words, our approach is proactive in the sense that most of the reasoning is made off line, to prove that a schedule will be feasible. But actual scheduling (i.e. assigning times to starting times of tasks), though being now straightforward, is made on line.

In scheduling, complex resource constraints must also be accounted for (Pinedo & Chao 1999). We are interested here in non-preemptive tasks (i.e. they cannot be interrupted) and disjunctive resources (i.e. discrete resources with capacity equal to one). In this framework two tasks competing for the same resource need be sequenced. For example, the same crane might be needed for two unloading tasks that should be both processed within a given time window. Usually, when executing an STP, such decisions have already been made (and the added precedence link proven consistent), since a constraint such as *task i before or after task j* cannot be expressed through simple binary constraints between time-points. Nevertheless, in domains with temporal uncertainties, it might be the case that one sequencing choice

is compulsory to make the network Dynamically Controllable in some situations, while the reverse choice is needed in other situations, none being valid in all situations. This calls for Dynamic Sequencing strategies, which means postponing such decisions until execution. We would then rather be able to check off line that this might be done on line without constraint violations. Such a new property has been called Dynamic Sequentiability (Vidal & Bidot 2001).

In general cases, this property cannot be checked in polynomial time since sequencing is already an NP-complete problem in many scheduling problems without uncertainty (Lenstra, Rinnooy Kan, & Brucker 1977). But there a number of propagation techniques exist to filter out values from temporal domains, that are not compatible with the sequencing constraints, allowing to speed up the search for a feasible schedule or to detect early an inconsistency (Baptiste, Le Pape, & Nuijten 2001). For instance the Forbidden Precedence rule checks whether one of the two possible sequences may be proven infeasible with respect to temporal domains of start and end times of two tasks i and j (Erschler, Roubellat, & Vernhes 1976; Torres & Lopez 2000). A more elaborated one, the Extended Forbidden Precedence rule, which is based on *energetic reasoning* (Lopez & Esquirol 1996), checks it taking also into account all other tasks that use the same resource and might occur between i and j.

In this paper, after recalling some background on the topic in section 2, we will focus in section 3 on the Forbidden Precedence rule; we will show how the rule should be written in the framework of STPs with uncertainty. Section 4 will follow the same lines addressing the Extended Forbidden Precedence rule. We will conclude with a couple of words about foreseen extensions of the work.

Background

A Simple Temporal Network (STN) (Dechter, Meiri, & Pearl 1991) is an STP represented as a graph $< V, G >$ in which the vertices in V are the time-points that are the variables of the problem, while edges (or links) in G are binary numerical constraints g_{xy}, in the shape of simple intervals $[l(g_{xy}), u(g_{xy})]$ of possible durations between two time-points x and y. Please note that the inverse constraint implicitly exists and is $g_{yx} = [l(g_{yx}), u(g_{yx})] = [-u(g_{xy}), -l(g_{xy})]$, and that no specific constraint between x and y results in the initial interval $]-\infty, +\infty[$.

To check global consistency of an STN, one might use filtering techniques, namely arc-consistency (AC) and path-consistency (PC) techniques that both run in polynomial time in STNs. PC for instance is a local shortest path propagation algorithm: it computes any binary constraint g_{xy} between points x and y by intersecting it with all paths going through a third time-point z:

$$g_{xy} = g_{xy} \cap (g_{xz} \odot g_{zx})$$

which amounts to, considering that composition simply sums up the lower and upper bounds of the intervals, and intersection takes the max of the lower bounds and the min of the upper bounds:

$$g_{xy} = [\max(l(g_{xy}), l(g_{xz}) + l(g_{zy})),$$
$$\min(u(g_{xy}), u(g_{xz}) + u(g_{zy}))]$$

AC merely updates the temporal domain of each time-point y (i.e. the interval of possible times for y), by computing the sum of the temporal domain of another time-point x and the interval expressing the duration between x and y. This may be seen as a specific and more restricted case of PC since a temporal domain of x is the duration interval between the origin of time 0 and x. The advantage of PC is that the *complete minimal network* is computed: for any two time-points x and y, PC provides the duration interval containing these and only values that are consistent with the other constraints of the problem. Such strength of PC will be widely used in this paper.

Figure 1 illustrates AC and PC through small examples.

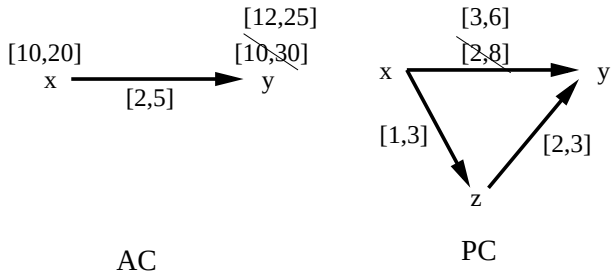


Figure 1: AC and PC algorithms

A Simple Temporal Network with Uncertainty (STNU) is similar to an STN except a subset of G called C represent specific links called *contingent*, which may be thought of as representing causal processes of uncertain duration; their finish timepoints, called *contingent* timepoints, are controlled by Nature, subject to the limits imposed by the bounds on the contingent links. All other timepoints, called *executable* timepoints, are controlled by the agent. Thus, an STNU is a 3-tuple $\Gamma = < V, G, C >$. We require $0 < l(c) < u(c) < \infty$ for each contingent link $c \in C$.

An STNU may be regarded as a family of STNs: a *projection* (Vidal & Fargier 1999) of Γ is an STN derived from Γ, replacing each contingent link c by an interval with equal upper and lower bounds $[v, v]$ for some v such that $l(c) \leq v \leq u(c)$. The set of values v for all the contingent links represent one *situation* that the executing agent might face on line, when durations of contingent links are eventually observed. Then one can define a *schedule* as an assignment of fixed times to all time-points, and an *execution strategy* as a mapping from the set of projections (or situations) to the set of schedules. An execution strategy is *viable* if and only if for all situations the associated schedule is consistent.

Global consistency needed to be redefined in terms of *controllabilities*. We will not get into the details of these properties in this paper (see (Morris, Muscettola, & Vidal 2001) for details), but instead just focus on the most relevant one, the *Dynamic Controllability*: in short, an STNU is dynamically controllable if and only if there exists a viable execution strategy that can be carried out on line *as far as* the contingent durations are observed. Thus, a Dynamic ex-

ecution strategy might be safely run on line, since it assigns a time to each executable timepoint that may depend on the outcomes of contingent links in the past, but not on those in the future (or present). This corresponds to requiring that only information available from observation may be used in determining the schedule on line.

To check dynamic controllability, one might first run a PC algorithm. If a contingent link is squeezed, then it means some of its uncontrollable values are not consistent, therefore the problem will be infeasible for such values. But this is not enough, since the contrary is not true in general. A further PC-like filtering algorithm that still runs in polynomial time might be designed to get the minimal STNU. This algorithm called 3DC+ (Morris, Muscettola, & Vidal 2001) either proves inconsistency or provides the executing agent with duration intervals restricted to dynamically controllable values only, making it possible to safely and easily execute the schedule on line according to observations made. We just quickly recall here the basics of this algorithm (see (Morris, Muscettola, & Vidal 2001) for details): one needs only considering triangles xyz in which one constraint g_{xy} is contingent (if two are contingent the triangle will be considered twice, and if all three links are contingent the triangle is trivially not dynamically controllable). We consider $g_{xy} = [u, v]$, $g_{xz} = [a, b]$ and $g_{zy} = [c, d]$ (figure 2; a contingent constraint is depicted as a dotted arrow).

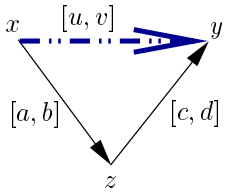


Figure 2: Triangular network for 3DC+

1. case 1: if $d < 0$, then necessarily y is before z and no further restriction is necessary.
2. case 2: if $c \geq 0$, then necessarily z will be executed before y, hence without having observed it yet, therefore g_{xz} must be restricted so that things will work fine whatever values are taken by g_{xy}, which raises $g_{xz} = [a, b] \cap [v - d, u - c]$. For instance, the initial network of figure 3(a) is stable after PC, but is further restricted by 3DC+ to the network of figure 3(b).

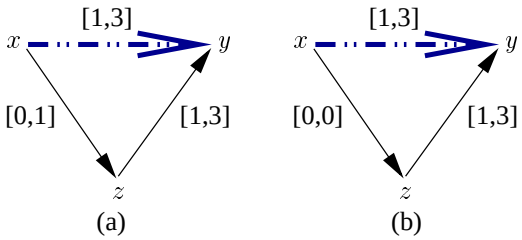


Figure 3: Illustration of 3DC+ case 2

3. case 3: $c < 0$ and $d \geq 0$, i.e. z might occur before or after y. Here the general rule to apply is to update $g_{xz} = [a, b] \cap [\min(v - d, u), b]$ and in the case where $v-d > u$, one needs to add a $wait < y, v-d >$ on the link g_{xz}, which means that after activating x, one should wait either for the occurrence of y or at least $v-d$ time units before activating z. Such $wait$ constraints are added to the STNU model and *regressed*, which is a backward propagation process, still made locally through triangles, and thus still tractable. Figure 4 presents an example where $v - d > u$. As previously, the figure shows the initial network (a) and the network propagated by 3DC+ (b).

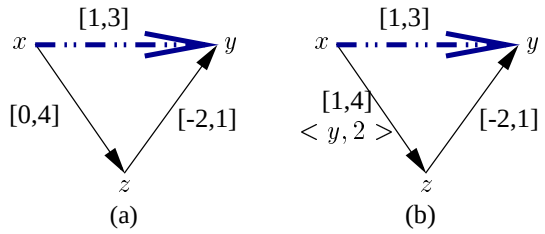


Figure 4: Illustration of 3DC+ case 3

Existing work only focused on temporal constraints, disregarding resource constraints that are of high significance in planning and scheduling applications. As mentioned in the introduction, a disjunctive resource compels all tasks needing this resource to be sequenced. Sequencing decisions, just as task activation time decisions, should be left to the on-line executing agent, since in some situations one sequencing will be needed while the inverse choice must be taken in another situation. Anyhow, to ease feasibility checking of the problem and detect some sequencing that are forbidden anyway, filtering algorithms over sequencing decisions are of high added value. Efficient though non-complete ones exist in classical scheduling; this paper aims at adapting them in the uncertain framework.

In the remaining of the paper, we will use the following notations for each task k:

- $k^- \in [\underline{k}^-, \overline{k}^-]$: start time-point
- $k^+ \in [\underline{k}^+, \overline{k}^+]$: end time-point
- $p_k = k^+ - k^- \in [\underline{p}_k, \overline{p}_k]$: duration
- $w_k^\Delta \in [\underline{w}_k^\Delta, \overline{w}_k^\Delta]$: consumption over an interval Δ

Forbidden Precedence

The reader should keep in mind that the meaning of a forbidden precedence is that adding this precedence to the problem (before execution) would lead to an inconsistency. When temporal uncertainties are accounted for, it is enough to find one situation in which adding this precedence would make the projection inconsistent: such a sequencing may be decided on line when facing a situation that supports it, but it should not be added during off-line scheduling when all situations are still likely to occur. We will first recall the clas-

sical way of expressing the rule in scheduling without temporal uncertainty. This has not been extended to the STN; we need to do this first before extending to the STNU.

The STN context

Let us consider two (non contingent) tasks i and j which must be sequenced (for instance because they compete for the same resource).

Classical rule formulation in scheduling Proposition 1, illustrated by figure 5, allows us to conclude that "i before j" is forbidden (denoted by $i \not\prec j$) (Erschler, Roubellat, & Vernhes 1976).

Proposition 1 If $\overline{j}^+ - \underline{i}^- < \underline{p}_i + \underline{p}_j$ then $i \not\prec j$.

Proof. $\underline{i}^-$ is the earliest possible start time for i , therefore $\underline{i}^- + \underline{p}_i$ is the earliest possible end time for i. Similarly, $\overline{j}^+ - \underline{p}_j$ is the latest possible start time for j. Therefore $i \prec j$ implies that $\underline{i}^- + \underline{p}_i < \overline{j}^+ - \underline{p}_j$. Reverting this proposition gets to Proposition 1. □

Figure 5: Forbidden precedence $i \not\prec j$

New formulation based on a minimal STN One can see the previous rule uses information on the possible times of i^- and j^+, which are in an STN the temporal domains, i.e. the duration between 0 and the time-point.

That may be generalized in a complete minimal network obtained through a PC propagation (figure 6). In this minimal network, a forbidden precedence between i and j, exists if and only if a lower bound of the link j^-i^+ is positive, that means j^- is before i^+, which forbids $i \prec j$. It yields the following proposition.

Proposition 2 *If $a' > 0$ then $i \not\prec j$.*
Symmetrically if $c' > 0$ then $j \not\prec i$.

Proof. In a complete minimal network obtained through a PC propagation, all paths from j^- to i^+ have been searched for. Therefore in the edge j^-i^+ valuated by $[a', b']$, a' is the highest lower bound for j^-i^+. Then $a' > 0$ means $i \not\prec j$. □

Proposition 3 *Proposition 2 subsumes Proposition 1.*

Proof. The proof is straightforward since after PC propagation a' stands for the greatest minimal path between j^- et i^+. In particular $a' \geq \underline{p}_j - \overline{j}^+ + \underline{i}^- + \underline{p}_i$. □

Adaptation to STNUs

We will now assume that some of the tasks may be contingent, which means we need considering an STNU propagated through 3DC+ instead of an STN. In figure 6, that

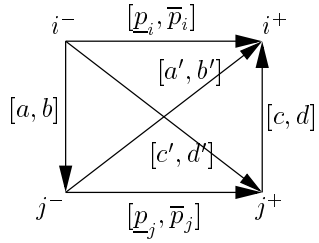


Figure 6: STN after propagation through PC

means the links i^-i^+ and j^-j^+ may be contingent[1].

For instance let i be contingent. After propagation a' corresponds to a shortest path from j^- to i^+, but it might result from a path going through i^-, being the sum $-b+\underline{p}_i$. Since i is contingent we must consider the worst situation in which this path would take longer, this is when i takes its upper bound $\overline{p}_i$. A precedence is indeed forbidden if there exists at least one situation in which it is forbidden. Otherwise the off-line scheduling process could be allowed to enforce such a precedence in the network, which would lead to a possible failure at execution time if i takes its upper bound.

So let us look at the case where $p_i = \overline{p}_i$. One can see there will be a problem if $\overline{p}_i > b$ since j^- would necessarily be released by the execution process before i^+ has occurred. It is easy to check that this new condition $\overline{p}_i - b > 0$ subsumes the previous one $a' > 0$, that is $\overline{p}_i - b \geq a'$. Indeed, if only PC is applied to this network, the link i^-j^- valuated by $[a, b]$ is intersected with the path $i^-i^+j^-$, i.e. $[\underline{p}_i, \overline{p}_i] \odot [-b', -a']$, then one has $b \leq \overline{p}_i - a'$. Since 3DC+ will only restrict further the value of the link i^-j^-, the property $\overline{p}_i - b \geq a'$ is still enforced by 3DC+. In other words, there might be cases in which $\overline{p}_i - b > 0$ while $a' < 0$ as the next example will show. Considering all other possible cases, one gets the following updated rule.

Proposition 4 *If i is not contingent and $a' > 0$ then $i \not\prec j$.*
If i is contingent and $\overline{p}_i - b > 0$ then $i \not\prec j$.
If j is contingent and $\overline{p}_j + c > 0$ then $i \not\prec j$.
Symmetrically,
If j is not contingent and $c' > 0$ then $j \not\prec i$.
If i is contingent and $\overline{p}_i - d > 0$ then $j \not\prec i$.
If j is contingent and $\overline{p}_j + a > 0$ then $j \not\prec i$.

In the example of figure 7, i is a contingent task while j is not. Propagation through PC provides us with $[-1, 9]$ on the link j^-i^+. Condition $i \not\prec j$ might be deduced by Proposition 4 (since $\overline{p}_i - b = 5 - 4 > 0$), but cannot be deduced considering the length of the shortest path from i^+ to j^-, that is a', since $a' = -1 < 0$.

Extended forbidden precedence

We will now introduce a more effective rule in terms of deduction by taking into account other tasks that compete for

[1]We still consider that other links are not contingent, which is usually the case in real-life planning, but our scheme might easily be extended to cases with contingent links between tasks.

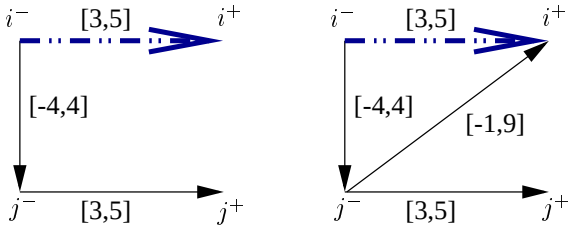


Figure 7: Illustrative example of Proposition 4

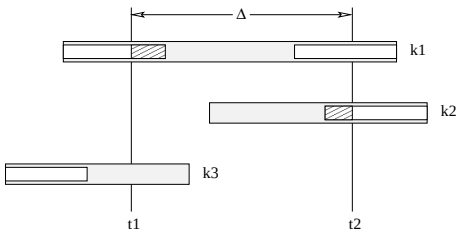


Figure 9: Minimal consumption of three tasks

the same resource and that are to occur within the same reference interval. For that purpose we need to define the *minimal consumption* of a task k over a reference interval Δ (Esquirol, Lopez, & Huguet 2001). As in the previous section, we first give mathematical expressions as classically stated in scheduling, improve them for the STN framework, then adapt them in the STNU.

Classical rule formulation in scheduling

Consumption We denote by w_k^Δ the consumption of task k (i.e. how long k uses the resource) over a reference interval $\Delta = t_1 t_2$. Two cases must be distinguished:

1. $k^- k^+ \bigcap \Delta = \emptyset \Longrightarrow w_k^\Delta = 0$;
2. $k^- k^+ \bigcap \Delta \neq \emptyset \Longrightarrow w_k^\Delta = \min(k^+, t_2) - \max(k^-, t_1)$.

This is illustrated by figure 8 where striped areas represent the consumption of each task between t_1 and t_2.

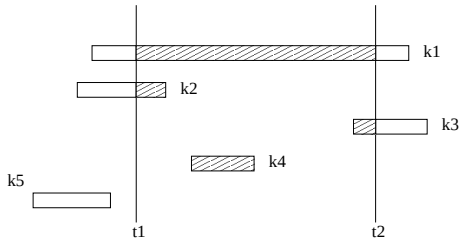


Figure 8: Consumption of five tasks

One then gets:

$$w_k^\Delta = \max[0, \min(k^+, t_2) - \max(k^-, t_1)]$$

which amounts to, knowing that $k^+ - k^- = p_k$

$$w_k^\Delta = \max[0, \min(p_k, t_2 - t_1, k^+ - t_1, t_2 - k^-)] \quad (1)$$

One is usually especially interested in computing the lower and upper bounds of the consumption: for the consumption of task k over interval Δ, we might derive from equation (1) the *minimal* (or *necessary*) consumption noted $\underline{w_k^\Delta}$, and the maximal consumption noted $\overline{w}_k^\Delta$.

The former is obtained by considering the minimal duration of the task, and by shifting it to its left and right utmost positions, retaining the minimum value of all intersections between such positions and the reference interval Δ. That is illustrated in figure 9 and raises:

$$\underline{w_k^\Delta} = \max[0, \min(\underline{p_k}, t_2 - t_1, \underline{k}^+ - t_1, t_2 - \overline{k}^-)] \quad (2)$$

The maximal consumption is on the contrary obtained by considering the maximal duration and positions which intersection with interval Δ is maximal:

$$\overline{w}_k^\Delta = \max[0, \min(\overline{p}_k, t_2 - t_1, \overline{k}^+ - t_1, t_2 - \underline{k}^-)] \quad (3)$$

The relevant notion for our purpose is obviously the minimal consumption: when trying to check whether i before j is feasible, we intend to take into account that another task k will *necessarily* consume the resource, between i^- and j^+, for *at least* some time T. Therefore we will not consider anymore the maximal consumption in the remainder of the paper.

Rule formulation based on temporal domains In the STN context, extending the forbidden precedence rule means taking as a reference interval $\Delta = \underline{i}^- \overline{j}^+$. Proposition 5, illustrated by figure 10, allows the deduction of a forbidden precedence between tasks i and j by considering the minimal consumptions of other tasks competing for the same resource over Δ (Lopez & Esquirol 1996; Esquirol, Lopez, & Huguet 2001).

Proposition 5 *If* $\overline{j}^+ - \underline{i}^- < \underline{p}_i + \underline{p}_j + \sum_{k \neq i,j} \underline{w_k^{i^- \overline{j}^+}}$ *then* $i \not\prec j$.

The proof is straightforward since the minimal consumption expresses the necessity that the other tasks will consume the resource over a subset of the interval.

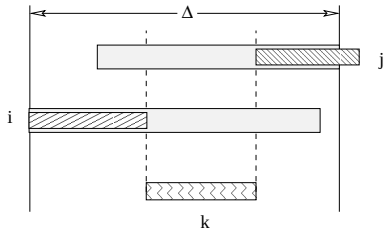


Figure 10: Extended forbidden precedence $i \not\prec j$ accounting for k

New formulation in the STN context

General formulation of the consumption Let us consider the minimal consumption of a task k between i^- and j^+ according to the outcome of a PC algorithm in an STN. Figure 11 illustrates the situation will all relevant links.

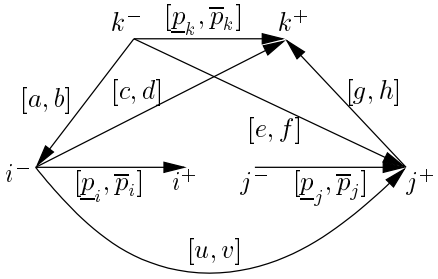


Figure 11: Minimal consumption of a task k in an STN

Here $\Delta = i^-j^+ = [u,v]$ is the reference interval. The general formulation of the minimal consumption of k over this interval is as follows:

$$\underline{w}_k^\Delta = \max\{0, \min[\underline{p}_k, v, \max(c, \underline{p}_k - b), \max(e, \underline{p}_k - h)]\} \quad (4)$$

Let us justify the formulation. The two former terms $\underline{p}_k$ and v correspond to the obvious cases, when the task occurs fully outside Δ, and when it covers the interval Δ, respectively.

The third term is when the task is shifted to the left. In such a situation i^-k^+ will take its lower bound c, and actually the interval i^-k^+ looks like the amount of k that lies after i^-, therefore c should be the quantity to consider. But one may also consider k is left-shifted when k^-i^- takes its upper bound. Then b represents the part of k that lies outside the reference interval. Therefore the quantity $\underline{p}_k - b$ could be considered as the minimal consumption of k. The dominating value will be the maximal one, since a propagation algorithm retains the max of all lower bounds. Similar reasoning leads to e and $\underline{p}_k - h$ as possible quantities to represent the minimal consumption of k when it is right-shifted.

Now all that follows consists in determining which quantity subsumes the other, distinguishing between contingent and non-contingent cases for k.

Extended forbidden precedence rule in a minimal STN Formula (4) can be simplified considering how PC updates the links. Composition of intervals $[-b,-a]$ and $[\underline{p}_k, \overline{p}_k]$ raises $[\underline{p}_k - b, \overline{p}_k - a]$ which is intersected with $[c,d]$. Since the network is stable $\underline{p}_k - b$ should not update c, therefore $c \geq \underline{p}_k - b$. Similarly $e \geq \underline{p}_k - h$. We should then use c for a task shifted to the left and e for a task shifted to the right.

In the network of figure 12, after a propagation through PC one gets: $\underline{p}_k - b = 1 - 1 = 0$ and $c = 1$; $\underline{p}_k - h = 1 - (-1) = 2$ and $e = 4$.

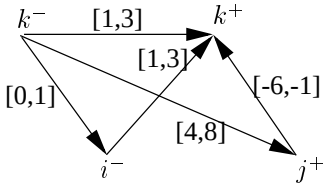


Figure 12: Example for calculating the consumption

The minimal consumption is then expressed by:

$$\underline{w}_k^\Delta = \max[0, \min(\underline{p}_k, v, c, e)] \quad (5)$$

To get the Extended Forbidden Precedence Rule, one needs to consider the reference interval as being $i^-j^+ = [u,v]$. Proposition 5 ends up being:

Proposition 6 *If* $v < \underline{p}_i + \underline{p}_j + \sum_{k \neq i,j} \underline{w}_k^{i^-j^+}$ *then* $i \not\prec j$.

The following example (figure 13) illustrates that Proposition 6 subsumes the classical formulation of Proposition 5.

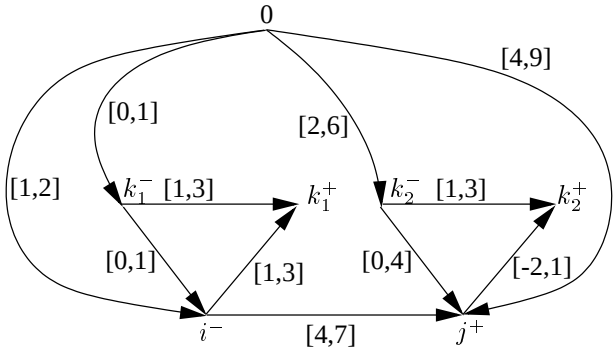


Figure 13: Illustrative example of Proposition 6

For the minimal consumption over the interval $\Delta = \underline{i}^-\overline{j}^+$, with the original formulation:

$$\underline{w}_k^\Delta = \max[0, \min(\underline{p}_k, t_2 - t_1, \underline{k}^+ - t_1, t_2 - \overline{k}^-)]$$

one gets $\underline{w}_{k_1}^\Delta = \max[0, \min(1, 9-1, 1-1, 9-1)] = 0$ and $\underline{w}_{k_2}^\Delta = \max[0, \min(1, 9-1, 3-1, 9-6)] = 1$. Over the interval $\Delta = i^-j^+$, with the new formulation

$$\underline{w}_k^\Delta = \max[0, \min(\underline{p}_k, v, c, e)]$$

one gets $\underline{w}_{k_1}^\Delta = \max[0, \min(1, 7, 1, 4)] = 1$ and $\underline{w}_{k_2}^\Delta = \max[0, \min(1, 7, 2, 0)] = 0$. Then if $\underline{p}_i = 4$ and $\underline{p}_j = 3$, with Proposition 5 the test $8 < 4 + 3 + 0 + 1$ does not permit any deduction, while rule 6 provides us with the test $7 < 4 + 3 + 1 + 0$ which implies $i \not\prec j$.

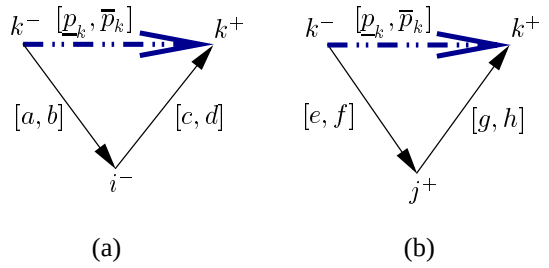
Adaptation to STNUs

We now suppose that k is contingent.

Minimal consumption The formulation of the minimal consumption is based upon the one in STNs with k not being contingent (formula (4)). For each k contingent, one must consider the upper duration $\overline{p}_k$ instead of $\underline{p}_k$ since, as said before, we should manage to infer a forbidden precedence if at least one situation forbids it, and since the consumption will always be higher with k taking its upper duration it is enough to consider the situation with $p_k = \overline{p}_k$. We hence get the following general formulation:

$$\underline{w}_k^\Delta = \max\{0, \min[\overline{p}_k, v, \max(c, \overline{p}_k - b), \max(e, \overline{p}_k - h)]\} \quad (6)$$

As in STNs, we will now try to simplify the two latter terms of this formula, i.e. for a task k shifted to the left and shifted to the right. Figure 14 shows which are the links of interests in these two cases.



(a) (b)

Figure 14: A left-shifted (a) and right-shifted (b) task

First, if only a classical PC algorithm was run on the network, then one would get in the former case $\overline{p}_k - c \geq b$ (otherwise as seen before b would have been updated). That amounts to $\overline{p}_k - b \geq c$, which allows us to simplify the term for a task shifted to the left, only retaining the stronger term $\overline{p}_k - b$. Similarly, for a task shifted to the right, $\overline{p}_k - e \geq h$ (otherwise h would have been updated by the PC algorithm), which amounts to $\overline{p}_k - h \geq e$.

This is actually enough to get the simplification, since 3DC+ will only restrict the constraint domains further and therefore can only make the inequalities stronger. Anyhow we will consider all cases of 3DC+ to show how the result holds. This algorithm indeed restricts the constraints in different ways according to the relative placement of i^-, j^+, and the ending points of k.

• **Task shifted to the left**

1. $d < 0$ (i^- after k^+).
   3DC+ does not make any specific restriction, therefore the former result simply holds. Moreover, in this case task k will necessarily be before i^-, thus outside the reference interval: since $c < d < 0$ the term $\overline{p}_k - c$ is greater and hence dominated by $\overline{p}_k$ in formula (6).
2. $c \geq 0$ (i^- comes before k^+).
   With 3DC+, one gets $b \leq \underline{p}_k - c$, thus $\overline{p}_k - b \geq \underline{p}_k - b \geq c$.
   In the example of figure 3 (where k^- stands for x, k^+ for y, and i^- for z), the initial network (a) cannot be updated by PC; one then gets $\overline{p}_k - b = 3 - 1 = 2$ and $c = 1$. In the final network (b), 3DC+ has restricted further value b. This allows us to deduce a higher necessary consumption since now $\overline{p}_k - b = 3 - 0 = 3$.
3. $c < 0$ and $d \geq 0$.
   With 3DC+, only a is further restricted. Adding a $wait$ on k^-i^- will also only affect the lower bound of k^-i^-, but the value of the $wait$ still must be lower than b. Therefore here as in case 1 of 3DC+ the former result $\overline{p}_k - b \geq c$ holds from PC updates.

• **Task shifted to the right**
We refer here to figure 14(b).

1. $h < 0$ (j^+ after k^+).
   Here again, 3DC+ provides no further restriction. Moreover, in this special case k will be fully contained within the interval Δ: since $h < 0$, the term $\overline{p}_k - h$ will be greater and hence dominated by $\overline{p}_k$ in formula (6).
2. $g \geq 0$ (j^+ before k^+).
   With 3DC+, one gets $e \geq \overline{p}_k - h$. With PC we had $e \leq \overline{p}_k - h$, which means necessarily $\overline{p}_k - h = e$. This is a property of 3DC+, that makes both terms equivalent here. We will choose $\overline{p}_k - h$ to remain consistent with other cases.
   Using the propagated network of figure 3 (where k^- stands for x, k^+ for y, and j^+ for z), one gets $e = 0$ and $\overline{p}_k - h = 3 - 3 = 0$.
3. $g < 0$ et $h \geq 0$.
   With 3DC+, one gets $e \leftarrow \max(e, \min(\overline{p}_k - h, \underline{p}_k))$, which amounts to two distinct cases:
   - If $\overline{p}_k - h \leq \underline{p}_k$ then $e \geq \overline{p}_k - h$. Since we had $e \leq \overline{p}_k - h$ from PC, we get $e = \overline{p}_k - h$: as for case 2, this is a special case where both terms could be used;
   - If $\overline{p}_k - h > \underline{p}_k$ then e might simply be updated with $\underline{p}_k$ which means $e \geq \underline{p}_k$, which as said before does not affect the result of PC: $\overline{p}_k - h \geq e$. The only interesting part is that one might get a $wait$ of $< k^+, \overline{p}_k - h >$ on k^-j^+. That means in the worst case the lower bound e will be increased up to $\overline{p}_k - h$ if k^+ does not occur before. Which still entails $\overline{p}_k - h \geq e$.

   In figure 4, on the network obtained by 3DC+ (b) (where k^- stands for x, k^+ for y, and j^+ for z), one gets $e = 1$ and $\overline{p}_k - h = 3 - 1 = 2$.

As a matter of conclusion, when k is contingent, the minimal consumption of k over a reference interval Δ is:

$$\underline{w}_k^{\Delta} = \max\{0, \min[\overline{p}_k, v, \overline{p}_k - b, \overline{p}_k - h]\} \qquad (7)$$

Extended forbidden precedence rule in an STNU The reference interval being $\Delta = i^-j^+ = [u, v]$, the rule is similar to Proposition 6, but the minimal consumption $\underline{w}_k^{i^-j^+}$ will be computed from equation (5) if k is not contingent and from equation (7) if k is contingent. Moreover, the terms $\underline{p}_i$ and $\underline{p}_j$ should be replaced by $\overline{p}_i$ if i is contingent and $\overline{p}_j$ if j is contingent.

Proposition 7 *i and j not contingent: If $v < \underline{p}_i + \underline{p}_j + \sum_{k \neq i,j} \underline{w}_k^{i^-j^+}$ then $i \not\prec j$.*

i and j contingent: If $v < \overline{p}_i + \overline{p}_j + \sum_{k \neq i,j} \underline{w}_k^{i^-j^+}$ then $i \not\prec j$.

i contingent and j not contingent: If $v < \overline{p}_i + \underline{p}_j + \sum_{k \neq i,j} \underline{w}_k^{i^-j^+}$ then $i \not\prec j$.

i not contingent and j contingent: If $v < \underline{p}_i + \overline{p}_j + \sum_{k \neq i,j} \underline{w}_k^{i^-j^+}$ then $i \not\prec j$.

We consider the same example as in figure 13, where now k_1 and k_2 are contingent. Figure 15 shows the network obtained after 3DC+ propagation.

With formula (7), it yields: $\sum_{k\in\{k_1,k_2\}} \underline{w}_k^{i^-j^+} = 3+2 = 5$. With Proposition 7, one consider the first case where i and j are both non-contingent. If $\underline{p}_i = \underline{p}_j = 2$, the test $7 < 2+2+5$ holds which implies $i \not\prec j$.

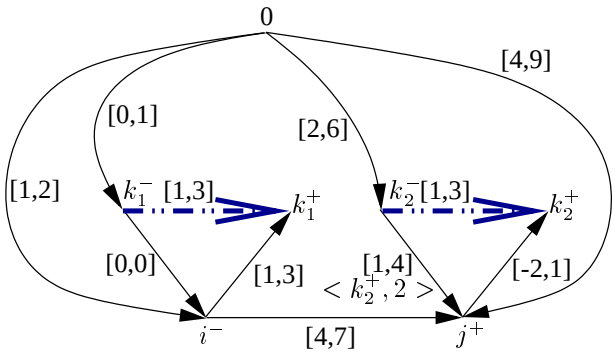


Figure 15: Illustrative example of Proposition 7

Concluding remarks

In this paper we have focused on two propagation rules that are known to be effective in task scheduling, when dealing with disjunctive resources. These are the Forbidden Precedence and the Extended Forbidden Precedence rules. First, using the strength of the STN minimal network, we modified the rules to take into account updated constraints between time-points instead of temporal domains, and showed the gain in terms of more deductions made. In the STNU framework, such deductions should be made more cautiously, since such rules guide the off-line scheduling process, and if a precedence has not been proven to be forbidden then it might be enforced in the network before execution. When temporal uncertainties and hence dynamic on-line execution strategies are to be considered, then one should forbid a precedence ($i \not\prec j$) as soon as there exists one situation in which the precedence is inconsistent. This made us considering worst cases with respect to contingent durations and change the rules accordingly.

These results are a first step in designing techniques to check off line the dynamic sequentiability of a task plan. The overall goal of these proactive reasoning techniques is to design predictive schedules that are not completely set, final decisions being taken on line. This added flexibility improves the robustness of the solution when facing time discrepancies from predicted task durations, and limits the need for elaborated (purely reactive) on-line scheduling, that would be more time consuming and could lead to troublesome deadends.

So the given rules might only prune the search and ease the on-line process by enforcing the precedence relations that are not forbidden. On the contrary, having $i \not\prec j$ and $j \not\prec i$ does not necessarily mean in STNUs that the problem has no solution; it only means there are some situations in which the former is forbidden while there are some situations (a priori others) in which the latter is forbidden. Therefore there is still work to be done to conclude the feasibility or not of the dynamic sequencing problem.

Anyway, we stronly believe that such incomplete rules are still very effective filtering techniques that will help the overall checking process. Therefore we plan to study the adaptation of more elaborated ones like edge-finding techniques, and evaluate these techniques on experimental examples.

References

Baptiste, P.; Le Pape, C.; and Nuijten, W. 2001. *Constraint-based scheduling*. Boston/Dordrecht/London: Kluwer Academic Publishers.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49:1–95.

Erschler, J.; Roubellat, F.; and Vernhes, J.-P. 1976. Finding some essential characteristics of the feasible solutions for a scheduling problem. *Operations Research* 24(4):774–783.

Esquirol, P.; Lopez, P.; and Huguet, M. 2001. Propagation de contraintes en ordonnancement. In Lopez, P., and Roubellat, F., eds., *Ordonnancement de la production*. Paris: Herms Science Publications. 131–167.

Laborie, P., and Ghallab, M. 1995. Planning with sharable constraints. In *Proc. of the 14th International Joint Conference on A.I. (IJCAI-95)*.

Lenstra, J.; Rinnooy Kan, A.; and Brucker, P. 1977. Complexity of machine scheduling problems. *Ann. Discrete Math.* 1:343–362.

Lopez, P., and Esquirol, P. 1996. Consistency enforcing in scheduling: A general formulation based on energetic reasoning. In *Proc. of the 5th International Workshop on Project Management and Scheduling*, 155–158.

Morris, P.; Muscettola, N.; and Tsamardinos, I. 1998. Reformulating temporal plans for efficient execution. In *Proc. of the International Conference on Principles of Knowledge Representation and Reasoning (KR-98)*.

Morris, P.; Muscettola, N.; and Vidal, T. 2001. Dynamic control of plans with temporal uncertainty. In *Proc. of the 17th International Joint Conference on A.I. (IJCAI-01)*.

Pinedo, M., and Chao, X. 1999. *Operations scheduling with applications in manufacturing and services*. Irwin/Mac Graw Hill.

Schwalb, E., and Dechter, R. 1997. Processing disjunctions in temporal constraint networks. *Artificial Intelligence* 93:29–61.

Torres, P., and Lopez, P. 2000. On not-first/not-last conditions in disjunctive scheduling. *European Journal of Operational Research* 127(2).

Vidal, T., and Bidot, J. 2001. Dynamic sequencing of tasks in simple temporal networks with uncertainty. In *Proceedings of CP'01 workshop on Constraints and Uncertainty*.

Vidal, T., and Fargier, H. 1999. Handling contingency in temporal constraint networks: from consistency to controllabilities. *Journal of Experimental & Theoretical Artificial Intelligence* 11:23–45.

IDEA: Planning at the Core of Autonomous Reactive Agents

Nicola Muscettola **Gregory A. Dorais** **Chuck Fry** **Richard Levinson** **Christian Plaunt**
NASA QSS NASA

NASA Ames Research Center
Moffett Field, California 94035
mus@email.arc.nasa.gov

Abstract

Several successful autonomous systems are separated into technologically diverse functional layers operating at different levels of abstraction. This diversity makes them difficult to implement and validate. In this paper, we present IDEA (Intelligent Distributed Execution Architecture) a unified planning and execution framework. In IDEA a layered system can be implemented as separate agents, one per layer, each representing its interactions with the world in a model. At all level, the model representation primitives and their semantics is the same. Moreover, each agent relies on a single model, plan database, plan runner and on a variety of planners, both reactive and deliberative. The framework allows the specification of agents that operate within a guaranteed reaction time and supports flexible specification of reactive vs. deliberative agent behavior. Within the IDEA framework we are working to fully duplicate the functionalities of the DS1 Remote Agent and extend it to domains of higher complexity than autonomous spacecraft control.

Introduction

Several successful autonomous systems are separated into technologically diverse functional layers operating at different levels of abstraction (Bonasso et al. 1997) (Currie and Tate 1991) (Wilkins et al. 1994). However, there are some significant drawbacks to this approach. Developing layered systems is complex. For example, it is unreasonable to expect that domain experts (e.g., system and mission engineers in a spacecraft domain) will directly encode their knowledge in a form usable by the different agent layers. Instead, this encoding becomes the responsibility of specialists familiar with each agent layer, which increases development cost and reduces the applicability of the autonomous software. Another problem is the frequent need to encode the same requirement in different forms in the different layers. The difficulty of tracking encoding discrepancies can decrease the reliability of the autonomous software. In this paper, we describe IDEA (Intelligent Distributed Execution Architecture) an approach to planning and execution that provides a unified representational and computational framework for an autonomous agent. IDEA provides a well-founded virtual machine that integrates planning as the reasoning module at the core of the execution engine. The virtual machine is composed by four main components whose interplay provides the basis for the agent's autonomous behavior: the domain model, the plan database, the plan runner, and the reactive planner. Deliberative planning is not a core requirement for the virtual machine but, through modeling and problem solving on the plan database, IDEA provides the means to program arbitrary combinations of reactive and deliberative problem solving. IDEA also defines a simple protocol for communication among several separate IDEA agents, i.e., agents implemented using the IDEA virtual machine. We believe that this representational and problem-solving approach can be applied at all levels of the architecture of a complex agent, such as Remote Agent (Bernard et al. 1998). We have recently taken a first significant step toward demonstrating this by re-implementing the high-level control layer of the Remote Agent. This includes closed-loop reactive planning after an unrecoverable hardware fault to put the spacecraft in standby while allowing the deliberative planner to regenerate the mission plan to adapt to the degraded spacecraft capabilities.

By defining a virtual machine IDEA aims at the agent's "assembly level". We believe that using IDEA is not incompatible with current high-level execution languages (Gat 1996) (Simmons and Apfelbaum 1998) since programs written in these languages could be compiled into IDEA's "assembler" and executed by an IDEA virtual machine. Moreover, IDEA aims at defining the required functionalities and interfaces of the modules constituting the virtual machine. As such, IDEA encourages the use of different technologies and implementations for the plan database and the reactive and deliberative planners (Kim, Williams and Abramson 2001).

In the rest of the paper we briefly describe the Remote Agent architecture as an example of the state of the art in multi-layered agents. We then describe how idea differs from current multi-layered architectures. We sketch the IDEA virtual machine and point out some of its implications, mainly with respect to the reactivity and interaction between reactive and deliberative decision-making.

Layered Agent Architectures: Remote Agent

The Remote Agent (RA) was developed at the NASA Ames Research Center and at the Jet Propulsion Laboratory. RA is an autonomous control system capable of closed-loop commanding of spacecraft and other complex systems. RA was demonstrated by running on-board the Deep Space 1 (DS1) spacecraft and controlling its operations for a total of two days in May 1999 (Bernard

et al. 1998) (Nayak et al. 1999). Unlike traditional spacecraft command sequencers, RA was designed to be goal-achieving and robust. While a command sequencer simply issues low-level commands at fixed times, a goal-achieving system receives a specified state to be maintained for a specified period of time and from this it determines the relevant commands and when to issue them. A command sequencer is brittle when confronted with command failures and cannot further proceed, but RA can modify pre-planned commands in order to overcome obstacles that would normally prevent the achievement of a goal. Operational constraints were explicitly encoded into RA models. RA used these models to avoid violating the constraints regardless of the commanded goals.

The RA architecture integrates three layers of functionality: a constraint-based planner/scheduler (PS) (Jonsson et al. 2000) a reactive executive (EXEC) (Pell et al. 1999), and a Model Identification and Recovery system (MIR) consisting of a model-based truth maintenance system with diagnosis and recovery module (Williams and Nayak 1996) (Figure 1). Each layer uses a different modeling language and a different way to specify problem-solving control.

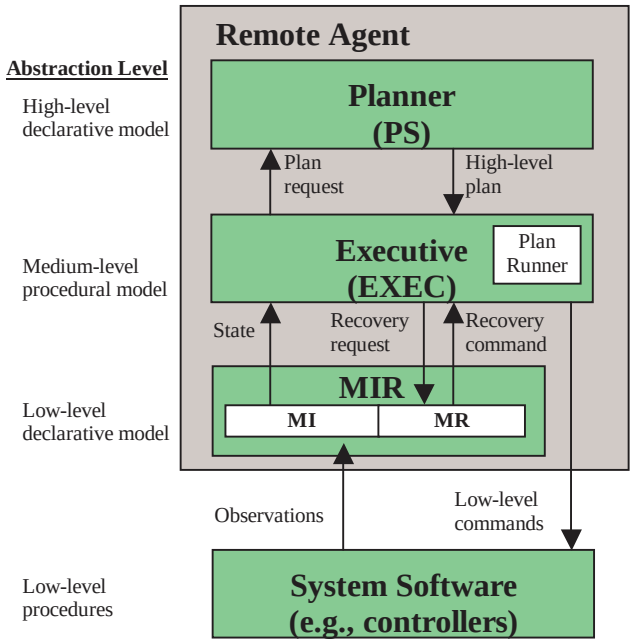


Figure 1: Remote Agent's layered architecture

At the highest level is PS, which uses a high-level declarative modeling language (HSTS DDL) to define the state machines and the temporal constraints needed to create valid plans. PS uses depth-first, backtrack search as the basic problem-solving engine. In order to produce plans in reasonable times, developers can use a simple language to specify choice selection heuristics. For the DS1 RA, we were able to write heuristics that drastically reduced backtracking, limiting it to shallow trees. This allowed PS' response time to stay between ½ hour and 4 hours depending on the size of the planning problem. This was achieved on a 20 MHz CPU for ½ of the available CPU time and within 32 MB of available memory.

The Executive (EXEC) occupies the second layer. EXEC's function is to translate high-level actions in the plan into a stream of timed, low-level commands to System Software. EXEC does so with two separate mechanisms. First it interprets the plan one step at a time with a specialized module called the plan runner. For each action currently in execution, the plan runner checks whether all logical and temporal termination conditions for the action are satisfied. If so, it terminates the action, it propagates the action termination time to the rest of the plan, and it starts the next action in the plan. When executing an action, EXEC runs a procedure associated with it in EXEC's model. Each procedure is written in ESL, an extension of LISP (Gat 1996). It specifies how to achieve the success states associated with the action using low-level commands to System Software. An ESL procedure operates at a level of abstraction higher than that of low-level commands in order to enhance reactivity. On DS1, an EXEC procedure needed to respond to any handled event within a worst-case 4-second bound.

EXEC relies on MIR to support low-level sensor interpretation and commanding. MIR provides two main functions. MI (Mode Identification) estimates state and notifies EXEC when a state changes. MI uses a detailed model of the system components (e.g., switches). Typically MI needs to consider interactions between several subsystems (e.g., sensors) in order to determine the state of some device (e.g., whether a thruster is ON or OFF). MR (Mode Recovery) uses the same model of MI and determines the least costly path from the MI estimated (faulty) system state and the one EXEC requires in order to satisfy the plan. MR also guarantees that the recovery actions do not pass through invalid states communicated by EXEC. For DS1, the maximum response time for MR was 5 seconds while MI could generally generate a diagnosis within a few hundred milliseconds

Using different problem solving modules with different representation languages and different reasoning engines had a direct advantage. In large part the modules constituting RA were based on technology already available. For DS1, it was therefore possible to concentrate on the still very hard problem of weaving these modules into a single, coherent agent. Also, one may argue that the representation and problem solving capability of each module could be tuned to maximize performance at that level. However, this heterogeneous approach made it difficult to validate all the models and procedures and to insure that they did not conflict.

The structure of IDEA

After an in depth analysis of RA's functionality, we believe that it is possible to duplicate it within a new, unified agent framework, where all layers have the same structure. In this section we give an outline of the main components in IDEA.

Tokens and Procedures

In IDEA, the fundamental unit of execution is a *token*, a time interval during which the agent executes a *procedure*. A procedure has the following general form:

$$P(i_1, \ldots, i_n \rightarrow m_{1, \ldots,} m_k \rightarrow o_1, \ldots, o_m;\ s)$$

Each i_i, m_i and o_i represents respectively an *input, mode* and *output* argument. It is possible for any or all of *n, k* and *m* to be zero. For example, if *n=0*, the procedure has no input arguments. A procedure has also a status value *s*. Normally, at any time during its execution, a procedure returns a value for each o_j. There are no constraints either on the order or on the exact time at which output values are returned. When the procedure returns a value for the status *s*, however, the token is terminated and one or more tokens may be started. To execute a procedure the value of all input arguments i_i must be known. If so, *P* can be called and the time of invocation of *P* is the token start time. The procedure continues execution until one of two things happen: 1) a status value is returned; or 2) the agent decides to interrupt the token's execution (e.g., because the token has timed-out, i.e., the current time is equal to the latest end time of the token). The time at which this happens is the token end time. While inputs, outputs and status play an active role in the execution of a token, the mode arguments play only an indirect role. Their value is not monitored at execution but can be arbitrarily modified by a planning activity at any time during the agent's problem solving.

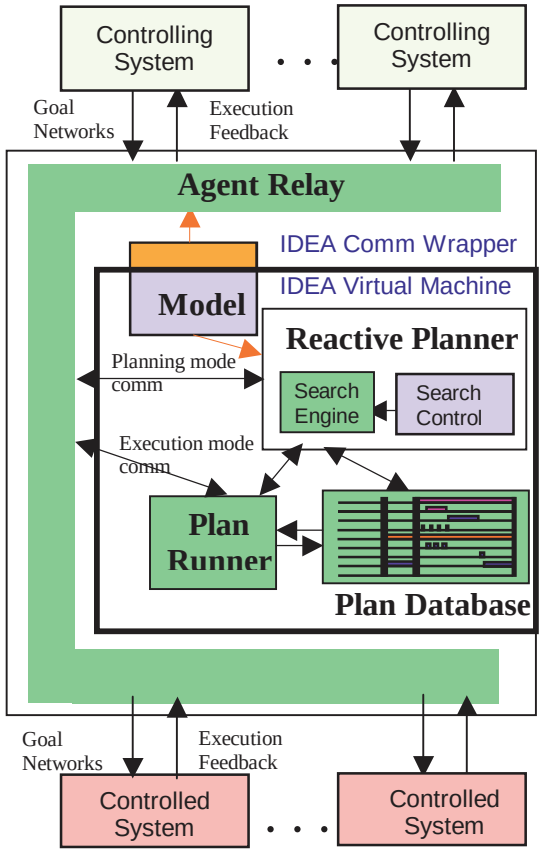


Figure 2: Structure of an IDEA agent

Communication Wrapper and Virtual Machine

Figure 2 gives an overview of the basic components of an IDEA agent. The agent communicates with other agents (either controlling or controlled by the agent) using a communication wrapper. The function of this wrapper is to send messages that initiate the execution of procedures by other agents or to receive goals that are treated by the agent as tokens. The arguments and the start and end time of each received token are treated as parameters used by the internal problem solving of the IDEA agent to decide what to do next. An IDEA agent can communicate with multiple agents both controlling and controlled. Moreover two agents could mutually control each other. Therefore, there is no restriction on the communication topology of a multi-agent system implemented with IDEA agents.

The format of the allowed communications is governed by the central *Model* that describes which procedures can be exchanged with which external agents. It also specifies which procedure arguments are expected to be determined before a goal is sent to another agent (input arguments) and on which arguments the agent is expecting execution feedback from some other agent executing the token (output and status arguments). As we will see this model is also central to the functioning of the virtual machine. To communicate with other agents the relay relies on an underlying inter-process communication mechanism that is not part of IDEA proper. Our current implementation relies on the IPC package from CMU (Simmons and James 2001) and we are also exploring the use of real-time CORBA (Real-Time CORBA 2002).

Plan Database and Model

The IDEA agent executes tokens only after they have appeared in a plan maintained in a central database. This can happen either because a controlling agent has communicated new goals or because some internal planning (reactive or deliberative) has generated appropriate subgoals. Although our reference implementation is based on the constraint-based EUROPA planning technology (Jonsson 2000), the use of different planning technologies is possible as long as they satisfy IDEA's requirement. In particular, the database must be partitioned into a series of parallel timelines, each representing the evolution over time of a dynamic property of a subsystem. To be considered for execution, a token must lay on an appropriate timeline. Sequences of tokens on a timeline will be executed sequentially and in parallel with tokens on other timelines. From now on we will continue discussing IDEA assuming the existence of a sophisticated constraint representation and propagation in the database, although this is not a strong requirement of IDEA.

At any point in time, the *Plan Database* describes the portion of the past that is remembered, the tokens currently in execution, and the currently known future tokens, including all the possible ways in which they can execute. Each token parameter (input, mode, output, status, and start and end time) has an associated variable. All these variables are connected by explicit constraints into a single constraint network. For example, the start and end time variables of each token are always related by an explicit duration constraint. The network implicitly restricts the possible value of each argument. The constraint database provides constraint propagation services that impose appropriate levels of consistency (e.g., arc consistency or path consistency) and can restrict

the range of variables to appropriate sets of values (possibly a singleton). For example, consider a simple case with two timelines, one representing the actions of a robot and the other representing the state of the robot's on-board battery. The plan may contain a robot action:

recharge ([10, 20]→ →; nominal)

This takes as input the level of charge of the battery, has no mode and output arguments, and is expected to return in a nominal state. The [10, 20] range means that the actual value of the input battery state of charge must be between 10 and 20 units for the procedure to be legally executed. The exact input value could be obtained by executing a token *read_state_of_charge (→ → soc; s*) on the battery state of charge timeline. Such a token could be present in the plan and constrained to execute before the recharge token. The communication of the state of charge between the two procedures can be obtained by a co-designation constraint between the output of *read_state_of_charge* and the input of *recharge*.

Tokens and constraints between them must respect the requirements of the agent's central domain Model. For example, the domain model could contain the constraint that before recharging the battery, it is necessary to *read_state_of_charge* from the battery. If this is the case, then *recharge* will not be executable unless such model constraint is satisfied in the plan at the time of execution.

Procedures can be executed only if the value of their parameters is consistent with the plan database constraints. The framework does not require that all database constraints be fully consistent at all times. It is possible to allow model constraints to be unsatisfied or for constraints to be inconsistent. The only consistency requirement is local and pertains to all the tokens that are currently being executed, about to be executed or that have already completed execution. This situation is similar to classic repair-based scheduling methods, where the scheduler can relax some constraints in the plan and attempt to satisfy them later. Since inconsistencies can only involve future tokens, the agent should have a reasonable belief that there will be a way to fix the inconsistency before the future tokens involved are executed. However the latter is not a strong requirement in this framework since usually it is possible to degrade performance by rejecting lower priority goals.

Generating and Running Plans

The core execution component of the agent is the *Plan Runner*, an extension and generalization of the RA plan runner. The plan runner is very simple so that it can be extremely efficient and easy to validate.

The Plan Runner is activated asynchronously when either a message has been received from another agent (e.g., a new goal is being communicated or the value of an output parameter becomes available for execution feedback) or an internal timer has gone off (e.g., the maximum allowable duration of a token has been achieved). When the Plan Runner wakes up, it makes the messages available for inclusion in the Plan Database and then immediately calls a *Reactive Planner*. The Reactive Planner has the responsibility to return with a plan that is locally executable. The planner is essentially in charge of guaranteeing two conditions: (1) consistency of token parameters with the plan constraints; and (2) support for the token according to the domain model.

Checking plan constraints is obtained as part of the constraint propagation within the Plan Database. This automatically communicates the effect of a received output or status value to the unexecuted part of the plan. Similarly, the actual end time for a token is propagated to the rest of the plan.

Checking model support for a token requires guaranteeing that a new token must start when the token immediately preceding on the timeline ends. Before starting a new token and invoking the new procedure, the Reactive Planner checks whether the token is indeed supported by the model in the plan. This means that there must be a set of constraints in the plan that corresponds to a set of requirements necessary for the token execution according to the model. If this is the case, the Reactive Planner may further constrain the procedure's arguments so that it can be called during the current execution cycle. This may require constraining the input variables so that all of the procedure's input arguments are bound to a single value. If so, the Plan Runner starts execution of the token procedure with the input variable found in the plan.

It may be that one of the two conditions above is not satisfied. This can happen, for example, if the output returned by a procedure does not match the set of possible return values in the plan, or if some model constraints are missing in the plan. For example, the plan runner may be on the verge of executing a *recharg*e token but the plan may not have an explicit constraint connecting recharge with a specific past *read_state_of_charge* token. In this case the Reactive Planner has the responsibility to fix the plan so that execution can continue. This may involve resolving execution exceptions (such as the missing constraint between *recharge* and *read_state_of_charge* described before) or refining future token parameters on the basis of the information received during the execution of current tokens (e.g., decide to execute a token as early as possible because of the value of some received output argument).

The total cycle time of the Plan Runner and Reactive Planner is bound by a fixed amount of time, the execution latency (Muscettola et al. 1998). The Plan Runner is expected to wake up, process all received messages, call the Reactive Planner, receive termination notification from the Reactive Planner, send appropriate messages to external agents and suspend itself within the execution latency. If this does not happen, then the agent will have irrecoverably failed and some low-level fault protection behavior will have to take over control. This hard requirement ensures that the agent will operate within a well-defined real time guarantee, a condition that is usually overlooked in intelligent agents research but is crucial to the design and implementation of a viable embedded control system.

Reactive and Deliberative Planning

IDEA allows the use of several planning modules in the same agent, each potentially using a different internal logic and working with a different scope. All of these modules satisfy the same input/output behavior: given an initial plan database, a planner generates a new plan

database that satisfies some given plan quality criterion (Jonsson et al. 2000). For example, the plan quality criterion may require that all tokens present in the initial plan database be present in the final plan and be fully causally supported. This may require removing inconsistencies present in the initial state, and generating new tokens and constraints according to the requirements of the domain model. A planner can be invoked in a reactive or proactive fashion. The first case occurs within the execution cycle of the Plan Runner, the second when the agent anticipates potential problems in the future and asks the planner to intervene. Deliberative planning can also be invoked to produce a high quality plan for a future horizon (e.g., an optimized observation plan for the next day), an activity that cannot be adequately carried out within the reactive execution latency. We will discuss later how this can be accomplished. Here we want to point out that there is no limitation on how small a planning problem could be, provided that the generated plan resolves any local plan flaw that was present in the plan before the invocation of the planner. For example, consider our example of an unsupported recharge token. The plan database may contain a previously executed token that invoked *read_state_of_charge*. On the basis of the domain model it may be determined that the result of that procedure invocation is still viable as an input to *recharge*. Therefore, the planner may simply create the temporal constraint and the parameter co-designation constraint from *read_state_of_charge* to *recharge*. Subsequent constraint propagation will assign a unique value for the input parameter of *recharge*. The plan quality criterion may allow the planner to stop and signal the resolution of the flaw. The Plan Runner can now resume execution and start execution of *recharge*.

Implications of the New Framework

Centrality of the model

The proposed framework strongly relies on a single, core domain model semantic. Unlike RA where models were internal to each layer and could have very different semantics, the common IDEA "modeling assembler" forces all agents to share the same semantics. At present, the modeling language used is the DDL language used in the PS model of the DS1 Remote Agent (Jonsson et al. 2000). Layering of the agent's functionality depends on partitioning the overall model into groups of timelines of different abstraction levels, each being the responsibility of a separate IDEA agent. For example, RA EXEC's action decomposition procedures are implemented by simply specifying an appropriate set of timelines and constraints in the model and by relying on fast, reactive planning for next action selection (see below). Partitioning a model among several agents is important to appropriately balance the responsiveness of each control agent with its ability of taking into account more complex constraints and longer horizons when deciding the next step. For example, a decision on what scientific observation to execute next at the highest level of abstraction may require looking ahead several steps in the current plan. This means that the reactive behavior at the higher level may require a relatively large execution latency (e.g., 10 seconds). At the lowest level, however, devices may have to be controlled with a much shorter latency (e.g., responding to a fault within tens of milliseconds). This may limit the amount of interactions and look-ahead that an agent will be allowed to take into account, trading off responsiveness for myopia. The coordination between different agents at different levels of abstraction allows us in principle to achieve the best compromise and design of the overall control system. Defining a robust methodology of the design of such a multi-agent, multi-latency control system is a current area of research.

In each agent, the plan database always checks consistency with the domain model. For example, a planner can lay a procedure invocation on a timeline only if the procedure type is associated with the timeline in the model. Also, the plan runner refuses to execute a token that is locally inconsistent or with partially supported model requirements.

The model can be acquired incrementally (i.e., one requirement at a time) during system design and engineering and at any time it contains all of the known constraints and desired behaviors in nominal and fault-protection conditions. Having the model as a single locus for this information and making the model directly usable by automatic reasoning systems (e.g., the planners) makes this knowledge directly usable at execution. This is in contrast to traditional software practices for complex systems (e.g., spacecraft flight software), where there is always a significant gap between specifications (in natural language or other semi-formal format) and implementations (a low-level language such as C or C++).

Reactivity

Even within a single IDEA agent, one important aspect is its *reactivity*, i.e., the time needed by the agent to decide what to do next in a way consistent with its predictions and with its goal. As we mentioned before, short response times depend on limiting the scope of the planning problem. Selecting the next action may require significant effort, requiring the intervention of a "deliberative planner" to bridge the gap between the current state and the goals. However, in general the amount of planning effort depends on the required level of plan quality (e.g., your next action must guarantee achievement of all future goals with minimal resource usage), on how much information is available before planning, and on the uncertainty on the values returned by procedure executions. In several cases the model may force the choice of the next action (e.g., turn on the heater if the temperature is too low) but the information needed to make the decision may not be available ahead of time (e.g., while the agent is keeping the temperature in range, it does not know future temperature changes and, therefore, whether it will need to turn on the heater or the cooler next). In this case planning may just need to determine the next token and, therefore, may need very little time. Later we will discuss how more expensive planning is integrated in the agent's behavior.

Time-bounded response

One of the critical parameters in this agent framework is the *execution latency*, i.e., the time needed by the plan runner to terminate execution of a token and start execution of the next on a timeline. At first this would appear to severely restrict the amount of intelligence that an agent can bring to bear when reacting to faults. If we look closer, however, this requirement simply states that a subsystem (timeline) can remain without commanding for a maximum amount of time equal to the latency. This requirement is equivalent to establishing a minimum sampling rate in a traditional control system. The agent can react intelligently by relying on a number of pre-compiled alternative solutions (scripts). When invoked, the planner could quickly select a script by matching its plan database with the script applicability conditions. Then, the planner could download the first token in the script and immediately signal the plan flaw resolution so that the plan runner can resume. Subsequently, the planner can download the remaining tokens in the script. This script interpretation (together with local replanning to react to new sensor data) essentially describes the functionality of the action execution capabilities of the RA EXEC module.

In some situations there may not be a planner (scripted or not) that can respond within the latency. In this case the system will need to provide a "standby procedure", i.e., a procedure or combination of procedures that maintains a safe state while the planner addresses the original plan flaw. Once the planner solves the problem, the system can exit the standby state and continue nominal execution. Note that the standby procedure, the planner behavior and the "standby exit" procedure are all described in the domain model and must be loaded into the plan database like any other procedure. In other words, standby is a concept that is explicitly modeled like any other system requirement. The planner will decide to go into standby within the latency time. This will gain enough time to take the next steps in a deliberate way.

Modeling the control system

Although a planner may need more time than the latency to modify the plan database, no special architectural support is given for this deliberative activity. For example, the agent may need to call the planner before a predicted plan flaw will actually appear in execution. This can be obtained by modeling the planner like any other subsystem, i.e., by specifying a timeline that can take tokens whose execution explicitly invokes the planner. The model may also include constraint requirements for "planned" planner invocations (Pell et al. 1997). For example the model may say how to evaluate the time needed by the planner to produce a solution, and it may require that planning does not occur in parallel with other CPU intensive activities. Proactive planner invocations will therefore appear in a plan. In summary, our framework does not "hard-wire" the relation between reactivity and deliberation but allows explicit programming of the interaction policy with a much wider and adjustable range of possibilities.

Final Remarks

It is commonly accepted that reactive and deliberative behaviors in an agent require very different representations and inference mechanisms. The framework discussed in this paper aims at providing both capabilities within a single, simple representational, planning and execution framework. This unification is based on the observation that "planning" can be arbitrarily simple for an appropriate definition of a planning problem. This can include the selection of the next action to execute from a script, a typical operation performed by procedural executives. IDEA aims at supporting all functionalities of the Remote Agent architecture. We have generated an implementation of IDEA using the EUROPA planning technology. We have re-implemented the high-level control layer of the Remote Agent and are currently applying IDEA to other applications such as the control of an interferometry testbed at the Jet Propulsion Laboratory and an analysis of the low-level fault protection system for the Deep Space 1 and Deep Impact spacecrafts from JPL.

References

D. E. Bernard, G. A. Dorais, C. Fry, E. B. Gamble Jr. B. Kanefsky, J. Kurien, W. Millar, N. Muscettola, P. P. Nayak, B. Pell, K. Rajan, N. Rouquette, B. Smith, B. C. Williams, "Design of the remote agent experiment for spacecraft autonomy." In *Proc. of the IEEE Aerospace Conference*, March 1998.

R. P. Bonasso, R. J. Firby, E. Gat, D. Kortenkamp, D. Miller, and M. Slack, "Experiences with an Architecture for Intelligent, Reactive Agents", *J. of Experimental and Theoretical AI*, January, 1997.

K. W. Currie, A. Tate. "O-Plan: the Open Planning Architecture," *AI Journal*, 52(1), pp. 49-86, 1991.

Erann Gat, "ESL: A language for supporting robust plan execution in embedded autonomous agents," Proceedings of the AAAI Fall Symposium on Plan Execution, AAAI Press, 1996.

A. Jonsson, and J. Frank, "A Framework for Dynamic Constraint Reasoning using Procedural Constraints, in Workshop on Constraints in Control, part of the 5th International Conference on Principles and Practices of Constraint Programming, (CP99), 1999.

A.K. Jonsson, P. Morris, N. Muscettola, K. Rajan, B. Smith "Planning in interplanetary space: theory and practice", in *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems* (AIPS '00), Breckenridge, Colorado, 2000.

Phil Kim, Brian Williams, Mark Abramson. Executing Reactive Model-based Programs Through Graph-based Temporal Planning, *Proc. of IJCAI 2001*, Seattle, WA, 2001.

N. Muscettola, P. Morris, B. Pell, B. Smith, "Issues in temporal reasoning for autonomous control systems." In Proc. of the Second Intl. Conf. on Auton. Agents (AGENTS'98), Minneapolis, MN, 1998.

P. P. Nayak, D. E. Bernard, G. Dorais, E. B. Gamble Jr., B. Kanefsky, J. Kurien, W. Millar, N. Muscettola, K.

Rajan, N. Rouquette, B. D. Smith, W. Taylor, Y. W. Tung, "Validating the DS1 Remote Agent Experiment", in *Proc. of the Fifth Intl. Symposium on Artificial Intelligence, Robotics and Automation n Space* (iSAIRAS'99), pp. 349, 356, Nordwijk, The Netherlands, June 1999.

Barney Pell, Erann Gat, Ron Keesing, Nicola Muscettola, and Ben Smith. "Robust Periodic Planning and Execution for Autonomous Spacecraft." In Proceedings of IJCAI, 1997.

Barney Pell, Ed Gamble, Erann Gat, Ron Keesing, Jim Kurien, Bill Millar, P. Pandurang Nayak, Christian Plaunt, and Brian Williams. "A Hybrid Procedural/Deductive Executive For Autonomous Spacecraft." In Autonomous Agents and Multi-Agent Systems, 2:1:7-22 1999.

Real-time CORBA with TAO (The ACE ORB), http://www.cs.wustl.edu/~schmidt/TAO-intro.html

Reid Simmons, David Apfelbaum. "A Task Descriprion Language for Robot Control", in *Proc. Conference on Intelligent Robotics and Systems, 1998.*

Reid Simmons, Dale James, *Inter-Process Communication v3.4*, Carnegie Mellon University, February 2001, available at http://www-2.cs.cmu.edu/afs/cs/project/TCA/ftp/icp.ps.gz

D. E. Wilkins, K. L. Myers, J. D. Lowrance, and L. P. Wesley, "Planning and Reacting in Uncertain and Dynamic Environments", in *Journal of Experimental and Theoretical Artificial Intelligence*, 6:197-227, 1994. See also: http://www.ai.sri.com/people/wilkins/papers.html

B. C. Williams, P. P. Nayak. "A Model-Based Approach to Reactive Self-Configuring Systems", in *Proc. of the Thirteen Nat. Conf. on Artificial Intelligence* (AAAI '96), Portland, Oregon, 1996.

Execution, monitoring and replanning in dynamic environments*

Oscar Sapena, Eva Onaindia
Dpto. Sistemas Informaticos y Computacion
Universidad Politecnica de Valencia
{osapena,onaindia}@dsic.upv.es

Abstract

In this paper we present SimPlanner, an integrated planning and execution-monitoring system. SimPlanner allows the user to monitor the execution of a plan, interrupt this monitoring process to introduce new information from the world and repair the plan to get it adapted to the new situation. SimPlanner has been successfully applied to small problems of mobile robots navigation in dynamic environments.

Introduction

Research on AI planning usually works under the asumption that the world is accessible, static and deterministic. However, in dynamic domains things do not always proceed as planned. Interleaving planning and executing brings many benefits as to be able to start the plan execution before it is completed or to incorporate information from the external world into the planner (Stone 1996). Recent works on this field analyse the combination of an execution system with techniques as plan synthesis or anticipated planning (Despouys & Ingrand 1999). Other works on reactive planning (Wilkins & Myers 1996) are more concerned with the design of planning architectures rather than exploiting the capabilities of the replanning process (Wilkins 1988).

SimPlanner is a planning simulator that allows the user to monitor the execution of a plan and introduce/delete information at any time during execution to emulate an external event. The issue of deciding when and how to interleave planning and execution is a well-recognized problem and it is not tackled in this paper (Ambros-Ingerson & Steel 1988). We will assume it is the user who decides when to execute a plan step.

SimPlanner is a domain-independent, synchronous replanner that repairs a plan when this is no longer executable after the occurrence of an unexpected event. Like other planning systems (Wilkins & Myers 1996), the objective of SimPlanner is to avoid generating a complete plan each time by retaining as much of the original plan as possible. SimPlanner has been specially designed for replanning in STRIPS-like domains and has been successfully applied to a great variety of different domains.

*This work has been partially supported by projects DPI2001-2094-C03-03 (MCyT), UPV n. 20010017 and UPV n. 20010980.

Monitoring the execution of a plan

Replanning is introduced during plan execution when an unexpected event occurs in the world. One problem with unexpected effects is deciding how they interact with the effects of the action that was currently being executed. Our solution is to assume the action took place as expected and simply to insert the unexpected effects after the execution of the action.

When an event is produced it is necessary to verify the overall plan is still executable. Many replanning systems only perform a precondition checking to verify whether next action is executable. This option is less costly and much easier to implement in many real applications where the sensory system does not capture all predicates that have changed in the problem. However, this apparently efficient approach may turn out to be inefficient in the long term as many unnecessary actions might be introduced due to changes in the plan are not foreseen enough time in advance. For instance, if a road is blocked off some meters ahead but it is still possible to move on to the next connection point, a mobile robot will make an unsuccessful movement as it will have to backtrack once it reaches the blocked road.

Our proposal is a fast and efficient algorithm that repairs the plan as a whole and finds the optimal solution (minimum number of actions) for most of the tested problems. Replanning is also necessary when new goals are added in the problem, but this issue is out of the scope of this paper.

Graphical interface

Figure 1 shows the graphical interface to monitor a plan execution. The problem corresponds to one of the instances in the robot domain which SimPlanner has been tested on (this domain will be explained later in section *An application example*). In the left upper part of the screen it is shown the literals of the current state of the execution; the lower section shows the literal goals or objectives of the final situation. On the right side of the screen a graph representing the plan under execution is displayed. The circles stand for the actions in the plan. Those actions ready to be executed at the next time step are double-circled. The right lower part displays information about the action selected by the user in the upper window.

At any time during the simulation it is possible to modify the current state to introduce new information from the

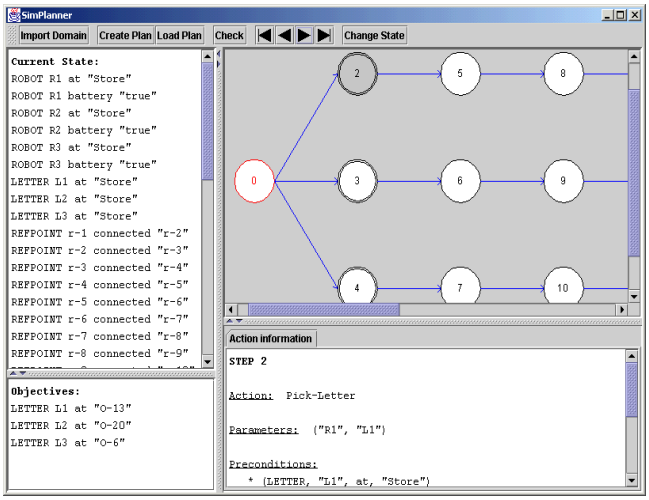


Figure 1: SimPlanner main interface window

external world. Through the interface shown in Figure 2 the user can eliminate those literals which are no longer true or insert new literals in the current state.

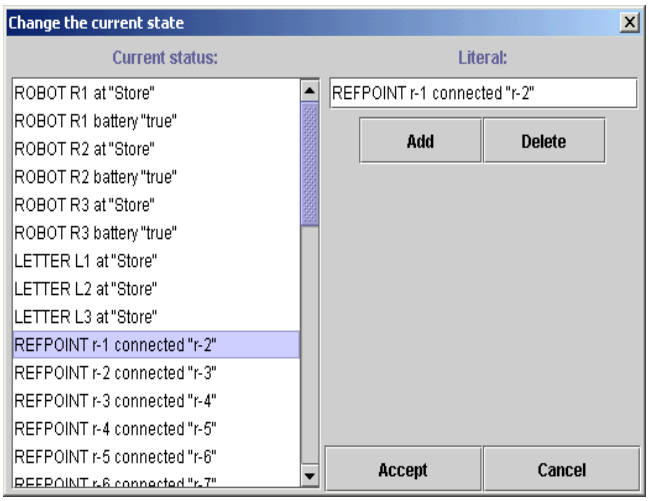


Figure 2: Interface for input information

Replanning during execution

The replanning algorithm starts from the current state in the plan execution, when the unexpected event has been input. The objective is to find out which state should be reached next in the problem so as to retain as much of the original plan as is reasonable without compromising the optimal solution. In other words, we want to compute that state from which as many actions as possible of the old plan are still applicable. Figure 3 shows the three main steps in the algorithm to obtain the new plan: 1) compute the possible reachable states, 2) select the optimal state and 3) construct the final plan.

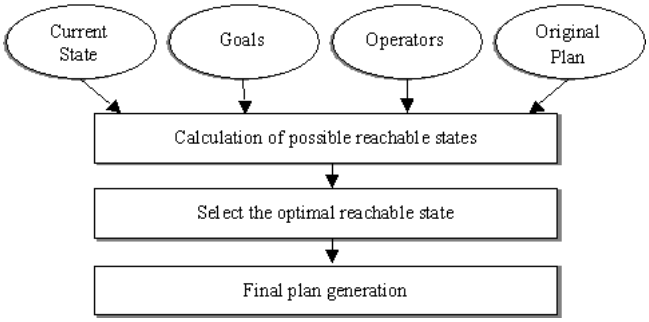


Figure 3: Outline of the replanning algorithm

Next section explains the algorithm in detail and the following one provides an example in the mobile robot domain to clarify SimPlanner behaviour.

SimPlanner algorithm

Initially, the user is monitoring the execution of a partial plan $P = (a_0 \rightarrow \{a_1 \rightarrow \ldots \rightarrow a_{n-1}\} \rightarrow a_n)$, that is parallel execution of a set of sequences of actions, when an unexpected event is introduced in the system. The remaining partial plan to be executed is defined as $\Pi = (a_0\prime \rightarrow \{\ldots\} \rightarrow a_n)/\forall a_i \in \Pi \rightarrow a_i \in P$ and a_i has not been executed yet. $a_0\prime$ represents the new current situation (an initial action with effects and no preconditions).

SimPlanner does not handle durative actions but allows parallel execution assuming the cost of each action is constant. The objective of SimPlanner is to minimize the overall number of actions and maintaining as maximum number of parallel actions as possible. Following we will explain each step of SimPlanner algorithm (Figure 4).

Problem Graph (PG). The first step of the algorithm is to build the PG, a graph inspired in a Graphplan-like expansion (Blum & Furst 1997). The PG may partially or totally encode the planning problem. The PG is a relaxed graph (delete effects are not considered) which alternates literal levels containing literal nodes and action levels containing action nodes.

The first level in the PG is the literal-level L_0 and it is formed by all literals in the initial situation $a_0{}'$. The PG creation terminates when a literal level containing all of the literals from the goal situation is reached in the graph or when no new actions can be applied. This type of relaxed graph is commonly used in many heuristic search planners (Haslum & Geffner 2000)(Hoffmann & Nebel 2001) as it allows to easily extract an approximate plan.

Necessary states. Second step is to compute the necessary state to execute each action in Π. A necessary state for an action a_i is the set of literals required to execute a_i and all its successors ($\mathrm{Succ}(a_i)$). In order to compute the necessary states, literals are propagated from the goal a_n to the corresponding action by means of the recursive formula shown in the algorithm (step 2 in Figure 4).

Set of possible reachable states. This set comprises the necessary states to execute a set of parallel actions $\{a_i\}$ and

Algorithm SimPlanner (Π) $\rightarrow$ plan Π'

1. Build a Problem Graph (**PG**) alternating literal levels and action levels $(L_0, A_1, L_1, A_2, ...)$, where:

$$A_j = \{a_j : \ \mathsf{Pre}(a_j) \in L_{j-1} \wedge a_j \notin A_i \ , i < j\}$$
$$L_j = L_{j-1} \cup \ \mathsf{Add}(a) \ \forall a \in \ A_j$$

2. Compute the necessary state, $S(a)$, for each $a \in \Pi$

$$S(a_i) = \begin{cases} \{l : l \in \mathsf{Pre}(a_n)\}, i = n \\ \bigcup_{a_j \in \mathsf{Next}(a_i)} S(a_j) \bigcup (\mathsf{Pre}(a_i) - \mathsf{Add}(a_i)) - \\ \qquad \bigcup_{a_k \in \mathsf{Paral}(a_i)} \mathsf{Add}(a_k), \text{ otherwise} \end{cases}$$

where:

$\mathsf{Next}(a_i) = \{a_j \in \Pi \wedge a_i \rightarrow a_j\}$
$\mathsf{Succ}(a_i) = \bigcup_j a_j \in \Pi / a_i \rightarrow \ldots \rightarrow a_j$
$\mathsf{Paral}(a_i) = \{a_j \in \Pi \wedge a_j \notin \mathsf{Succ}(a_i) \wedge a_i \notin \mathsf{Succ}(a_j)\}$

3. Compute the set of possible reachable states $RS = \{S(c_1), \ldots, S(c_n)\}$, where:

$$c_i = \{a_1, a_2, \ldots\} / \forall p \in \mathsf{Path}(\Pi) \rightarrow \exists! \ a_k \in p \wedge a_k \in c_i$$

$\mathsf{Path}(\Pi)$ is the set of all possible paths between $a_0{\prime}$ and a_n and $S(c_i) = \bigcup_{\forall a_j \in c_i} S(a_j)$.

4. Select the optimal reachable state $S_{opt} = argmin\{f(x) : x \in RS\}$, where $f(x) = g(x) + h(x)$.

5. Construction of the final plan $\Pi' = (a_0{\prime} \rightarrow \{\ldots\} \rightarrow S_{opt}) \otimes (S_{opt} \rightarrow \{\ldots\} \rightarrow a_n)$.

Figure 4: SimPlanner algorithm

all their successors $\mathsf{Succ}\{a_i\}$. A state will be definitely reachable if its literals make up a feasible situation in the new problem.

In a totally sequential plan Π, the possible reachable states will coincide with the necessary states to execute each action in Π. However, when Π comprises parallel actions it is necessary to compute the combinations of parallel actions, c_i, such that each element in c_i belongs to a different path from the current state to the goal state. In other words, c_i is a set of actions that can all be executed at the same time step. Consequently, the necessary state to execute actions in c_i denotes a state possibly reachable from $a_0{\prime}$.

Optimal state. In order to select the optimal reachable state, we define a heuristic function $f(x) = g(x) + h(x)$ associated to the cost of a minimal plan from the current situation $a_0{\prime}$ to the goal state a_n over all paths that are constrained to go through node x. $g(x)$ is the cost of the plan from x to a_n and is calculated straightforward from the original plan. $h(x)$ is the estimated length of an approximate plan P' from $a_0{\prime}$ to x. An outline of the algorithm to compute the approximate plan is shown in Figure 5.

Step 3.1. Firstly, we form a set $UP \subset L$ with the unsolved preconditions of the nearest action a to $a_0{\prime}$. If $|UP| > 1$ then literals which appear later in the PG are removed from UP. If again $|UP| > 1$ we count the number of ways of solving each $l \in UP$ ($\{a \in A_i : l \in L_j \wedge l \in \mathsf{Add}(a) \wedge i <= j\}$), and select the literal with the lowest number of actions.

Algorithm Approximate plan $(a_0{\prime}, x) \rightarrow$ plan P'

1. Build a fictitious action $a_n{\prime}$ with preconds and no effects associated to state x
2. $P' = a_0{\prime} \rightarrow a_n{\prime}$
2. $L = x - \mathsf{Add}(a_0{\prime})$.
3. while $L \neq 0$
   3.1 select $l \in L$
   3.2 select best a_j for l
   3.3 insert a_j in P'
   3.4 update L
4. return P'

Figure 5: Outline of the algorithm to build an approximate plan

Step 3.2. The best action for $l \in \mathsf{Pre}(a)$ will be the action a_j which minimizes the number of *flaws* (preconditions not yet solved or preconditions of other actions which are deleted by a_j). To compute the number of flaws we have to check all possible positions of a_j in the plan provided that $a_j < a$.

Step 3.3. The new action a_j is inserted in the position obtained in the previous step. This position may be sequential - between two actions- or parallel to one or more actions. In this latter case, it must be possible to execute all actions in parallel, i.e. none of the actions will require and Add effect of another action or delete any of its preconditions. When this is not possible, actions must be executed sequentially.

The length of the returned plan P' will be the value of the heuristic function $h(x)$. Some of the properties of the heuristic function are:

- if x is an inconsistent or non-reachable state (all literals in x cannot be true at the same time), $h(x)$ returns ∞,
- if x is reachable from $a_0{\prime}$ so will be the states following x. This helpful information reduces vastly the cost of computing $h(x)$,
- although $h(x)$ is a non-admissible heuristic, it returns the optimal state for most of the test cases in empirical evaluations.

An application example

In this section we show the correct behaviour of our replanning system through a problem in the mobile robots domain, the *mail delivery* problem. We want to solve the following problem: there are three letters **C1**, **C2** and **C3**, initially located in the store **S**, and three robots **R1**, **R2** and **R3** which are also at the store in the initial situation. Letter **C1** must be delivered to office **O-13**, **C2** to **O-20** and **C3** to **O-6**. The

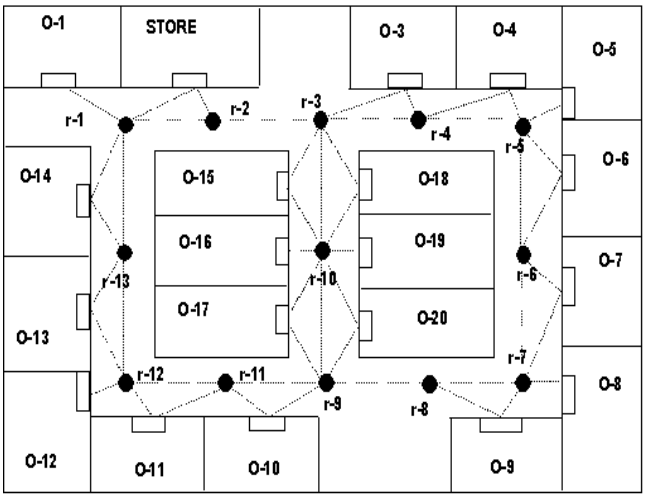


Figure 6: Robot domain

robots can only move through the reference points or from one office to a reference point and vice versa following the connection lines in Figure 6.

The operators in the domain are go_from $?x$ $?y$ $?r$ where $?r$ is a robot and $?x$ and $?y$ represent a reference point, an office or the store, pick_up $?c$ $?r$ and deliver $?c$ $?x$ $?r$, where $?c$ is one of the letters, $?r$ one robot and $?x$ one of the locations.

One of the optimal solutions (w.r.t. the overall number of actions) for this problem consists of three parallel sequences of ordered actions, each sequence being a the set of actions to be executed by one of the robots (Table 1, 2, 3). Another solution would be to send **R1** to transport **C1** and robot **R2** or **R3** to deliver firstly **C2** and **C3** afterwards.

Table 1: Optimal plan for the mail delivery problem (Robot 1)

Time	Sequence 1	
t1	a2	pick_up C1 R1
t2	a5	go_from S r_1 R1
t3	a8	go_from r_1 r_13 R1
t4	a11	go_from r_13 O_13 R1
t5	a14	deliver C1 O_13 R1

The new situation occurs after executing actions at time t2 when both R2 and R3 are situated at r_2 and R1 at r_1. At this time the user changes the current state and informs the planning system that the aisle connecting offices O_15, O_18, O_16 etc. through r_3 is blocked. Therefore, office O_20 is only accessible through the other extreme of the aisle, i.e through r_9. The remaining plan Π to be executed will comprise the set of actions of each sequence from t3 to t7.

Next step is to compute the necessary state for each action in Π. On doing this through the propagation process explained in the previous section, SimPlanner detects that

Table 2: Optimal plan for the mail delivery problem (Robot 2)

Time	Sequence 2	
t1	a3	pick_up C2 R2
t2	a6	go_from S r_2 R2
t3	a9	go_from r_2 r_3 R2
t4	a12	go_from r_3 r_10 R2
t5	a15	go_from r_10 O_20 R2
t6	a17	deliver C2 O_20 R2

Table 3: Optimal plan for the mail delivery problem (Robot 3)

Time	Sequence 3	
t1	a4	pick_up C3 R3
t2	a7	go_from S r_2 R3
t3	a10	go_from r_2 r_3 R3
t4	a13	go_from r_3 r_4 R3
t5	a16	go r_4 r_5 R3
t6	a18	go r_5 O_6 R3
t7	a19	deliver C3 O_6 R3

action a_{12} is not executable because it is not possible to go from r_3 to r_{10}. Therefore, there is no need to compute $S(a_9)$ because a_{12} is a successor of a_9. Notice that a necessary state $S(a_i)$ is the set of literals to execute action a_i and all its successors; consequently, there is no feasible necessary state for a_9 and none of its predecessor actions since the part of the plan from a_3 up to a_{12} is not reusable any more. This determines that SimPlanner does not reuse any parts of a plan that come before a failed action.

Once we know a_9 and a_{12} are not executable and, taking into account that Π contains parallel sequences of actions, we build all possible combinations of parallel actions to form the reachable states. These combinations will be $\{S(a_8) \cup S(a_{15}) \cup S(a_{10})\}$, $\{S(a_8) \cup S(a_{15}) \cup S(a_{13})\}$, ..., $\{S(a_8) \cup S(a_{17}) \cup S(a_{10})\}$, etc. Then we apply the evaluation function to each reachable state. The results are shown in Table 4.

For state RS1 the heuristic value is 6, which is the number of actions to transform the current state into a state where R1 is at r_1 ($S(a_8)$), R3 is at r_2 ($S(a_{10})$), and R2 is at r_10 ($S(a_{15})$). Notice that it is not necessary to apply any action to reach $S(a_8)$ or $S(a_{10})$, and the minimal length plan to reach $S(a_{15})$ is 6.

Values for states RS2, RS3, ... are not shown in the table as these nodes are pruned when the computed value is greater than $f(\text{RS1})$. Next best value is $f(\text{RS6})$. The heuristic function also returns a value of 6 since this is the number of actions to deliver C2 in O_20 from the current state. However, the value of $g(x)$, length of the plan to reach the goal state, is shorter than for $f(\text{RS1})$ and consequently this is the optimal reachable state.

Notice that in this example $h(x)$ always returns a value equal to $h^\star(x)$ (cost of optimal path), which is not always

Table 4: Evaluation of reachable states ($\star$ means the node is pruned)

	Reachable state	$h(x)$	$g(x)$	$f(x)$
RS1	$S(a8) \cup S(a15) \cup S(a10)$	6	10	16
RS2	$S(a8) \cup S(a15) \cup S(a13)$	$\star$	9	∞
RS3	$S(a8) \cup S(a15) \cup S(a16)$	$\star$	8	∞
RS4	$S(a8) \cup S(a15) \cup S(a18)$	$\star$	7	∞
RS5	$S(a8) \cup S(a15) \cup S(a19)$	$\star$	6	∞
RS6	$S(a8) \cup S(a17) \cup S(a10)$	**6**	**9**	**15**
RS7	$S(a8) \cup S(a17) \cup S(a13)$	$\star$	8	∞
RS8	$S(a8) \cup S(a17) \cup S(a16)$	$\star$	7	∞
.....	....	...	...	...
RS_i	$S(a11) \cup S(a15) \cup S(a10)$	$\star$	9	∞
.....	....	...	...	...
RS_j	$S(a11) \cup S(a17) \cup S(a10)$	$\star$	8	∞
.....	....	...	...	...
RS_k	$S(a14) \cup S(a15) \cup S(a10)$	$\star$	8	∞
.....	....	...	...	...
RS_l	$S(a14) \cup S(a17) \cup S(a10)$	$\star$	7	∞
.....	....	...	...	...
RS_n	$S(a_n)$	$\star$	0	∞

the case. Moreover, the other feasible solution for R2 to deliver letter C2 through r_3, r_4, r_5, r_6, r_7, r_8 and r_9 has not even been calculated. This is because the Problem Graph finishes at the first literal level at which the goal literals appear and the approximate plan found from this level corresponds to the minimal length plan.

Finally, a plan from $a0'$ to the best reachable state $RS6$ is generated. This plan (Table 5) is composed of six actions and it is the optimal solution. This plan will be added to the remaining plan Π thus given rise to the plan shown in Figure 7.

In order to build the plan shown in Table 5, we have developed a planning algorithm that uses the same data structures and heuristic evaluation than SimPlanner . The planner has been designed to return a plan action by action, starting from the first executable action. Currently, we are working on the full integration of the planning and replanning algorithms to get planning and execution running concurrently.

The plan obtained when computing $h(x)$ is used as an upper bound to the planner. For most of the test cases, the plan returned by the planner was the same as the obtained in the computation of $h(x)$.

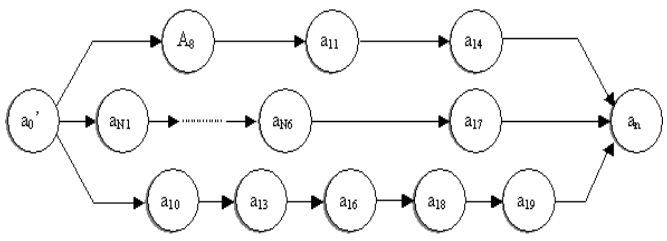


Figure 7: Final plan for the robot domain

Table 5: Optimal plan to reach $RS6$

$a_0\prime$	current state
a_{N1}	go_from r_2 r_1 R2
a_{N2}	go_from r_1 r_13 R2
a_{N3}	go_from r_13 r_12 R2
a_{N4}	go_from r_12 r_11 R2
a_{N5}	go_from r_11 r_9 R2
a_{N6}	go_from r_9 O_20 R2
$a_n\prime$	optimal reachable state

Experimental Results

SimPlanner has been tested in several domains with different type of input information about the current state. The tested domains are Hanoi, Monkey, Blocks-world, Logistics and Mobile robots navigation. For each of these domains we have introduced several unexpected changes:

- *Hanoi*: the appearance of new disks and modifications in the location of the existing disks
- *Monkey*: changes in the location of several objects (the monkey, the bananas, the knives, the boxes, etc.)
- *Blocks-world*: same kind of modifications as in the *Hanoi* domain,
- *Logistics*: we introduced unexpected events as breakdowns in planes. In this case it is necessary to replan some of the routes as there are planes that are no longer available.
- *Mobile robots navigation*: we have tested several situations like blocking off the path followed by a robot or using up the battery power of a robot.

In all cases, the obtained plan was the optimal one. Figure 8 shows the comparative times for seven different problems between generating a complete plan from scratch or repairing only the affected parts of the plan (replanning process). Temporal cost for replanning includes the cost of computing the reachable state plus time for generating the plan. The less meaningful the changes are the greater the saving time is (in figure 8, 'slight changes' mean modifications that only affect a small part of the plan whereas 'major changes' are those which affect the most part of the plan). In problem **P7**, the new current state forces to create a completely new plan so the cost of replanning is slightly higher. The complexity of the tested problems is not very high (for instance, examples up to seventeen blocks were tested in *blocks-world* domain).

We have also made the same tests with planner STAN2000 (Fox & Long 1999) (Bacchus 2000). The time difference between planning and replanning is about the same proportion as the results shown in Figure 8.

We have not considered here the quality of plans resulting from replanning versus planning from scratch. However, we can affirm that in case $h(x)$ returns the optimal value the quality of a completely new plan - in terms of number of actions- is not greater than the obtained with the replanning process.

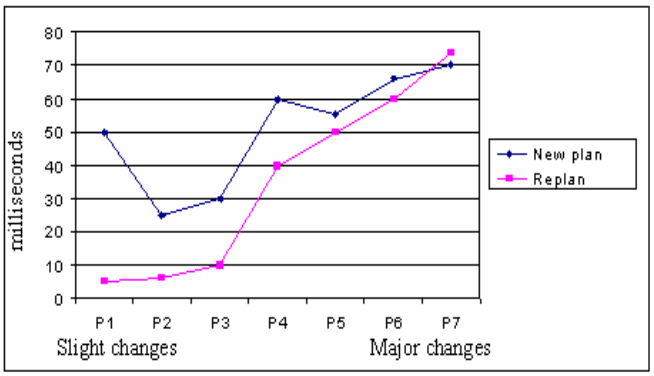


Figure 8: Comparative results: generating a complete plan versus replanning

Conclusions

SimPlanner is a planning and execution system which allows the user to monitor the execution of a plan, interrupt the monitoring to input new information and repair the plan under execution when an unexpected event is input. SimPlanner performs an execution monitoring rather than simply testing the next action to execute. This way, SimPlanner anticipates forthcoming situations and adjusts the plan in accordance.

The key point in SimPlanner is the replanning module. SimPlanner uses a graph-based planning approach supported by heuristic search techniques to efficiently replan in dynamic environments. Although we cannot guarantee $h(x)$ is an admissible heuristic function, the obtained results are always optimal or close to the optimal solution. SimPlanner is also able to compute a very fast solution, one of the most important requirements in replanning systems.

Along with the replanning module, we have developed an heuristic planner which builds an executable plan action by action. Our next goal is to integrate both algorithms in SimPlanner so as to obtain a single system for planning and execution concurrently. Additionally, we intend to extend SimPlanner to deal with time and consumable resources and obtain optimal responses in terms of distances, fuel consumption, time, etc.

References

Ambros-Ingerson, J., and Steel, S. 1988. Integrating planning, execution and monitoring. In *Proceedings AAAI-88, 83–98*.

Bacchus, F. 2000. Aips 2000 competition results. Technical report. Available at http://www.cs.toronto.edu/aips2000/.

Blum, A., and Furst, M. 1997. Fast planning through planning graph analysis. *Artificial Intelligence* 90:281–300.

Despouys, O., and Ingrand, F. 1999. Propice-plan: Toward a unified framework for planning and execution. In *Proc. European Conference on Planning (ECP-99)*, 280–292.

Fox, M., and Long, D. 1999. Stan public source code. Available at http://www.dur.ac.uk/CompSci/research/stanstuff/.

Haslum, P., and Geffner, H. 2000. Admissible heuristics for optimal planning. In *International Conference on AI Planning and Scheduling, 2000*.

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

Stone, P.and Veloso, M. 1996. User-guided interleaving of planning and execution. In Press, I., ed., *Frontiers in Artificial Intelligence and Applications*. 103–112.

Wilkins, D., and Myers, K. 1996. Asynchronous dynamic replanning in a multiagent planning architecture. In *Advanced Planning Technology*. 267–274.

Wilkins, D. 1988. *Practical Planning: Exending the Classical AI Planning Paradigm*. Morgan Kaufmann Publishers, San Mateo, CA, 1988.

Explanation-based repair techniques for solving dynamic scheduling problems

Abdallah Elkhyari[1] and **Christelle Guéret**[1,2] and **Narendra Jussien**[1]
[1] École des Mines de Nantes – BP 20722 – F-44307 Nantes Cedex 3 – France
[2] IRCCyN – Institut de Recherche en Communications et Cybernétique de Nantes
email: {aelkhyar, gueret, jussien}@emn.fr
Keywords: Dynamic scheduling, repair-based techniques, explanation-based constraint programming, Resource-Constrained Project Scheduling Problem.

Introduction

Scheduling problems have been studied a lot over the last decade. Due to the complexity and the variety of such problems, most works consider static problems in which activities are known in advance and constraints are fixed. However, every scheduling problem is subject to unexpected events (consider for example a new activity to schedule, or a machine breakdown). In these cases, a new solution is needed in a preferably short time taking these events into account and as close as possible to the current solution.

In this paper, we present an exact approach for solving dynamic scheduling problems. This approach uses explanation-based constraint programming and operational research techniques. Our tools have been designed for a general scheduling problem: the Resource-Constrained Project Scheduling Problem (RCPSP).

Problem description

The RCPSP can be defined as follows: let $A = \{1, \ldots, n\}$ be a set of activities, and $R = \{1, \ldots, r\}$ a set of renewable resources. Each resource k is available in a constant amount R_k. Each activity i has a duration p_i and requires a constant amount r_{ik} of the resource k during its execution. Preemption is not allowed. Activities are related by precedence constraints, and resource constraints require that for each period of time and for each resource, the total demand of resource does not exceed the resource capacity. The objective considered here is to find a solution for which the end of the schedule is minimized. This problem, denoted by $\mathtt{PS/prec/C}_{max}$ (Brucker *et al.* 1999), is *NP-hard* (Blazewicz, Lenstra, & Rinnoy Kan 1983).

The static RCPSP has been extensively studied (Brucker *et al.* 1998). A major difficulty in this problem is to maintain the resource limitation over the horizon time. Several deduction rules exist: *core-times* (Klein & Scholl 1999), *energetic-reasoning* (Erschler & Lopez 1990), *task-interval* (Caseau & Laburthe 1996), etc.

The dynamic RCPSP is seldom studied. Two classical methods are used to solve it:

- recomputing a new schedule each time an event occurs. This is quite time consuming and may lead to a solution quite different from the previous one.
- building a partial schedule and completing it progressively as time goes by (like in on-line scheduling problems). In this case the schedule cannot be constructed in advance.

Recently, (Artigues & Roubellat 2000) introduced a formulation of the RCPSP based on a flow network model. They developed a polynomial algorithm based on this model in order to be able to insert an unexpected activity.

Explanation-based constraint programming

Constraint programming techniques have been widely used to solve scheduling problems (Klein 1999). Constraint programming is based upon the notion of *constraint satisfaction problems*.

A *constraint satisfaction problem* (CSP) consists in a set V of variables defined by a corresponding set of possible values (the domains D) and a set C of constraints. A solution for the CSP is an assignment of the variable such that all the constraints are satisfied.

Explanation-based constraint programming (*e-constraints*) has already proved its interest in many applications (Jussien 2001). This section recalls what is an explanation and how it can be used.

Explanations

In the following, we consider a CSP (V, D, C). Decisions (variable assignments) made during the enumeration phase of the resolution of this problem correspond to adding or removing constraints from the current constraint system (*eg.*, upon backtracking).

A **conflict set** (*a.k.a.* **nogood**) is a subset of the current constraints system of the problem that, left alone, leads to a contradiction (no feasible solution contains a conflict set). A conflict set divides into two parts: a subset of the original set of constraints ($C' \subset C$) and a subset of decision constraints introduced so far in the search: $\neg (C' \wedge v_1 = a_1 \wedge \cdots \wedge v_k = a_k)$.

In a conflict set composed of at least one decision constraint, a variable v_j is selected and the previous formula can be rewritten as[1]: $C' \wedge \bigwedge_{i \in [1..k] \setminus j} (v_i = a_i) \Longrightarrow v_j \neq a_j$

[1] A conflict set that does not contain such a constraint denotes an over-constrained problem.

The left hand side of the implication constitutes an **eliminating explanation** for the removal of value a_j from the domain of variable v_j and is noted $\texttt{expl}(v_j \neq a_j)$.

Classical CSP solvers use domain-reduction techniques (removal of values). Recording eliminating explanations is sufficient to compute conflict sets. Indeed, a contradiction is identified when the domain of a variable v_j is emptied. A conflict set can easily be computed from the eliminating explanations associated with each removed value: $\neg\left(\bigwedge_{a \in D_{v_j}} \texttt{expl}(v_j \neq a)\right)$

There exist generally several eliminating explanations for the removal of a given value. Recording all of them leads to an exponential space complexity. Another technique relies on *forgetting* (erasing) eliminating explanations that are no longer relevant[2] in the current variable assignment. By doing so, the space complexity remains polynomial. We keep only **one** explanation at a time for a value removal.

Using explanations

Explanations can be used in several ways (Jussien 2001). For example, when debugging, explanations can be used to: **clearly** explain failures, explain differences between intended and observed behavior for a given problem (why is value n not assigned to variable x ?).

Explanations can also be used to determine direct or indirect effects of a given constraint on the domains of the variables of the problem, and for dynamic constraint removal. This is the case with the justification system used in (Bessière 1991) for solving dynamic CSP. This justification system is actually a partial explanation system. Moreover, being able to explain failure and to dynamically remove a constraint facilitates the building of dynamic over-constrained problem solvers.

We added explanation handling within a branch and bound algorithm in order to provide a dynamic RCPSP problem solver.

Solving dynamic RCPSP

We developed an environment for solving dynamic RCPSP that is based upon:

- a branch and bound algorithm (inspired from (Brucker *et al.* 1998)) within a constraint programming solver: at each node, deduction rules are applied in order to determine redundant information. In the following, we call *constraint* each initial constraint of the problem (precedence and resource constraints), but also each decision taken by the branching scheme, and each deduction made (thanks to propagation rules) during the search.
- an extensive use of *explanations*. Explanations are recorded during the search, and improved thanks to propagation rules (namely *core-times* and *task-interval*) which have been upgraded in order to provide a precise explanation for every deduction made.

[2] An explanation is said to be relevant if all the decision constraints in it are still valid in the current search state.

Our environment can efficiently handle dynamic events. Indeed, an unexpected event leads to add, modify or remove a constraint in the system. In the first two cases, if the current solution is no more valid, then the explanations tell us which are the constraints responsible of the contradiction. Repairing is done by removing at least one constraint from the explanation. The resulting solution is generally quite similar to the previous one, and is found faster than if we have had to solve the problem from scratch.

Notice that, as we saw before, if the constraints of the explanation are only initial constraints of the problem, then the problem is over-constrained. In this case, our system explains to the user why the problem has no solution. Responsible constraints can therefore be relaxed.

Conclusion

We developed a dynamic environment for solving general scheduling problems which can let the user interact with his/her problem: define a new time-window for an activity, add/remove precedence constraints, add/remove *must-overlap* constraints (*i.e.* stating that two activities must overlap), define/reduce/remove a required time-lag between two activities, etc.

We are currently experimenting our tools in order to show the interest of a real dynamic approach compared to re-execution from scratch. We do expect promising results.

References

Artigues, C., and Roubellat, F. 2000. A polynomial activity insertion algorithm in a multi-resource schedule with cumulative constraints and multiple modes. *European Journal of Operational Research* 127(2):297–316.

Bessière, C. 1991. Arc consistency in dynamic constraint satisfaction problems. In *Proceedings AAAI'91*.

Blazewicz, J.; Lenstra, J.; and Rinnoy Kan, A. 1983. Scheduling projects subject to resource constraints: classification and complexity. *Discrete Applied Mathematics* 5:11–24.

Brucker, P.; Knust, S.; Schoo, A.; and Thiele, O. 1998. A branch and bound algorithm for the Resource-Constrained Project Scheduling Problem. *European Journal of Operational Research* 107:272–288.

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

Caseau, Y., and Laburthe, F. 1996. Cumulative scheduling with task-intervals. In *Joint International Conference and Symposium on Logic Programming (JICSLP)*.

Erschler, J., and Lopez, P. 1990. Energy-based approach for task scheduling under time and resource-constraints. In *Second international workshop on Project Management and Scheduling (PMS)*, 115–121.

Jussien, N. 2001. e-constraints: explanation-based constraint programming. In *CP01 Workshop on User-Interaction in Constraint Satisfaction*.

Klein, R., and Scholl, A. 1999. Computing lower bounds by destructive improvement: an application to Resource-Constrained Project Scheduling Problem. *European Journal of Operational Research* 112:322–345.

Klein, R. 1999. *Scheduling of Resource Constraints Projects*. Boston: Kluwer Academic.

A constraint optimization framework for real-time applications

Simon de Givry, Philippe Gérard, Laurent Jeannin,
Juliette Mattioli, Nicolas Museux, Pierre Savéant

THALES Research & Technology
Domaine de Corbeville 91404 Orsay cedex France
simon.degivry@thalesgroup.com

This position paper presents the constraint technology that has been developed[1] at THALES since 1997 for introducing Constraint Programming (CP) in THALES operational systems (see [Givry.et.al01a] for a longer presentation). These systems involve combinatorial optimization problems such as planning and scheduling problems that can be expressed with finite-domain variables and constraints. Typical examples of THALES systems concern supervision, for weapon allocation, radar configuration, weapon deployment and aircraft sequencing. All these systems are subject to specific requirements coming from the operational constraints of embedded real-time systems and from the strategic context of Defense applications:

- The system involves several functions/tasks such as situation assessment, resource management, visualization, etc.; each task is periodical and the period can be much shorter than a second;
- There is a memory space limit (a few megabytes);
- The system has to be supported for a long time, typically over 20 years for Defense applications, including several retrofitting (functional and platform evolutions);
- The system can be reused and modified for building a specific system for a new client (product line);
- The development of the system must be made and mastered in house for reasons of confidentiality and market protection.

The CP paradigm partially meets these requirements. A constraint model has modularity properties, i.e. adding/removing a constraint is easy, which enables an incremental development process, reducing the development time and effort. CP solvers provide efficient algorithms through the use of global constraints. The declarative nature of CP enables the programmer to focus on the application requirements rather than on debugging low-level programming errors. Validated CP models can be reused in a product line approach.
Unfortunately, off-the-shelf CP solvers do not provide any guarantee on time and space usage. The classical backtracking search algorithm used in CP does not take into account any time contract. Recently an effort was made to provide better search algorithms in CP solvers, for instance in [Beldiceanu.et.al98,Laburthe98,Perron99], but without any explicit time contract. Our aim is to extend CP solver with new search features that would keep the same nice software engineering properties as for modeling. This led to develop a high-level language for designing search algorithms. This approach allows to propose a set of search primitives on top of the real-time finite-domain constraint solver Eclair© [Laburthe.et.al98,Platon01]. The resulting search algorithms are based on partial search methods and take into account the time contract explicitly. Such algorithms can take advantage better of platform evolutions.

Eclair offers time and space guarantees. Deadlines are guaranteed by the operating system alarm and Eclair is able to restore a coherent state after an interruption in order to deliver a valid solution, or just a partial solution (when not all variables are instantiated). The memory allocation for the constraints is static: a global constraint model is built once and only parts of the model are made active and used at a given cyclical call. The memory consumed during the search is limited by using only restricted depth-first search or restricted best-first search.

Partial search methods are anytime algorithms [Zilberstein96] based on tree search methods having better quality profiles than the classical backtracking search algorithm. The main idea is to apply some arbitrary limits on the nodes visited in the tree search[2], depending on the behavior of the heuristics and on the remaining computation time. We distinguish four approaches: the iterative weakening methods (e.g. [Harvey&Ginsberg95]), the real-time search methods (e.g. [Korf90]), the iterative sampling methods (e.g. [Gomes.et.al98]) and the interleaving methods (e.g. [Meseguer97]). These methods use one or several *search schemes*[3]. The practical complexity of the search can be increasing, self-adjusting, or stable. In [Givry.et.al99], we propose the notion of *parameterized search* applied to one search scheme. The

[1] This work is partially funded by the EOLE project [Eole01].

[2] This description of partial search is compatible with the depth-first search principle. In [Perron99], partial search methods are based on the order of node exploration, which is memory consuming.

[3] A search scheme is a procedure which describes a search tree. For example, a combination of choice points.

parameters of the search limits are given explicitly. We can tune the degree of incompleteness of the search by varying the values of the parameters. A *tuning policy* indicates the relevant values of the parameters for different time contracts. In [Givry.et.al01b], we integrate the parameterized search approach into a *hybridization scheme* to express partial search based on several search schemes. The hybridization scheme is a sequence or an interleaving of parameterized searches. The searches can cooperate by exchanging solutions. A *time-sharing policy* specifies how to distribute the time contract to the searches.

Our constraint optimization framework is called ToOLS© (Templates Of On-Line Search). A search algorithm is expressed in ToOLS as the conjunction of four distinct components:

- A set of heuristics to rank every choice;
- A set of primitives to express a search scheme independent of any time limit; it is composed by predefined choice points and combinations of choice points as in the OPL language [Hentenryck99];
- A set of primitives to express the search limits that depend on the current node, the current path or the current sub-tree; the resulting parameterized search algorithm controls the size of the explored search tree defined by one search scheme;
- A temporal strategy defined by a hybridization scheme, i.e. a cooperation of several parameterized searches, dealing with time allocation and selecting the tuning strategy of the parameters (static tuning, iterative tuning or adaptive tuning).

A *template of search* defines an abstract component of a search algorithm that can be reused to speed up the development process of customized partial search algorithms. This framework makes it easier to try new combinations of search limits and new temporal strategies.

Experiments on the weapon allocation problem show that partial search algorithms significantly improve the solution quality compared to a traditional approach [Givry.et.al99] and also demonstrates the gain in development time of new customized search algorithms. The code is clearer and more concise when using the search primitives. Another application in the Telecom domain is currently tested in our framework [Eole01].

The hybridization scheme is a way to define specific local search methods, such as large neighborhood search based on a sequence of partial searches in different neighborhoods. Pure local search methods could also be introduced in our framework as a black-box used by the hybridization scheme. The temporal control could be enhanced by an on-line learning mechanism, using the fact that similar problems are repeatedly solved in a real-time system. [Crawford.et.al01] gives the base for this mechanism.

References

[Beldiceanu.et.al98] Beldiceanu, N., E. Bourreau, H. Simonis, and D. Rivreau (1998). Introduction de métaheuristiques dans CHIP. In Proc. of MIC-98.

[Crawford.et.al01] Lara S. Crawford, Markus P.J. Fromherz, Christophe Guettier, Yi Shang. A Framework for On-line Adaptive Control of Problem Solving. In Proc. of CP-2001 workshop on On-Line combinatorial problem solving and Constraint Programming, Paphos, Cyprus, December 2001.

[Givry.et.al99] Simon de Givry, Pierre Savéant, Jean Jourdan. Optimization combinatoire en temps limité : Depth first branch and bound adaptatif. In Proc. of JFPLC-99, pages 161-178, Lyon, France, 1999.

[Givry.et.al01a] S. de Givry, P. Gérard, J. Jourdan, J. Mattioli, N. Museux, P. Savéant. How does constraint technology meet industrial constraints ? In Proc. of ESA workshop on On-Board Autonomy, pages 189-200, Noordwijk, The Netherlands, 17-19 October 2001, 12p.

[Givry.et.al01b] S. de Givry, Y. Hamadi, J. Mattioli, M. Lemaître, G. Verfaillie, A. Aggoun, I. Gouachi, T. Benoist, E. Bourreau, F. Laburthe, P. David, S. Loudni, S. Bourgault. Towards an on-line optimization framework. In Proc. of CP-2001 workshop on On-Line combinatorial problem solving and Constraint Programming, Paphos, Cyprus, December 2001. http://www.lcr.thomson-csf.com/projects/www_eole/workshop/olcp01-eole.ps

[Gomes.et.al98] C. Gomes, B. Selman, H. Kautz. Boosting Combinatorial Search Through Randomization. In Proc. of the 15th National Conference on Artificial Intelligence (AAAI-98), pages 431--437, Madison, WI, USA, 1998.

[Eole01] RNRT EOLE project, http://www.lcr.thomson-csf.com/projects/www_eole.

[Harvey&Ginsberg95] William D. Harvey, Matthew L. Ginsberg. Limited discrepancy search. In Proc. of IJCAI-95, pages 607-613, Montréal, Canada, 1995.

[Hentenryck99] P. Van Hentenryck. OPL: The Optimization Programming Language. The MIT Press, Cambridge, Mass., 1999.

[Korf90] Richard E. Korf. Real-time heuristic search. Artificial Intelligence, 42:189-211, 1990.

[Laburthe98] François Laburthe. SaLSA: a language for search algorithms. In Proc. of CP-98, pages 310-324, Pisa, Italy, October 26-30 1998.

[Laburthe.et.al98] Laburthe, F., P. Savéant, S. de Givry, and J. Jourdan. Eclair: A library of constraints over finite domains. Technical Report ATS 98-2, Thomson-CSF LCR, Orsay, France, 1998.

[Meseguer97] Meseguer, P. (1997). Interleaved depth-first search. In Proc. of IJCAI-97, Nagoya, Japan, pp. 1382-1387.

[Platon01] PLATON Team (2001). Eclair reference manual, Version 6.0. Technical Report Platon-01.16, THALES Research and Technology, Orsay, France.

[Perron99] L. Perron. Search Procedures and Parallelism in Constraint Programming. In Proc. of CP-99, pages 346-360, Alexandria, Virginia, 1999.

[Zilberstein96] Shlomo Zilberstein. Using Anytime Algorithms in Intelligent Systems. AI Magazine, 17(3):73-83, 1996.

Dynamic Scheduling and Plan Execution for Operations Automation in Multi-Satellite Control Centers

Pierrick GRANDJEAN, Pascal ALBAREDE

ASTRIUM
31, avenue des Cosmonautes, 31402 Toulouse, France
Tel: +33 5 62 19 67 81 Fax: +33 5 62 19 69 64
email: pierrick.grandjean@astrium-space.com

Abstract

As the number of spacecraft involved in Space Systems increases in satellite constellations or large satellite fleets, improving operations efficiency becomes an important goal of satellite operators, while maintaining a high level of safety, reliability and flexibility.

ASTRIUM has developed an Operations Schedule Manager that integrates dynamic, reactive scheduling and schedule execution, which is now operational in control centers.

Introduction

Improving operations efficiency and managing the complexity of fleet management are expected from operations automation (relieving operator from routine operations), operations concepts unification between heterogeneous satellites, and resource sharing (e.g. antennas).

Dynamic schedule management and schedule execution are critical to reach these goals.

In addition, the very high level of safety and service availability requirements for Space Systems must be maintained. These needs translate into operations flexibility and reactivity requirements.

This paper deals with the interactions between planning and execution, across :

- Operational requirements for dynamic scheduling and schedule execution
- The Schedule Manager approach and its implementation

Operational requirements

In addition to standard task scheduling constraints, satellite operations generate constraints such as transition delays (antenna pointing and lock), resource compatibility and links between ground and satellite resources.

A critical aspect of operational scheduling is the necessary ability to always produce an executable schedule even when all constraints cannot be satisfied. This must rely on operational rules such as priority ranking between tasks and constraints.

Schedule execution control requirements

Schedule execution must support either automated tasks executed by applications or manual tasks whose progress is indicated by operators.

The user interface must provide means to manually control task execution (acknowledgment, restart, abortion) and to modify task properties (such as their duration, priority, etc.).

On-line scheduling requirements

A frozen schedule is not compatible with emergency operations. Reactivity and flexibility are key operational requirements

Reactivity is the capacity to update the schedule according to upcoming events in near real-time. Events may be : task start, task termination, task duration extension, etc. Consequences of these events must be propagated as soon as they occur, providing controllers with a consistent view of the future at any time.

Flexibility means the capacity to manage resources allocation in real-time and to edit the schedule at any time. The main requirements are the following :

- Asynchronous short-time re-scheduling,
- Immediate scheduling of new unplanned tasks for emergency execution,
- Automated postponing of tasks that are waiting for resources used by other tasks,
- Assisted switching to backup resources in case of failure,
- Interruption of on-going tasks for allocating already used resources to higher priority tasks
- Manual allocation of resources to override automated scheduling

Integration of Scheduling and Execution

We propose an approach where resource optimization and operations execution are fully integrated in a single application, the **Schedule Manager**. The Schedule Manager is at the same time a **scheduling server** and an **execution controller**.

At the heart of a the control center client-server architecture, the Schedule Manager acts as the orchestra conductor of the Ground Segment : it allocates resources, drives equipment configuration, starts tasks at proper dates and monitors their execution until their termination.

3. A tight coordination between scheduling and execution when scheduling operates near the current time or when execution manages resources, through the management of individual software locks on resources and/or tasks.

4. A powerful and configurable User Interface that provides a global, detailed and consistent situation assessment, updated in real-time.

This approach supports any type of missions. A major reference is the new INTELSAT Control Center that runs safe and highly automated operations of 30 satellites and is now operational at INTELSAT headquarters and Earth stations.

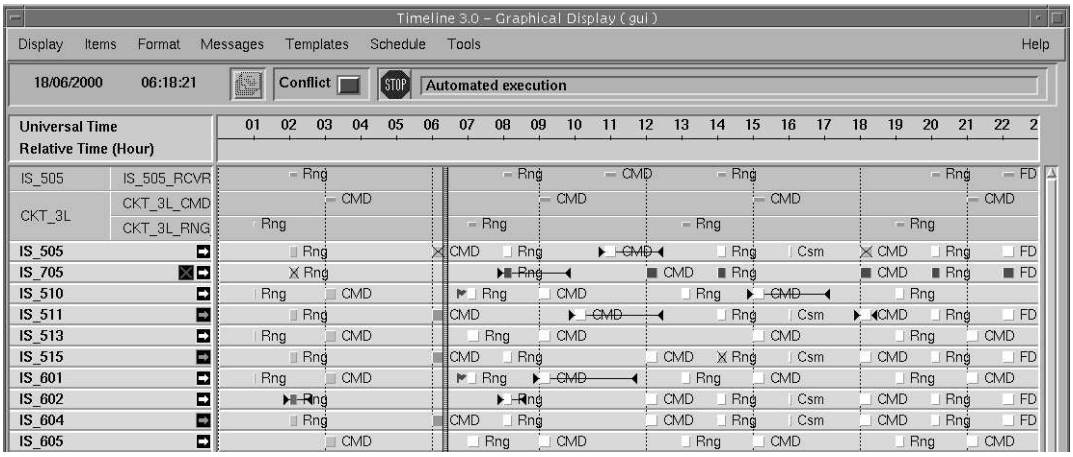


Figure 1 : TIMELINE User Interface

Scheduling is triggered when tasks are introduced or are modified, or when unexpected events occur. Scheduling and schedule execution must run in parallel and must be coordinated when scheduling operates near the current time in order to maintain schedule consistency.

Implementation by the TIMELINE product

The TIMELINE software product fully implements the proposed approach. It relies on the four following features :

1. A pragmatic focusing scheme enabling short re-scheduling computation time (from 5 to 30 seconds), even within very large schedules of thousands of tasks. Scheduling is based on CSP algorithms and ranking of constraints.

2. A multi-threaded software architecture that allows users and external clients to interact with the schedule even while re-scheduling.

Conclusion

Plan execution and re-scheduling capabilities provide an unprecedented level of automation in Spacecraft Ground Systems managing large satellite fleets on a limited set of ground equipment.

More detailed information can be found in the following references. Feel free to ask the authors for a copy.

References

Grandjean P. and Lecouat F., 2001. *Resource Optimization and Automated Operations Supervision : a winning Pair towards Operations Cost Reduction*. 4th International Symposium on Reducing the Cost of Spacecraft Ground Systems and Operations.

Lecouat, F. and Darroy J.M., 2000. *Tools for Operations Preparation and Automation : the OpsWare Approach*. 6th International Symposium SpaceOps 2000.

Rescheduling strategies for managing manufacturing systems complexity

Luisa Huaccho Huatuco

Manufacturing Systems Research Group, Department of Engineering, University of Oxford, Parks Road, Oxford, OX1 3PJ, United Kingdom; e-mail: lhuaccho@robots.ox.ac.uk

Introduction

The role of rescheduling has been widely acknowledged for helping the production floor to meet its objective functions, e.g. maximising throughput, in spite of the occurrence of internal or external disturbances (Jain and Elmaraghy 1997; Efstathiou 1996; Wu and Li 1995). However, little work has been done to assess the effects of rescheduling strategies on the management of manufacturing systems complexity. It is the aim of this research work to investigate the extent and nature of rescheduling effects on the overall complexity of the manufacturing facility.

Here, complexity is defined, from an information-theoretic point of view, as the expected amount of information required to describe the state of the manufacturing system (Calinescu et al. 2000; Efstathiou et al. 1999). In that connection, complexity is associated with the variety and uncertainty within manufacturing systems. Complexity can be classified into static and dynamic (Frizelle 1998; Frizelle 1995). Static complexity is related to the schedule (variety) whereas dynamic complexity is related to the deviations from the schedule (uncertainty).

This research work is being carried out following a three-step methodology: theoretical development, case study and computer simulations. Each of these steps is explained in the next sections.

Theoretical development

The theoretical development consisted of the creation of a generic model for the management of manufacturing systems complexity (Huaccho Huatuco et al. 2001a). The model illustrates the interaction between the production and the scheduling functions. It takes into account the regular monitoring of the system in order to check its adherence to the schedule and to react accordingly if it has deviated from schedule. This generic model highlighted the key elements of rescheduling.

Two sets of measures were chosen for the analysis of this research topic: traditional and entropic. Traditional measures include: Mean Tardiness, Mean Flow Time and Average Machine Utilisation. Entropic measures include both Static and Dynamic complexity indices to quantify the information content of the schedules and the disruptions between consecutive schedules, respectively.

Additionally, two sub-models were developed for assessing the cost and value of the complexity generated and managed by industry when rescheduling.

This theoretical step was complemented with the practical side of the methodology, the case study, which is explained next.

Case study

The case study involved ALPLA UK, a major plastic-bottle supplier based in the UK. The case study stages were: familiarisation, data collection, analysis of results and generation of conclusions/recommendations to the company. Nine weeks of data were collected, consisting of: weekly production schedules, daily revised schedules, reasons for changes, log book notes about details of production deviating from schedule on a daily basis.

Applying the entropic measures described above it was possible to conclude that when rescheduling was performed, the information content of schedules increased in time whereas the disruption between schedules decreased in time (Huaccho Huatuco et al. 2001b). In connection with the latter result, a further analysis on the value of complexity showed that rescheduling reduced the Non-value Adding (NVA) part of the complexity whereas it increased its Value-Adding (VA) part (Huaccho Huatuco et al. 2001c).

In order to extend the results from this case study it is necessary to test other different rescheduling strategies. This cannot be done on real-world organisations, as it would imply disrupting the operation of the manufacturing system. This is where computer simulations play a key role, as described next.

Computer simulations

The computer simulations step constitutes work in progress. The purpose of this step is to compare different scenarios of rescheduling strategies, amalgamating the model and measures of the theoretical development with the experience gained from the case study.

Two sets of rescheduling strategies are being studied. First, rescheduling strategies to handle internal disturbances, such as: production variation and machine breakdowns. Second, rescheduling strategies to handle external disturbances, such as: customer changes and supplier failures to deliver.

Those rescheduling strategies will be analysed in terms of their effect on different kinds of complexity and how they contribute to the overall manufacturing system performance, using the measures mentioned earlier.

From the preliminary results of the computer simulations step, it is possible to conclude that, as the rescheduling strategies get more and more refined, the size of the algorithm and the amount of input data increase.

The analysis and comparison of these scenarios will allow generating some generic conclusions on the research topic.

Conclusions

The conclusions of this research work are two-fold. First, theoretically it will provide a generic model to assess different rescheduling strategies for managing manufacturing systems complexity. Second, practically that model can be customised to the needs of a specific company and therefore constitute a decision-making tool to evaluate different scenarios before rescheduling the real system.

Acknowledgements

The author acknowledges the financial support of ORS award 1999032139, and the access to data provided by ALPLA UK.

References

Calinescu, A.; Efstathiou, J.; Sivadasan, S.; Schirn, J.; and Huaccho Huatuco, L. 2000. Complexity in Manufacturing: An Information Theoretic Approach. In Proceedings of the International Conference on Complexity and Complex Systems in Industry, 30-44. The University of Warwick, UK.

Efstathiou, J.; Tassano, F.; Sivadasan, S.; Shirazi, R.; Alves, J.; Frizelle, G.; and Calinescu, A. 1999. Information Complexity as a Driver of Emergent Phenomena in the Business Community. In Proceedings of the International Workshop on Emergent Synthesis, 1-6. Kobe University, Japan.

Efstathiou, J. 1996. Anytime heuristic schedule repair in manufacturing industry. *IEE Proceedings Control Theory Applications* 143(2): 114-124.

Frizelle, G. 1998. *The Management of Complexity in Manufacturing*, 161-182. Business Intelligence, UK.

Frizelle, G. 1995. Measuring Complexity as an Aid to Developing Operational Strategy. *International Journal of Operation and Production Management* 15(5): 26-39.

Huaccho Huatuco, L.; Efstathiou, J.; Sivadasan, S. and Calinescu, A. 2001a. A model for the control of complexity in manufacturing systems. In Proceedings of the 10th IFAC Symposium on Information Control Problems in Manufacturing (INCOM 2001), 6p. Vienna University of Technology, Austria.

Huaccho Huatuco, L.; Efstathiou, J.; Sivadasan, S. and Calinescu, A. 2001b. The effects of rescheduling on manufacturing systems complexity. In Proceedings of the 17th National Conference on Manufacturing Research (NCMR 2001): Advances in Manufacturing Technology XV, 377-382. University of Cardiff, UK.

Huaccho Huatuco, L.; Efstathiou, J.; Sivadasan, S. and Calinescu, A. 2001c. The value of dynamic complexity in manufacturing systems. In Proceedings of the International Conference of the Production and Operations Management Society (POMS - Brazil 2001), 180-188. Business Administration School of Sao Paulo, Brazil.

Jain, A. K., and Elmaraghy, H. A. 1997. Production scheduling/rescheduling in flexible manufacturing. *International Journal of Production Research* 35(1): 281-309.

Wu, H.-H., and Li, R. K. 1995. A new rescheduling method for computer based scheduling systems. *International Journal of Production Research* 33(8): 2097-2110.

Bringing out IxTeT in the dynamic world ?

Solange Lemai and Romain Trinquart
LAAS-CNRS, Toulouse, 7 Av. du Colonel Roche, {lastname}@laas.fr

Introduction

Augmenting the strength of an artificial system with autonomy is one of the grounding goals of research in the field of planning. But paradoxically the problem of embedding a planner in a real system has not been tackled face-to-face until recently. Thanks to huge improvements in the capabilities of planners and thanks to the emulation of application domains, some important steps have been made in this direction and in 1999 the Remote Agent Experiment (Johnson *et al.* 2000) has provided demonstration of spacecraft autonomy based on various technologies including planning and scheduling.

Animated by the same will to plunge planning into reality and thus to endow a system with high level decisional abilities, we propose in this paper the analysis of an example taken from the space domain and from which we draw general requirements for a planning system to cope with a dynamic context. We then confront the IxTeT system (Ghallab & Laruelle 1994) with these requirements. It is a partial order planner relying on CSP techniques to handle a rich temporal representation. From our experience of this system, we identify its advantages but also its defaults with respect to this new challenging application. In adopting this critical point of view, we tend to establish a road-map of the future works needed to achieve a demonstrator of an autonomous spacecraft control using such planning technologies.

Planning & Scheduling : a component of a generic architecture for autonomous systems - Before focusing on Planning and Scheduling, we took a more general look at the problem of controlling a system meant to act in a dynamic context. This led us to consider a generic control architecture towards which several important realizations of the Robotics community converge, both in and out of the space domain ((Alami *et al.* 1998), (Volpe *et al.* 2001)). It splits the control of a system between two levels. The low level control is responsible for the activation and the control of the physical system (effectors and sensors). The high level control is in charge of commanding the low level control. It is itself composed of two submodules, a reactive part and a deliberative one (see Fig. 1). It has to construct sequences of commands carefully so as to ensure that all known operational constraints are satisfied. The complexity of this task is worsen by the impossibility, in a real application, to list all the situations that might be encountered. Thus part of the control schema has to be adaptable. It is the role of the planning and scheduling component to allow evolution from the initial design so as to match the current context. It is in charge of foreseeing, of anticipating future controls by taking into account the current state of the system, of its environment and the current set of goals to achieve. Starting from this informal specification of a planning component in a control architecture, we propose more precise requirements justified by the study of a space mission.

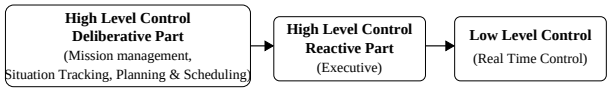


Figure 1: The classical 3-Layers Control Architecture

A mission example in the space context [1] **-** We consider a control system embedded in a satellite whose mission is to watch over fires and to monitor the activity of volcanos. The satellite is equipped with two cameras. The first one is used to detect out breaking fires : it can take wide pictures ahead of the satellite which are analyzed on-board. The second one is used to monitor volcanos and fire in case one is detected : it can take pictures of precise locations thanks to a scan mirror and of a higher resolution. The interaction between these two cameras entails dynamic context and involves the use of on-board planning : when a fire is detected, the satellite is required to send an alarm to the ground and to take a higher resolution picture of the fire, thus dynamically changing the activity of its second camera. Besides controlling these two cameras, the following activities should be tracked by the planning system: *orbit correction*, *attitude change*, *energy production*, *payload calibration*, *communication with the ground*... To each activity correspond one or several procedures managed by the executive.

Based on the analysis of this mission, the next section highlights some requirements for the successful use of a planner in a control system. It should be noted that in this study of on-line planning, interleaved with plan execution, the context of an autonomous satellite enables us to benefit from some restrictions on the encountered problems (a spacecraft moves in a more deterministic world than rovers for instance) and thus the following requirements might be incomplete with respect to the general problem of controlling an autonomous robotic system.

Requirements for dynamic planning

In this section, we try to figure out the main requirements for a planning system in the context of our mission example - requirements concerning both the expressiveness of its representation and the flexibility of its search process to take into account the interaction with the plan execution.

The first remark that can be done when considering a space domain, is the importance of time : almost all data concerning the satellite - its position on its orbit, the ground station visibility windows, the areas it flies over - are denoted by a temporal window. The space domain being closely related to the temporal issues, a planning system should be

[1] This example results from interactions with CNES and Astrium.

able to handle time, a first step being to handle actions with different durations. Furthermore, the activities of a satellite involve some complex temporal features. For instance, the duration of an *attitude change* activity depends on the angle of the manoeuvre, whereas the energy consumed depends on the duration of the activity. So the planning system should be able to express rich relations between the duration and the effects of an activity.

The planning system should also be able to manage resources of different types : symbolic ones (payload, solar panels, communication bus...) or with numeric quantities (power, energy, memory space, propellant). These resources can be borrowed, consumed and even produced during the mission.

Besides the information usually given to a planning system in its representation of the domain (initial state and operators), some external constraints have to be taken into account by the planner : contingent events such as eclipses, ground station visibility windows..., and the insertion of activities completely specified by some external module (parameters and execution time) such as *orbit correction* manoeuvres. Some of these external information can be computed in advance (by the Orbit Controller...). But the planning system may also need the help of specialized modules during its search to compute the parameters of actions (the mirror movement for instance).

Among the data computed in advance, some will remain valid on the horizon of one day (the orbit and the satellite position on the orbit given by the Navigator). Some others have to be computed for each orbit suh as the areas that the satellite flies over... So the planning system may have to consider different planning horizons, with different levels of abstraction. The planning duration should at least guarantee that a valid plan is available at the end of the current horizon.

In a satellite, FDIR (Failure Detection Identification and Recovery) mechanisms are implemented at each level of the architecture, to be able to efficiently react to failures. For instance almost all instruments are made redundant and, when one becomes out of order, a "reflex" action will switch to the other one. In this context, the planner does not need to have a detailed knowledge of the system and planning will be more efficient if it is done with a representation at a high level of abstraction. However, it can be interesting to keep the plan relatively flexible, especially concerning the temporal and resource consumption data, in the purpose of avoiding some costly replanning requests (the plan is further constrained during its execution).

In the context of interleaved planning and plan execution, the system sends back information, through the executive, to the planning system. This information is adapted to the system representation of the planner and concerns : the execution status of the activities, their dates of execution, the level of resources, the status of the instruments (available, failed)... To confront these data with the plan, an interface module in charge of the plan management is needed between the executive and the planning system. This *Plan Manager* contains the plan database, propagates the system information and checks its consistency with the constraints of the plan. As the executive (a procedural system) has no notion of time, the *Plan Manager* is also in charge of launching and stopping the procedures associated to the activities at the right time. For this purpose, the structure of the plan should contain information such as :

- a commitment status (an activity is being executed),
- an indication of the controllability of the activity (completely controllable : the system has to stop it / not controllable : the system has to wait for its termination),
- an indication of the possibility to interrupt an activity.

Moreover, the *Plan Manager* may have an interface with the users and manage the mission goals. It may be in charge of gathering and formulating all requests to the planning system. Requests may be of three types : compute a new plan for the next horizon, modify the current plan to add new goals (a fire has been detected...) or new activities, repair the current plan in case of failure. A plan failure can result from a failed execution status of an activity, or from the detection of inconsistency between the information sent back and the constraints of the plan database. The planning system needs then to keep trace, in the structure of the plan, of the dependency between the activities (to know which ones can still be executed and which ones should be replanned or even abandoned).

In this section, we have listed some general requirements for a planning system in the context of a satellite mission, each of these requirements being justified by an objective look at the example. In the process of this analysis, we went one step further and provided some elements of response, linked to the design of the architecture, by proposing the use of a Plan Manager. This choice also entails further specifications on the planner. In the next section, we investigate the adequation of a specific temporal planner with all these criteria.

Is IxTeT a good candidate for dynamic planning?

The IxTeT system is a lifted partial-order planner based on CSP managers to deal with a rich temporal representation. This representation is a functional one : it describes the world as a set of multi-valued functions of time, called attributes, and resources over which borrowing, consumption or production can be specified. The planner deals with a set of deterministic planning operators, called tasks, which are temporal structures giving partial specifications of the evolution of attributes over the task duration. Using ungrounded operators, the search process explores a tree in the plan space whose root node is a structure similar to a task which specifies the initial situation, goals with different associated dates and expected contingent events across the planning horizon.

This quick presentation of the IxTeT system already offers us opportunities to stress out some of the features that prompt us to consider it as a potential solution to meet the requirements expressed in the previous section.

The most immediate feature is the explicit handling of time : it is possible to express actions with different durations and effects occurring at different dates as well as persistency of some characteristics of the world over a time interval. Thus it is possible to handle concurrent actions in a detailed fashion. This explicit temporal qualification of the evolution of part of the world also provides support for expressing many required features in the initial partial plan, such as contingent events expected over the plan horizon or actions enforced by external modules to happen at given dates.

Another important feature of IxTeT is the extensive use of CSP managers which offers multiple advantages. The search process relies on these managers (one dealing with a temporal CSP, the other one handling atemporal variables) to solve conflicts in a partial plan by posting separation constraints or to explain goals by inserting operators and the related par-

tial binding constraints. But beyond these basic functionalities, much of the expressiveness is achieved through special constraints which are part of the description of tasks. For instance it is possible to express constraints linking timepoints and variables, such as algebraic equations : this is a way to describe the dependency between the duration of an action and its effects (see (Trinquart & Ghallab 2001)). Furthermore the use of complex constraints enables one to assert the use of a variable quantity of a resource over a time interval.
Last but not least, these CSP managers take part in the elaboration of flexible plans : they compute for each variable a minimal domain which reflects only the necessary constraints in the plan. Partially ordered and partially instantiated, plans can be further constrained at execution by a plan manager as was advocated in the previous section.

All of these interesting features encourage us to use the IxTeT system as the foundation to develop a control system endowed with planning capabilities. However this list of positive points should not push us to subjectivity; the remainder of this section is dedicated to the defaults of IxTeT and to the evaluation of the gap to fill in order to build a "dynamic planner".

IxTeT has mainly been used as an off-line planner, elaborating a plan based on the description of the goals, of the initial situation and of its operators. However, since it performs a search in the plan space, it can quite easely be adapted to incremental planning and to plan-merging operations. Such a work has been done for the PROBA project (managing the mission of an observation satellite) in (Gout, Fleury, & Schindler 1999), where goals (image requests) were incrementally inserted in the plan (but without invalidating it). In the case of our mission example, adding new goals may require the replacement of previously planned activities by new ones. Further work needs to be done to embed the ability of partially invalidating a plan and, from the resulting partial plan, elaborate a complete plan corresponding to new goals and a new initial state.

Thanks to the CSP-based approach, IxTeT provides flexible plans to the Plan Manager. However, these CSPs handle only controllable variables but many activities may involve uncontrollable ones. This problem has been addressed in the case of temporal variables - see (Morris, Muscettola, & Vidal 2001) for an adaptation of the temporal manager to verify the Dynamic Controllability Property- and in the case of atemporal variables - see (Fargier, Lang, & Schiex 1996) for consistency checking and minimal domain computation in a CSP mixing both controllable and non controllable variables.

Another interesting improvement of the IxTeT planner would be to enable the use of specialized modules during the search process to compute some parameters of the activities inserted in the plan. This work depends a lot on the type of information that is required. The main difficulty to do this comes from the fact that the planning process is based on least commitment and that activities are partially instantiated. It should be noted that some work has already been done in that way with the objective to combine IxTeT with a motion planner to compute certain data concerning the position of a robot (see (Lamare & Ghallab 1998)).

At last, we could not finish this study of IxTeT's defaults without mentioning that, in spite of its advantages - namely soundness and completeness - the planning process has one major drawback : it is not guaranteed to end in a limited time. Domain dependent heuristics can be defined to give an acceptable response time in most cases, but some standby procedures should be foreseen in case the planning process lasts too long.

Conclusion

In this short paper, we summarized the work we started to realize a prototype for demonstrating the use of advanced planning technologies in the context of an earth observation satellite mission. We started with an analysis of the mission context so as to exhibit requirements for the planning system used in this application. Then we discussed the adequation of the IxTeT system with such a realization. We draw positive conclusions which incite us to engage efforts in various directions to actually embed this system in a dynamic framework. The two main issues that we want to address in a close future are :

- the control of the search process to respond to the temporal constraints induced by the on-line setting,
- plan modification strategies in case of inadequacy to the state of the system resulting from execution.

References

Alami, R.; Chatila, R.; Fleury, S.; Ghallab, M.; and Ingrand, F. 1998. An architecture for autonomy. *International Journal of Robotics Research, Special Issue on Integrated Architectures for Robot Control and Programming* 17(4).

Fargier, H.; Lang, J.; and Schiex, T. 1996. Mixed constraint satisfaction: a framework for decision problems under incomplete knowledge. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*.

Ghallab, M., and Laruelle, H. 1994. Representation and Control in Ixtet, a Temporal Planner. In *Proceedings of the International Conference on AI Planning Systems*, 61–67.

Gout, J.; Fleury, S.; and Schindler, H. 1999. A new design approach of software architecture for an autonomous observation satellite. In *Proceedings of iSAIRAS*.

Johnson, A.; Morris, P.; Muscettola, N.; and Rajan, K. 2000. Planning in interplanetary space: Theory and practice. In *Proceedings of the International Conference on AI Planning Systems*.

Lamare, B., and Ghallab, M. 1998. Integrating a temporal planner with a path planner for a mobile robot. In *Proc. AIPS Workshop Integrating planning, scheduling and execution in dynamic and uncertain environments.*

Morris, P. H.; Muscettola, N.; and Vidal, T. 2001. Dynamic control of plans with temporal uncertainty. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

Trinquart, R., and Ghallab, M. 2001. An extended functional representation in temporal planning : towards continuous change. In *Proceedings of the European Conference on Planning (ECP)*.

Volpe, R.; Nesnas, I.; Estlin, T.; Mutz, D.; Petras, R.; and Das, H. 2001. The claraty architecture for robotic autonomy. In *Proceedings of the 2001 IEEE Aerospace Conference*.

Dynamic Arc-Consistency for Interval-based Temporal Reasoning

Malek Mouhoub and **Jonathan Yip**
Department of Math & Computer Science, University of Lethbridge
4401 University Drive, Lethbridge, AB, Canada, T1K 3M4
phone : (+1 403) 329 2557 fax : (+1 403) 329 2519
{mouhoub,yipj9}@cs.uleth.ca

Abstract

Many applications such as planning, scheduling, computational linguistics and computational models for molecular biology involve systems capable of managing qualitative and metric time information. An important issue in designing such systems is the efficient handling of temporal information in an evolutive environment. In a previous work, we have developed a temporal model, TemPro, based on the interval algebra, to express such information in terms of qualitative and quantitative temporal constraints. In order to find a good policy for solving time constraints in a dynamic environment, we present in this paper, a study of dynamic arc-consistency algorithms in the case of temporal constraints. We show that, an adaptation of the new AC-3 algorithm presents promising results comparing to the other dynamic arc-consistency algorithms. Indeed, while keeping an optimal worst-case time complexity, this algorithm has a better space complexity than the other methods.

Keywords: Temporal Reasoning, Dynamic Arc Consistency, Planning, Scheduling.

Introduction

Many applications such as planning, scheduling, computational linguistics(Song & Cohen 1991; Hwang & Shubert 1994), data base design(Orgun 1996) and computational models for molecular biology(Golumbic & Shamir 1993) involve managing temporal constraints. In linear planning, for example, during the search process the planner must order the set of actions forming the plan by imposing a collection of appropriate ordering constraints. These constraints are essential to guarantee the consistency of the resulting plan, that is, to guarantee that if the actions are executed starting at the initial state and consistently with these constraints, then the goal will be achieved. In nonlinear planning(Chapman 1987; Penberthy & d. Weld 1994) where the actions in a plan are partially ordered, maintaining the consistency of the ordering constraints is required, for example, when the planner attempts to establish a subgoal by reusing an action already in the plan under construction. Reasoning about constraints that prevent an action A from lying within a certain interval between two other actions A_1 and A_2 is also important in planners such as UCPOP(Penberthy & d. Weld 1992). The development of a domain-independent temporal reasoning system is then practically important. An important issue in designing such systems is the efficient handling of qualitative and metric time information. Indeed, the separation between the two aspects does not exist in the real world. In our daily life activities, for example, we combine the two type of information to describe different situations. In the case of scheduling problems, we can have qualitative information such as the ordering between tasks and quantitative information describing the temporal windows of the tasks i.e earliest start time, latest end time and the duration of each task.

In a previous work(Mouhoub, Charpillet, & Haton 1998), we have developed a temporal model, TemPro, based on the interval algebra, to express such information in terms of qualitative and quantitative temporal constraints. TemPro translates an application involving time information into a binary Constraint Satisfaction Problem[1] where constraints are temporal relations, we call it Temporal Constraint Satisfaction Problem (TCSP)[2]. Managing temporal information consists then in solving the TCSP using local consistency algorithms and search strategies based on constraint satisfaction techniques.

The aim of our work here is to solve a TCSP in a dynamic environment. Indeed, in the real world, when solving a TCSP we may need to add new information or relax some constraints when, for example, there are no more solutions (case of over constrained problems). In this paper, we will mainly focus on maintaining the arc-consistency dynamically. In a previous work, we have used arc-consistency algorithms(Mouhoub, Charpillet, & Haton 1998) to reduce the size of the TCSP representing the initial problem, by removing some values that do not belong to any solution. Indeed, an arc-consistency algorithm removes all inconsistencies involving all subsets of 2 variables belonging to the set of variables of the problem. In a dynamic environment we need to check if there still exist solutions to the problem every time a constraint has been added or removed. Adding temporal constraints can easily be handled by the arc-consistency algorithms we have used, we have just to

[1] A binary CSP involves a list of variables defined on finite domains of values and a list of binary relations between variables.

[2] Note that this name and the corresponding acronym was used in (Dechter, Meiri, & Pearl 1991). A comparison of the approach presented in this paper and our model TemPro is described in (Mouhoub, Charpillet, & Haton 1998).

put in this case the new constraint in the lists of constraints to be checked. However, constraint relaxation cannot be handled by these algorithms. Indeed, when we remove a constraint, these algorithms cannot find which value, that has been already removed, must be put back and which one must not. We must then use incremental arc-consistency algorithms instead (called also dynamic arc-consistency algorithms). Some dynamic arc-consistency algorithms have already been proposed in the literature. We also present in this paper a new dynamic arc-consistency algorithm which is a modification of a recent arc consistency algorithm(Zhang & Yap 2001) in order to handle dynamic constraints. Comparisons tests of the different dynamic arc consistency algorithms were performed on randomly generated dynamic temporal constraint problems. The results show that the new algorithm we propose has better performance than the others in most cases.

The rest of the paper is organized as follows : in the next section, we will present through an example, the different components of the model TemPro and its corresponding resolution methods. Maintaining dynamic arc-consistency in the case of temporal constraints is then presented in section 3. Section 4 is dedicated to the experimental comparison of the different dynamic arc-consistency algorithms. Concluding remarks and possible perspectives of our work are then presented in section 6.

Knowledge Representation

Example 1 : Consider the following typical temporal reasoning problem[3] :

1. *John, Mary and Wendy* **separately** *rode to the soccer game.*
2. *It takes John* **30 minutes**, *Mary* **20 minutes** *and Wendy* **50 minutes** *to get to the soccer game.*
3. *John either* **started or arrived** *just as Mary* **started**.
4. *John either* **started or arrived** *just as Wendy* **started**.
5. *John left home* **between 7:00 and 7:10**.
6. *Mary and Wendy* **arrived together but started at different times**.
7. *Mary arrived at work* **between 7:55 and 8:00**.
8. *John's trip* **overlapped** *the soccer game.*
9. *Mary's trip took place* **during** *the game or else the game took place* **during** *her trip.*

The above story includes numeric and qualitative information (words in boldface). There are four main events : John, Mary and Wendy are going to the soccer game respectively and the soccer game itself. Some numeric constraints specify the duration of the different events, e.g. *20 minutes is the duration of Mary's event*. Other numeric constraints describe the temporal windows in which the different events occur. And finally, symbolic constraints state the relative positions between events e.g. *John's trip overlapped the soccer game*.

Given these kind of information, one important task is to represent and reason about such knowledge and answer queries such as : "is the above problem consistent ?", "what are the possible times at which Wendy arrived at the soccer game ?", ... etc.
To reach this goal, and using an extension of the Allen algebra(Allen 1983) to handle numeric constraints, our model TemPro transforms a temporal problem involving numeric and symbolic information into a temporal constraint satisfaction problem (TCSP) including a set of events $\{EV_1, \ldots, EV_n\}$, each defined on a discrete domain standing for the set of possible occurrences (time intervals) in which the corresponding event can hold; and a set of binary constraints, each representing a qualitative disjunctive relation between a pair of events and thus restricting the values that the events can simultaneously take. A disjunctive relation involves one or more Allen primitives.

Relation	Symbol	Inverse	Meaning
X precedes Y	P	$P^{\smile}$	XXX YYY
X equals Y	E	E	XXX YYY
X meets Y	M	$M^{\smile}$	XXXYYY
X overlaps Y	O	$O^{\smile}$	XXXX YYYY
X during y	D	$D^{\smile}$	XXX YYYYYY
X starts Y	S	$S^{\smile}$	XXX YYYYY
X finishes Y	F	$F^{\smile}$	XXX YYYYY

Table 1: Allen primitives

The initial problem of figure 1 corresponds to the transformation of the temporal reasoning problem, we presented before, to a TCSP using the model TemPro. Information about the relative position between each pair of events is converted to a disjunction of Allen primitives. Indeed, Allen(Allen 1983) has proposed 13 basic relations between time intervals : starts (S), during (D), meets (M), overlaps (O), finishes (F), precedes (P), their converses and the relation equals (E) (see table 1 for the definition of the 13 Allen primitives). For example, the information "John either started or arrived just as Wendy started" is translated as follows : J $(S \vee M)$ W. In the case where there is no information between a pair of events, the corresponding relation is represented by the disjunction of the 13 Allen primitives (since this constraint is not considered during the resolution process, it does not appear on the graph of constraint as it is the case in figure 1 concerning the relation between Wendy and the Soccer game).
The domain of each event corresponding to the set of possible occurrences (we call it SOPO) that each event can take is generated given its earliest start time, latest end time

[3]This problem is basically taken from an example presented by Ligozat, Guesgen and Anger at the tutorial : Tractability in Qualitative Spatial and Temporal Reasoning, IJCAI'01. We have added numeric constraints for the purpose of our work.

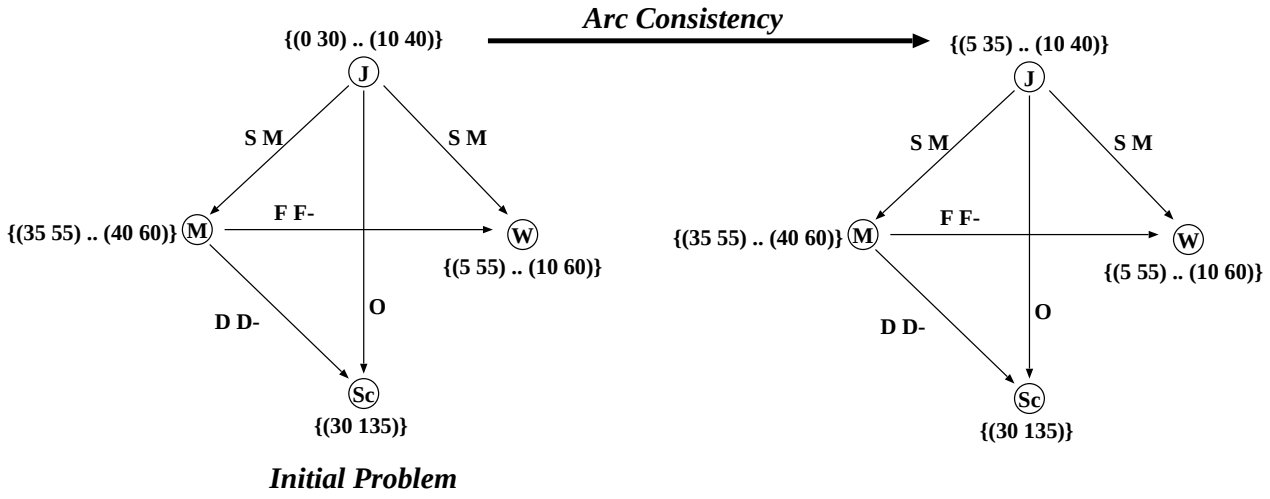


Figure 1: Applying arc-consistency to a temporal problem

and duration. In the case of Wendy's event, since we do not have any information about the earliest and latest time, these parameters are set respectively to 0 and the constant *horizon* (time before which all events should be performed). After a symbolic $\rightarrow$ numeric pre-process, these parameters are then set to 5 and 60 respectively.

Solving a TCSP consists of finding an assignment of a value from its domain to every variable, in such a way that every constraint is satisfied. Since we are dealing with a constraint satisfaction problem, deciding consistency is in general NP-hard[4]. In order to overcome this difficulty in practice, we have developed(Mouhoub, Charpillet, & Haton 1998) a resolution method performed in two stages. Local consistency algorithms are first used to reduce the size of the TCSP by removing some inconsistent values from the variable domains (in the case where arc-consistency is applied) and some inconsistent Allen primitives from the disjunctive qualitative relations (in the case where path consistency is performed). A backtrack search is then performed to look for a possible solution. When applying AC-3 to our temporal problem (see figure 1) the domain of event *J* is reduced.

Dynamic Maintenance of Local Consistency for Temporal Constraints

Dynamic Constraint Satisfaction Problem

A dynamic constraint satisfaction problem (DCSP) *P* is a sequence of static CSPs $P_0, \ldots, P_i, P_{i+1}, \ldots, P_n$ each resulting from a change in the preceding one imposed by the "outside world". This change can either be a restriction (adding a new constraint) or a relaxation (removing a constraint because it is no longer interesting or because the current CSP has no solution). More precisely, P_{i+1} is obtained by performing a restriction (addition of a constraint) or a relaxation (suppression of a constraint) on P_i. We consider that P_0 (initial CSP) has an empty set of constraints.

Dynamic Temporal Constraint Satisfaction Problem

Since a TCSP is a CSP in which constraints are disjunctions of Allen primitives, the definition of a dynamic temporal constraint satisfaction problem (DTCSP) is slightly different from the definition of a DCSP. Indeed in the case of a DTCSP, a restriction can be obtained by removing one or more Allen primitive from a given constraint. A particular case is when the constraint is equal to the disjunction of the 13 primitives (we call it the universal relation *I*) which means that the constraint does not exist (there is no information about the relation between the two involved events). In this particular case, removing one or more Allen primitives from the universal relation is equivalent to adding a new constraint. Using the same way, a relaxation can be obtained by adding one or more Allen primitives to a given constraint. A particular case is when the new constraint has 13 Allen primitives which is equivalent to the suppression of the constraint.

Figure 2 shows a restriction on the problem of example 1 obtained by removing some Allen primitives from the constraint between Wendy's event and the soccer game. This restriction is equivalent to the addition of the following information to our problem : *10. Wendy's trip took place during the game or else the game took place during her trip.*

Dynamic Arc-Consistency Algorithms

The arc-consistency algorithms we have used to solve a TCSP (Mouhoub, 1998) can easily be adapted to update the variable domains incrementally when adding a new constraint. In our example, adding the new constraint (see figure 2) will lead to an arc inconsistent TCSP which leads to an

[4]Note that some CSP problems can be solved in polynomial time. For example, if the constraint graph corresponding to the CSP has no loops, then the CSP can be solved in $O(nd^2)$ where n is the number of variables of the problem and d is the domain size of the different variables

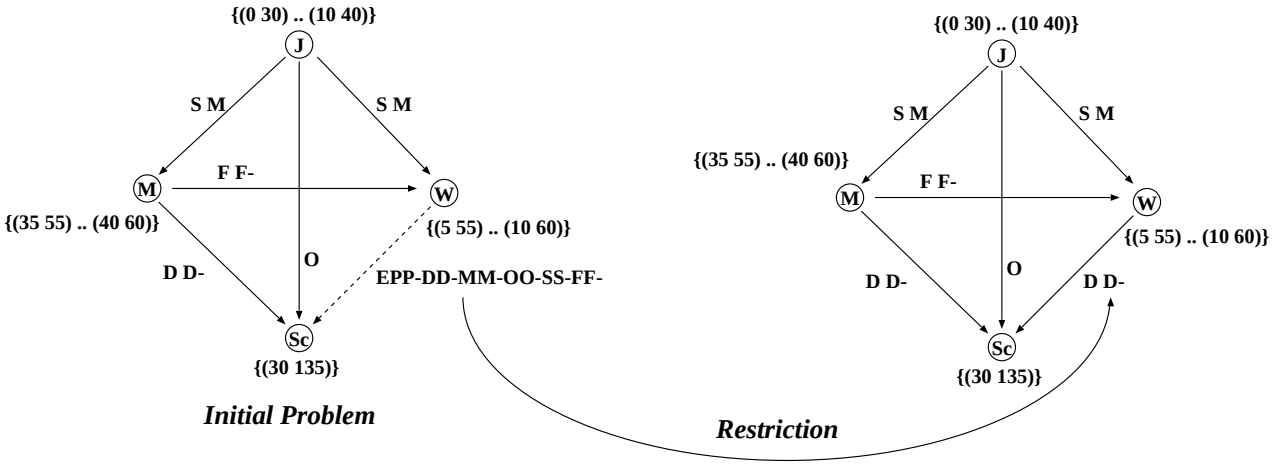


Figure 2: A restriction in a DTCSP

inconsistent TCSP. Let us assume now that to restore the arc-consistency we decide to relax the TCSP by adding one or more Allen primitives to a chosen constraint (one of the 10 constraints of our problem). In this case, the arc-consistency algorithms are unable to update the variable domains in an incremental way because they are not able to determine the set values that must be restored to the domains. The only way, in this case, is to reset the domains, add all the constraints (including the updated one) to the "unconstrained" TCSP (TCSP with no constraints) and then perform the arc-consistency algorithm. To avoid this drawback, dynamic arc-consistency algorithms have been proposed. Bessière has proposed DnAC-4(Bessière 1991) which is an adaptation of AC-4(Mohr & Henderson 1986). This algorithm stores a justification for each deleted value. These justifications are then used to determine the set of values that have been removed because of the relaxed constraint and so can process relaxations incrementally. DnAC-4 inherits the bad time and space complexity of AC-4. Indeed, comparing to AC-3 for example, AC-4 has a bad average time complexity(Wallace 1993). The worst-case space complexity of DnAC-4 is $O(ed^2 + nd)$ (e, d and n are respectively the number of constraints, the domain size of the variables and the number of variables). To work out the drawback of AC-4 while keeping an optimal worst case complexity, Bessière has proposed AC-6(Bessière 1994). Debruyne has then proposed DnAC-6 adapting the idea of AC-6 for dynamic CSPs by using justifications similar to those of DnAC-4(Debruyne 1995). While keeping an optimal worst case time complexity ($O(ed^2)$), DnAC-6 has a lower space requirements ($O(ed + nd)$) than DnAC-4. To solve the problem of space complexity, Neveu and Berlandier proposed AC|DC(Neuveu & Berlandier 1994). AC|DC is based on AC-3 and does not require data structures for storing justifications. Thus it has a very good space complexity ($O(e + nd)$) but is less efficient in time than DnAC-4. Indeed with its $O(ed^3)$ worst case time complexity, it is not the algorithm of choice for large dynamic CSPs. More recently, Zhang and Yap proposed an new version of AC-3 (called AC-3.1) achieving the optimal worst case time complexity with $O(ed^2)$ ((Zhang & Yap 2001))[5]. We have modified this algorithm in order to solve dynamic CSPs as we believe the new algorithm (that we call AC-3.1|DC) may provide better performance than DnAC-4 and DnAC-6.

AC-3.1|DC

Before we present the algorithm AC3.1|DC, let us recall the algorithm AC-3 and the new view of AC-3 (called also AC-3.1) proposed by Zhang and Yap(Zhang & Yap 2001).

Mackworth(Mackworth 1977) has presented the algorithm AC-3 for enforcing arc-consistency on a CSP. The following is the pseudo-code of AC-3 in the case of a TCSP. The worst case time complexity of AC-3 is bounded by $O(ed^3)$(Mackworth & Freuder 1985). In fact this complexity depends mainly on the way line 3 of the function *REVISE* is implemented. Indeed, if anytime the arc (i, j) is revised, b is searched from scratch then the worst case time complexity is $O(ed^3)$. Instead of a search from scratch, Zhang and Yap(Zhang & Yap 2001) proposed a new view that allows the search to resume from the point where it stopped in the previous revision of (i, j). By doing so the worst case time complexity of AC-3 is achieved in $O(ed^2)$.

Function $REVISE(i, j)$
1. $REVISE \leftarrow false$
2. **For** each interval $a \in SOPO_i$ **Do**
3. **If** $\neg compatible(a, b)$ **for each interval** $b \in SOPO_j$ **Then**
4. remove a from $SOPO_i$
5. $REVISE \leftarrow true$
6. **End-If**
7. **End-For**

[5] Another arc consistency algorithm (called AC-2001) based on the same idea as AC-3.1 (and having an $O(ed^2)$ worst case time complexity) was proposed by Bessière and Régin(Bessière & Régin 2001). We have chosen AC-3.1 for the simplicity of its implementation

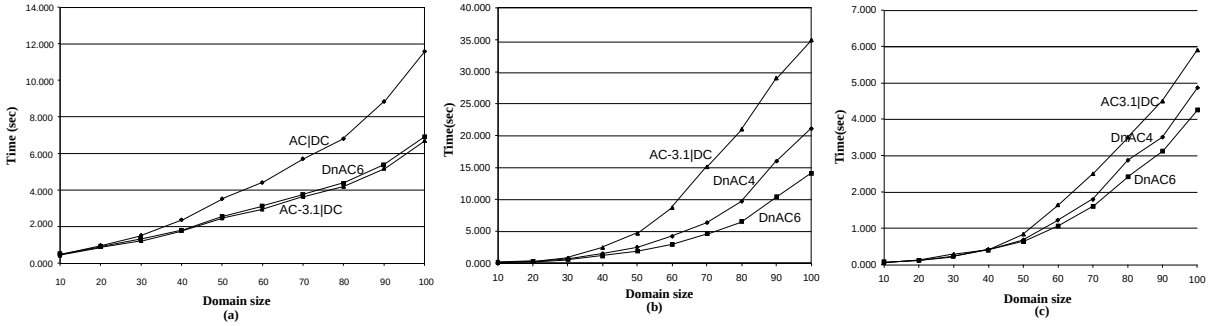


Figure 3: Experimental Tests on random DTCSPs

Algorithm AC-3

1. Given a TemPro network $TN = (E, R)$
   (*E: set of events,*
   R: set of disjunctive relations between events)
2. $Q \leftarrow \{(i, j) \mid (i, j) \in R\}$
   (list initialized to all relations of TN)
3. **While** $Q \neq Nil$ **Do**
4. $Q \leftarrow Q - \{(i, j)\}$
5. **If** $REVISE(i, j)$ **Then**
6. $Q \leftarrow Q \sqcup \{(k, i) \mid (k, i) \in R \wedge k \neq j\}$
7. **End-If**
8. **End-While**

In the case of constraint restriction, AC-3.1|DC works in the same way as AC-3.1. The worst-case time complexity of a restriction is then $O(ed^2)$. The more interesting question is whether AC-3.1|DC's time complexity can remain the same during retractions. Indeed, if we use the same way as for AC|DC (Neuveu & Berlandier 1994), one major concern is that during the restrictions, the AC-3.1 algorithm keeps a Resume table of the last place to start checking for consistency from. Unfortunately, during retractions, this Resume table may prove useless as values in the domain of nodes are restored. Our attempt was to follow an idea observed from the DnAC6 algorithm. Instead of replacing values in the node in the order they were deleted, the algorithm should place these values to be restored at the end of the list of values for that node, thereby keeping the Resume table intact. More precisely, the constraint relaxation, of a given relation (k,m) for example, is performed in 3 steps:

1. An estimation (over-estimation) of the set of values that have been removed because of the constraint (k,m) is first determined by looking for the values removed from the domains of *k* and *m* that have no support on (k,m),
2. the above set is then propagated to the other variables,
3. and finally a filtering procedure based on AC-3.1 is then performed to remove from the estimation set the values which are not arc-consistent with respect to the relaxed problem.

Since the time complexity of each of the above steps is $O(ed^2)$, the worst-case time complexity of a relaxation is $O(ed^2)$. Comparing to AC|DC, AC3.1|DC has a better time complexity. Indeed, the main difference between AC3.1|DC and AC|DC is the third step. This later step requires $O(ed^3)$ in the case of AC|DC (which results in a $O(ed^3)$ worst case time complexity for a restriction). In the case of AC3.1|DC, the third step can be performed in $O(ed^2)$ in the worst case because of the improvement we mentioned above. Comparing to DnAC-4 and DnAC-6, AC-3.1|DC has a better space complexity ($O(e+nd)$) while keeping an optimal worst-case time complexity ($O(ed^2)$).

Experimentation

In order to compare the performance of the 4 dynamic arc-consistency algorithms we have seen in the previous subsection, in the case of temporal constraints, we have performed tests on randomly generated DTCSPs. The criterion used to compare the above algorithms is the computing effort needed by an algorithm to perform the arc consistency. This criterion is measured by the running time in seconds required by each algorithm. The experiments were performed on a SUN SPARC Ultra 5 station. All the procedures are coded in C/C++. 3 classes of instances, corresponding to 3 type of tests, were generated as follows:

case 1: actions correspond to additions of constraints. $C = N(N-1)/2$ (constraints are added until a complete graph is obtained).

case 2: actions can be additions or retractions of constraints.
$C = N(N-1)/2$ additions $+N(N-1)/4$ retractions (the final TCSP will have $N(N-1)/4$ constraints).

case 3: this case is similar to case 1 but with inconsistent DTCSPs. Indeed in the previous 2 cases the generated DTCSPs are consistent. In this last case constraints are added until an arc inconsistency and thus a global inconsistency is detected (the inconsistency is detected if one variable domain becomes empty). Retractions are then performed until the arc-consistency is restored.

Results

Figure 3a) shows the results of tests corresponding to case 1. As we can easily see, the results provided by DnAC-6 and AC-3.1|DC are better than the ones provided by AC|DC and DnAC-4 (which do not appear on the chart). Since DnAC-6 requires much more memory space than AC-3.1|DC, this

latter is the algorithm of choice in the case of constraint additions. Figure 3b) and 3c) correspond to case 2 and case 3 respectively. DnAC-4 and DnAC-6 have better performance in this case than AC3.1|DC and AC|DC (the running time of AC|DC is very slow comparing to the other 3 algorithms). However, since AC3.1|DC does not require a lot of memory space, it has less limitations than DnAC-4 and DnAC-6 in terms of space requirements especially in the case of problems having large domain sizes.

Conclusion and Future Work

In this paper we present a comparative study of dynamic arc-consistent algorithms in the case of temporal constraint problems. The results shown demonstrate the efficiency of AC-3.1|DC, which is an adaptation of the new AC-3 algorithm for dynamic constraints, comparing to other algorithms. Indeed, while keeping an optimal worst-case time complexity, AC-3.1|DC requires less memory space than DnAC-4 and DnAC-6.

One perspective of our work is to look for a method to maintain path consistency when dealing with dynamic temporal constraints. Indeed, as we have shown in(Mouhoub, Charpillet, & Haton 1998), path consistency is useful in the filtering phase to detect the inconsistency when solving temporal constraint problems and also in the case where the numeric information is incomplete. The other perspective is to handle the addition and relaxation of constraints during the backtrack search phase. For example, suppose that during the backtrack search a constraint is added when instantiating the current variable. In this case, the instantiation of the variables already instantiated should be reconsidered and the domains of the current and future variables should be updated.

References

Allen, J. 1983. Maintaining knowledge about temporal intervals. *CACM* 26(11):832–843.

Bessière, C., and Régin, J. C. 2001. Refining the basic constraint propagation algorithm. In *Seventeenth International Joint Conference on Artificial Intelligence (IJCAI'01)*, 309–315.

Bessière, C. 1991. Arc-consistency in dynamic constraint satisfaction problems. In *AAAI'91*, 221–226.

Bessière, C. 1994. Arc-consistency and arc-consistency again. *Artificial Intelligence* 65:179–190.

Chapman, D. 1987. Planning for conjunctive goals. *Artificial Intelligence.* 32:333–377.

Debruyne, R. 1995. Les algorithmes d'arc-consistance dans les csp dynamiques. *Revue d'Intelligence Artificielle* 9:239–267.

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

Golumbic, C., and Shamir, R. 1993. Complexity and algorithms for reasoning about time: a graphic-theoretic approach. *Journal of the Association for Computing Machinery* 40(5):1108–1133.

Hwang, C., and Shubert, L. 1994. Interpreting tense, aspect, and time adverbials: a compositional, unified approach. In *Proceedings of the first International Conference on Temporal Logic, LNAI, vol 827*, 237–264.

Mackworth, A. K., and Freuder, E. 1985. The complexity of some polynomial network-consistency algorithms for constraint satisfaction problems. *Artificial Intelligence* 25:65–74.

Mackworth, A. K. 1977. Consistency in networks of relations. *Artificial Intelligence* 8:99–118.

Mohr, R., and Henderson, T. 1986. Arc and path consistency revisited. *Artificial Intelligence* 28:225–233.

Mouhoub, M.; Charpillet, F.; and Haton, J. 1998. Experimental analysis of numeric and symbolic constraint satisfaction techniques for temporal reasoning. *Constraints: An International Journal* 2:151–164, Kluwer Academic Publishers.

Neuveu, B., and Berlandier, P. 1994. Arc-consistency for dynamic constraint satisfaction problems : An rms free approach. In *ECAI-94, Workshop on Constraint Satisfaction Issues Raised by Practical Applications*.

Orgun, M. 1996. On temporal deductive databases. *Computational Intelligence* 12(2):235–259.

Penberthy, J., and d. Weld. 1992. Ucpop: A sound, complete, partial order planner for adl. In Nebel, B.; Rich, C.; and Swartout, W., eds., *Third International Conference on Principles of Knowledge Reprentation and Reasoning (KR'92)*, 103–114.

Penberthy, J., and d. Weld. 1994. Temporal planning with continuous change. In Nebel, B.; Rich, C.; and Swartout, W., eds., *Twelfth National Conference of the American Association for Artificial Intelligence (AAAI-94)*, 1010–1015.

Song, F., and Cohen, R. 1991. Tense interpretation in the context of narrative. In *AAAI'91*, 131–136.

Wallace, R. J. 1993. Why AC-3 is almost always better than AC-4 for establishing arc consistency in CSPs. In *IJCAI'93*, 239–245.

Zhang, Y., and Yap, R. H. C. 2001. Making ac-3 an optimal algorithm. In *Seventeenth International Joint Conference on Artificial Intelligence (IJCAI'01)*, 316–321.