

Enhancing the Anytime Behaviour of Mixed CSP-Based Planning

Christophe Guettier

SAGEM, 178, Rue de Paris, 91300 Massy, France
christophe.guettier@sagem.com

Neil Yorke-Smith

SRI International, Menlo Park, CA 94025, USA
nysmith@ai.sri.com

Abstract

An algorithm with the anytime property has an approximate solution always available; and the longer the algorithm runs, the better the solution becomes. Anytime planning is important in domains such as aerospace, where time for reasoning is limited and a viable (if suboptimal) course of action must be always available. In this paper we study anytime solving of a planning problem under uncertainty that arises from aerospace sub-system control. We examine an existing constraint model-based approach of the problem as a mixed constraint satisfaction problem (mixed CSP), an extension of classical CSP that accounts for uncontrollable parameters. We propose two enhancements to the existing decomposition algorithm: heuristics for selecting the next uncertain environment to decompose, and solving for incrementally longer planning horizons. We evaluate these enhancements empirically, showing that a heuristic on uncertainty analogous to ‘first fail’ gives the best performance, improving the anytime behaviour with respect to robustness to uncertainty. Further, we show that incremental horizon planning provides effective anytime behaviour with respect to plan executability, and that it can be combined with the decomposition heuristics.

Introduction

The increasing desire for autonomy in aerospace systems, such as Uninhabited Aircraft Vehicles (UAVs), leads to increasing complexity in planning, scheduling, and control problems (Verfaillie 2001). Constraint-based techniques have proved effective for addressing such problems in the aerospace domain (e.g. (Muscettola *et al.* 1998; Allo *et al.* 2002; Frank & Jónsson 2004)). The real-world requirements of such problems mean that preferences, uncertainty, and dynamic change must be handled. For this, the classical constraint satisfaction problem (CSP) is inadequate. One extension to handle uncertainty is the mixed CSP framework, introduced by Fargier *et al.* (1995; 1996).

Our motivation comes from a problem in online planning of the control of an aerospace component such as a thruster. In order to enhance autonomous behaviour, the plan produced must take account of environmental uncertainty the aerospace system may encounter. During execution, the plan needs only specify the immediate next control action: thus continuous, incremental planning is possible for the problem. A constraint-based formulation as a mixed CSP was given in (Yorke-Smith & Guettier 2003); in it uncontrollable

parameters model the uncertain evolution of physical quantities such as temperature.

An algorithm with the *anytime* property has an approximate solution always available; and the longer it runs, the better the solution becomes (Boddy & Dean 1994). If the algorithm is allowed to run to completion, a final solution is obtained. In the aerospace domain, planning is performed in an operational context where multiple levels of reactivity are required. This paper introduces an anytime solving approach to cope with uncertainty, in order to give more flexibility to the component management system during mission operations. Planning is invoked in the background, during cruise flight or in between critical phases of operation, and can be preempted at any time by higher priority tasks or by changes occurring at execution control levels. Thus the technique we described sits between and complements off-line planning and reactive component management.

This paper presents an experimental study of anytime planning with mixed CSPs. Specifically, we investigate the performance of the existing decomposition algorithm of (Fargier, Lang, & Schiex 1996) on our aerospace control planning problem as a case study. We describe two enhancements to the use of the algorithm designed to improve its anytime performance, and empirically assess their value. The two enhancements are decomposition heuristics for exploring the parameter space of uncertain environments, and incremental solving of the planning problem for successive horizons. The results show that a heuristic on uncertainty analogous to ‘first fail’ from the CSP literature gives the best performance, improving the anytime planning with respect to robustness to uncertainty. They also show that incremental horizon planning provides effective anytime behaviour with respect to plan executability, and that it can be combined with the decomposition heuristics.

Background and Problem Domain

Our motivation for studying planning with mixed CSPs comes from a planning problem arising in the aerospace domain, formalized as the *Sub-system Control Planning Problem* (SCPP); a detailed description is found in (Yorke-Smith & Guettier 2003; Yorke-Smith 2004). The planning is situated in an online context where both mission status and situation awareness may evolve in multiple, non-controllable ways. Since contingent events may unexpectedly occur, a safe course of action for the system is required to be immedi-

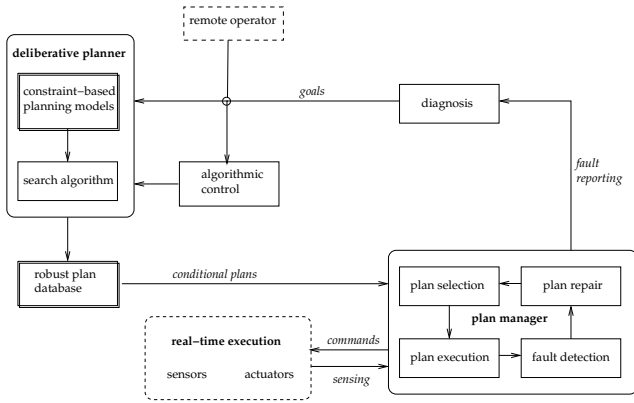


Figure 1: Outline architecture of a constraint-based agent.

ately available. The SCPP addresses deliberative plan generation in a dynamic context: deliberation is situated on board an autonomous system amid concurrent execution. Purely proactive, off-line planning is too slow, while purely reactive control provides a solution of too low quality.

Figure 1 illustrates how a constraint-based planning function can be integrated in an autonomous system, as the DSI Experiment demonstrated (Muscuttola *et al.* 1998).

Briefly, the input in an instance of the SCPP is a high-level description of an aerospace component, together with a description of the environmental uncertainty. A stated objective function on the performance of the component may also be given. The output sought is a plan for the commanding of the component. The plan must respect the behaviour guarantees that form part of the specification of the component. Further, as far as possible the plan must also optimize the performance of the component. The commanding is specified as a low-level automaton which models the component behaviour; for background, see (Arkin 1998).

For each aerospace component, an instance of the SCPP is parameterized by the planning horizon, $H \in \mathbb{N}$. Although the system plans supposing execution will begin at a future point in time, a decision can be demanded at any instant. Accordingly, during execution the plan needs only to specify the immediate next control action at the current horizon. During execution, the uncertain environment is observed just prior to each execution step.

Considered as a planning problem, the SCPP has: (1) non-deterministic actions due to contingent events, (2) fully observable states, and (3) semi-controllable effects, due to the environmental uncertainty. Planning consists of defining a consistent sequences of states in order to reach a given target state. This corresponds to the equipment changing modes of operation, in a feasible way, to reach a target mode. The timed sequence, the plan, must satisfy the model-based constraints and, possibly, optimize the performance function. As Figure 1 shows, the target mode is specified by mission and operational goals. For example, for a thruster component, the goal may be to achieve a certain thrust performance in a given time window, while maintaining the internal temperature within given limits.

The SCPP is stated as a unique rather than a sequential

decision problem. Our solution is a conditional plan that covers the anticipated environmental uncertainty. As will be explained, this contingent plan corresponds to the conditional decision of a full observability mixed CSP. An execution step consists of selecting a plan branch according to the observed environment at that step. The proactive approach ensures a valid plan is available, however the environment evolves within anticipated limits, once deliberation is complete. Since planning and execution are concurrent, deliberation may not have time to complete before the next decision is demanded; hence the requirement for anytime planning.

Mixed CSP and the Decomposition Algorithm

We now recall the mixed CSP framework and describe a constraint model-based representation of the SCPP as a mixed CSP. We then recall the algorithm presented in (Fargier, Lang, & Schiex 1996) for finding an optimal conditional decision in the case of full observability.

Mixed CSP

Fargier *et al.* (1995; 1996) introduced the *mixed CSP* framework, an extension to the classical CSP for decision making with incomplete knowledge.¹ In a mixed CSP, variables are of two types: decision and non-decision variables. The first type are controllable by the agent: their values may be chosen. The second type, known as *parameters*, are uncontrollable: their values are assigned by extrogeaneous factors. These factors are often referred to as ‘Nature’, meaning the environment, another agent, and so on.

Formally, a mixed CSP extends a classical finite domain CSP $\langle \mathcal{V}, \mathcal{D}, \mathcal{C} \rangle$, where \mathcal{V} is a finite set of variables, \mathcal{D} is the corresponding set of domains, and \mathcal{C} is a set of constraints:

Definition 1 A mixed CSP is a 6-tuple $\mathcal{P} = \langle \Lambda, L, V, D, \mathcal{K}, \mathcal{C} \rangle$ where:

- $\Lambda = \{\lambda_1, \dots, \lambda_p\}$ is a set of parameters
- $L = L_1 \times \dots \times L_p$, where L_i is the domain of λ_i
- $V = \{x_1, \dots, x_n\}$ is a set of decision variables
- $D = D_1 \times \dots \times D_n$, where D_i is the domain of x_i
- \mathcal{K} is a set of data constraints involving only parameters
- \mathcal{C} is a set of constraints, each involving at least one decision variable and zero or more parameters

A complete assignment of the parameters is a *realisation* (or world), and a complete assignment of the decision variables is a *decision* (or potential solution). A realisation is *possible* if the classical CSP $\langle \Lambda, L, \mathcal{K} \rangle$ is consistent, i.e. has at least one solution (otherwise the realisation is *impossible*). For every realisation r , the classical CSP $\langle \mathcal{V}, \mathcal{D}, \mathcal{C} \rangle$ formed as the projection of \mathcal{P} under realisation $\Lambda \leftarrow r$ is the *realised* (or candidate) problem induced by r from \mathcal{P} . A realisation is *good* if the corresponding realised CSP is consistent (otherwise *bad*). We say a decision d covers a realisation r if d is a solution to the realised CSP induced by r .

The nature of the outcome sought from a mixed CSP model depends on when knowledge about the state of the

¹The earlier work (Fargier *et al.* 1995) associates a probability distribution with each parameter; we follow the later work in which a (discrete) uniform distribution is assumed.

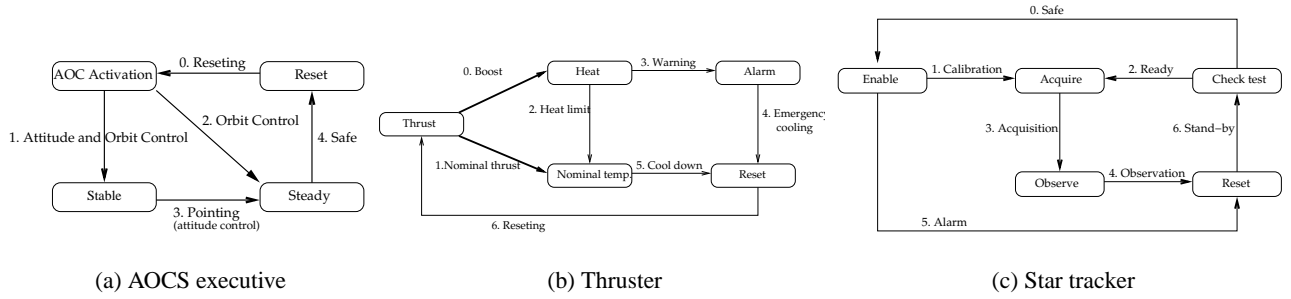


Figure 2: Discrete automata representing the behaviour of three spacecraft sub-systems

world will be acquired. If the realisation is observed before the decision must be made, we are in the case of *full observability*. In this case, the outcome sought is a *conditional decision* (or policy). This is a map between realisations and decisions that specifies, for a set of realisations R , a decision d for each $r \in R$ such that d covers r . We then say that the conditional decision *covers* R . Such a conditional decision is *optimal* if it covers every good realisation of \mathcal{P} . Deciding consistency of a binary mixed CSP is $\text{co-}\Sigma_2$ complete (Fargier, Lang, & Schiex 1996).

Mixed CSP Model of a SCPP Instance

A constraint-based formulation of the SCPP is given in (Yorke-Smith & Guettier 2003). Following earlier work (Allo *et al.* 2002), this model-based formulation represents the component activity over a fixed discrete horizon, using a constraint-based non-deterministic finite state automaton. This automaton, synthesized automatically from the problem specification, is represented concretely as a mixed CSP. A conditional decision policy for the mixed CSP model corresponds to a viable plan for the SCPP instance.

Importantly, although the constraints may be complicated, we formulate the model such that each constraint involves at most one parameter. This reduces the complexity of building the plan, because computing which realisations are covered by a decision in the resulting mixed CSP is simplified (Fargier *et al.* 1995). Parameters arise from uncertain environment conditions, such as temperature variation, in each state of the automaton. The model includes constraints describing evolution of fluent physical quantities according to the environmental uncertainty, such as:

$$\Theta_{i+1} = E_j \times (\Theta_i + T_i \Delta_j) \quad (1)$$

where Θ_i and T_i are discrete variables, E_i are Boolean, and Δ_j are parameters. The details of the model are not central to this paper; they may be found in (Yorke-Smith 2004).

There may be an additional minimum performance requirement on feasible plans. This requirement corresponds to a percentage of the maximum possible performance perf_{\max} (which can be computed a priori); it is imposed as an additional hard constraint in the model:

$$\text{perf}(S) \geq k \times \text{perf}_{\max} \quad (2)$$

where $\text{perf}(S)$ is the performance of a plan S , and $k \in [0, 1]$ is a given constant.

Figures 2(a)–2(c) show three discrete state automata. The automata represent the behaviour of three different, representative but simplified spacecraft sub-systems. They represent, respectively, an Attitude and Orbit Control System (AOCS), a thruster (Thruster), and a directional sensor (Tracker). While they bear some similarity in structure, the automata differ markedly in available actions and permitted timings, constraints, and evolution of fluent quantities such as energy (Yorke-Smith 2004). These differences impact strongly the problem difficulty, as exhibited by the experimental results that follow.

We build a mixed CSP model from the SCPP instance given by each automaton. Thus, the performance of solving these mixed CSPs will be the benchmark for our empirical study. These benchmarks are representative of some systems that are the source real-world of SCPP problems. Larger systems with more states and transitions can be envisaged, but they will not be necessarily formalized with automaton.

Decomposition Algorithm

We say an algorithm has an *anytime property* (Boddy & Dean 1994) if: (1) an answer is available at any point in the execution of the algorithm (after some initial time, perhaps zero, required to provide a first valid solution); and (2) the quality of the answer improves with an increase in execution time. The *performance profile* of an algorithm is a curve that denotes the expected answer quality $Q(t)$ with execution time t (Boddy & Dean 1989).

An algorithm to find an optimal conditional decision for a mixed CSP under full observability is presented in (Fargier *et al.* 1995; Fargier, Lang, & Schiex 1996). We call this the *decomposition algorithm* and denote it `decomp`. Because of the complexity of finding such a decision — both computational effort, and size of the outcome (in the worst case, one decision for every possible realisation) — `decomp` is designed as an anytime algorithm. Intuitively, it incrementally builds a list of decisions that eventually cover all good realisations. We omit discussion of some for us unnecessary subtleties about the algorithm.

Central to the method are sets of disjoint realisations called *environments*² and their judicious decomposition. Formally, an *environment* is a Cartesian product $l_1 \times \dots \times l_p$,

²Environmental uncertainty should be distinguished from this technical definition of an *environment* as a set of realisations.

Algorithm 1 Decomposition for an optimal cond. decision

```
1:  $B \leftarrow \emptyset$  {bad realisations}
2:  $D \leftarrow \emptyset$  {decision–environment pairs}
3:  $E \leftarrow L_1 \times \dots \times L_p$  {environments still to be covered}
4: repeat
5:   Choose an environment  $e$  from  $E$  {pick uncovered env.}
6:   let  $\mathcal{C}_e$  be constraints that enforce  $e$ 
7:   let  $P$  be the CSP  $\langle \Lambda \cup \mathcal{V}, L \cup D, \mathcal{K} \cup \mathcal{C} \cup \mathcal{C}_e \rangle$ 
8:   if  $P$  is consistent then
9:     let  $s$  be a solution of  $P$  {find dec. covering  $\geq 1$  real.}
10:    let  $v$  be  $s$  projected onto the domain variables  $\mathcal{V}$ 
11:     $R \leftarrow \text{covers}(v)$  {find all realisations covered by  $v$ }
12:    Add the pair  $(v, R)$  to  $D$ 
13:     $E \leftarrow \bigcup_{e' \in E} \text{decompose}(e', R)$  {removed covered}
14:  else {all realisations in  $e$  impossible}
15:    Add  $e$  to  $B$ 
16: until  $E = \emptyset$  {all possible realisations covered}
17: return  $(B, D)$ 
```

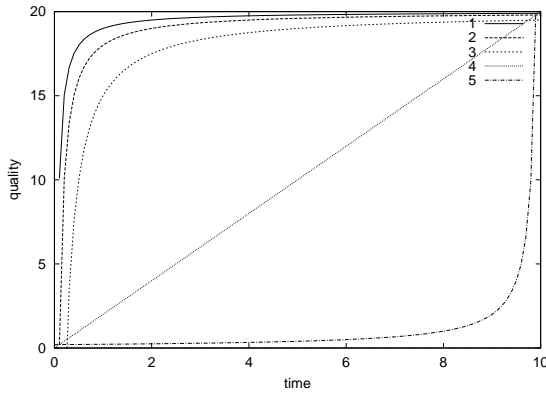


Figure 3: Idealized anytime performance profiles

where $l_i \subseteq L_i$: for example, if $L_1 = L_2 = \{a, b, c, d\}$, then an environment is $\{b, d\} \times \{c, d\}$. `decomp` is an anytime algorithm that incrementally computes successively closer approximations to an optimal decision. The number of realisations covered by the decision grows monotonically, and if allowed to finish without interruption, the algorithm returns an optimal conditional decision.

Pseudocode for `decomp` is given as Algorithm 1. (Fargier, Lang, & Schiex 1996) prove Algorithm 1 to be sound and complete: it eventually terminates, and if allowed to terminate, it returns a conditional decision that covers all good realisations. Moreover, if stopped at any point, D contains decisions for (zero or more) good realisations and B contains only bad realisations.

Enhancing the Anytime Behaviour of `decomp`

Summarizing, we have described a model of our motivating problem as a mixed CSP, and recalled the algorithm we call `decomp` for a full observability mixed CSP. We now introduce two orthogonal extensions of `decomp` designed to improve its anytime performance for the requirements arising for planning and scheduling in the aerospace domain.

To see what we mean by improved anytime behaviour, consider the performance profiles shown in Figure 3. The

horizontal axis depicts time t and the vertical axis solution quality $Q(t)$. The straight line 4 represents the behaviour of an algorithm that monotonically increases solution quality at a constant rate. The curves 1–3 depict better anytime behaviour than 4, with 1 the best, because solution quality rises more sharply earlier in the solving. In contrast, curve 5 depicts a poor anytime behaviour. Thus moving from 4 to 2, for instance, is an improvement in anytime behaviour. Note this is true even though both algorithms return the same solution quality at the end of the solving period shown. As a secondary aim, we would like, if possible, to have an earlier termination time in addition to improved anytime behaviour.

Observe that `decomp` is an anytime algorithm in terms of robustness to uncertainty, i.e. the number of realisations covered by the conditional decision it computes. Answer quality increases with time because, given execution may commence at any point, it is better to have a conditional plan more likely to cover the realisation actually observed. Indeed, if allowed to run to termination, the algorithm produces an optimal conditional decision; if stopped earlier, the conditional decision covers a subset of the good realisations. In terms of plan execution, however, `decomp` fails to ensure that a valid plan is *always* available (the first part of an anytime property): if the observed realisation is not covered by the conditional decision at the time of interruption, the algorithm does not provide a valid initial control action.

Note that, fitting the assumptions of the control problem, the model of uncertainty in the SCPP is deterministic (i.e. an implicit uniform distribution over the parameter domains). This does not rule out algorithms based on sampling the parameter space. However, sampling may not lead to an immediately executable control decision on interruption; like `decomp`, such an approach may be anytime with respect to plan robustness but not plan executability.

Environment Selection Heuristic

(Fargier, Lang, & Schiex 1996) note that heuristics may be used in line 5 of Algorithm 1, although none are proposed. The algorithm terminates when the set E is empty. Every iteration through the main loop removes one environment e from E . Judicious choice of e may speed the termination or improve the anytime behaviour w.r.t. robustness, or both.

We propose five heuristics for environment selection:

- **random**: pick the next environment at random. This is our default heuristic, used as a baseline to evaluate the others.
- **most uncertainty**: pick the environment with the most uncertainty. That is, choose e to maximize $\prod_{\lambda_i \in e} |L_i|$.
- **least uncertainty**: pick the environment with the least uncertainty. That is, choose e to minimize $\prod_{\lambda_i \in e} |L_i|$.
- **most restricting**: pick the environment that most constrains the variables' domains. That is, for each e , impose the constraints \mathcal{C}_e in line 6 of Algorithm 1, and compute $\prod_i |D_i|$. Choose e to minimize this quantity.
- **least restricting**: pick the environment that least constrains the variables' domains. I.e. impose the constraints \mathcal{C}_e , compute $\prod_i |D_i|$, and choose the maximizing e .

These heuristics are analogous to variable selection heuristics in finite domain CSP solving (Cheadle *et al.*

Algorithm 2 Computation by incremental plan horizon

```
1:  $S \leftarrow \emptyset$ 
2: for  $h = 1$  to  $H$  do
3:   let  $S_h$  be output of decomp on horizon  $h$  automaton
4:   if decomp ran to completion then
5:      $S \leftarrow S_h$ 
6:   else
7:     {keep existing decisions for uncovered realisations}
8:     for each realisation covered by  $S_h$  do
9:       update  $S$  by  $S_h$ 
10: return  $S$ 
```

2003). Pursuing this link, we also considered a heuristic to pick the most or least constraining environment: that whose realised CSPs are the most or least constrained (precisely, maximize or minimize the sum of a constrainedness metric, summed over all the realised CSPs corresponding to realisations in the environment). However, preliminary experiments indicated such a heuristic has poor performance. This seems to be caused by a weak correlation between the constrainedness of the realised CSPs arising from an environment, and the difficulty of solving the whole mixed CSP. Thus we did not consider such a heuristic further.

Incremental Horizon

The SCPP is naturally parameterized by the planning horizon, H . Running `decomp` to completion provides the sought optimal conditional plan. Interrupting the algorithm at any point provides a partial plan. As we have observed, since this plan is partial, in terms of execution it may not cover the realisation that actually occurs.

To better provide for plan execution, a second means of ensuring anytime behaviour is to iteratively plan for longer horizons, $h = 1, \dots, H$. A new plan is generated from scratch at each iteration, avoiding any myopia, but at the cost of not reusing the previous solution. We permit the algorithm to be interrupted at the completion of any horizon. The resulting optimal conditional decision for horizon h provides the initial steps of a complete plan for horizon H . We also permit `decomp` to be interrupted before completing a horizon. The plan for horizon h then consists of the decisions for the covered realisations, together with, for the uncovered realisations, the decisions from horizon $h - 1$.

More specifically, the time interval $[0 \dots h]$, $h \leq H$, defines a subproblem which is a subpart of the original SCPP instance. The subproblem is obtained by ignoring decision variables and parameters in the interval $[h+1, H]$, and relaxing associated constraints. Identifying these items to omit is straightforward due to the formulation of the model; omitting them yields a well-formed mixed CSP that describes the planning problem for the limited horizon h . The *incremental horizon* method starts from $h = 1$, and increments h each time the subproblem is successfully solved. If interrupted, the method thus provides a plan up to time event $h - 1$.

Algorithm 2 summarizes the method. As stated, conceptually it operates by solving incrementally larger subproblems. The advantage is that, in a given computation time, the plan produced may cover more of the good, possible realisations, compared to the plan produced by `decomp` for

horizon H in the same time. Indeed, suppose a plan for horizon H is desired and computation time is limited to T (which we do not assume is known to the algorithm). Running Algorithm 1 for time T might give a conditional plan that covers 70% of realisations, say. The conditional plan it yields is not optimal. Instead, running Algorithm 2 for the same time might give a plan that covers only 40% of realisations with a horizon- H decision, but all realisations are covered with some decision: say that for the horizon- $(H-1)$ decision. Thus we have an optimal conditional plan and, as the autonomous system begins plan execution, it can undertake further computation to extend the horizon- $(H-1)$ decisions to horizon- H .

In terms of execution, Algorithm 2 thus has the advantage over Algorithm 1 that a valid initial action is for certain available (once the problem is solved for horizon 1, which is expected rapid). Upon interruption, execution can proceed by checking whether the realisation observed is covered by the horizon h decision. If not, the horizon- $(h-1)$ decision for it is used. This checking requires little computation.

The incremental horizon method is orthogonal to the environment selection heuristics. Any heuristic may be used in the invocation of `decomp` in line 3 of Algorithm 2. In the experimental results that follow, we hence evaluate the behaviour of incremental horizon both with the default *random* heuristic and with the others proposed above.

Experimental Results

In this section we report an empirical assessment of the `decomp` algorithm on three SCPP instances. The aim of the experiments was to evaluate: (1) the impact of the environment selection heuristics on anytime behaviour with respect to robustness to uncertainty (measured by the completeness of the decision); and (2) the effectiveness of incremental horizon for producing anytime behaviour with respect to plan executability (measured by whether the decision contains an initial action for the actual realisation of the world).

The results reported were obtained on a 2GHz linux PC with 1GB of memory, using the constraint solver ECL¹PS^e version 5.7 (Cheadle *et al.* 2003); timings are in ms. Table 1 summarizes the characteristics of the three SCPP instances. For each automaton, we considered three degrees of uncertainty: moderate, average and large, denoted A–C respectively. We also considered performance requirements between 20–80% (recall equation (2)). This gives two parameters for each problem instance.

We imposed a timeout on any single run of `decomp`, depending on the complexity of the automaton; the values are given in Table 1. Note the nonlinear nature of the constraints of Tracker means that solving for this automaton is markedly

automaton	states per horizon	uncertainty per horizon			timeout
		A	B	C	
AOCS	5	2	4	5	200s
Thruster	8	7	14	23	2000s
Tracker	7	6	9	16	18000s

Table 1: Characteristics of the benchmark problem instances

more difficult; hence the greater timeout permitted.

Environment Selection Heuristic

We first consider the five environment selection heuristics. We measure quality by the number of good and possible realisations covered by the conditional decision, plus the number of bad realisations marked as bad, after a given computation time. That is, the quality is $Q_1(t) = |D| + |B|$, where D and B are as in the notation of Algorithm 1.

Figures 4(a)–4(f) show the quality (realisations covered) versus solving time (ms). Throughout, the vertical axis is shown on a log scale, i.e. $\log Q_1(t)$. Figure 4(a) shows the typical result for the AOCS instance: the best heuristic is *least uncertainty*, followed by *most restricting*; these are both better than *random*. The worst heuristic is *least restricting*; *most uncertainty* is slightly better.

Figures 4(b)–4(d) demonstrate the performance of the heuristics for Thruster is more varied. For most instances of uncertainty, performance, and horizon, *least uncertainty* is the best heuristic and *random* is second or third. However, for some instances, *least uncertainty* does not have maximal $Q_1(t)$ for all t . First look at Figures 4(c)–4(d). Note the scales of these two graphs are chosen to best compare the relative heuristic performances, not to show their overall anytime shape; hence some curves quickly reach the maximum quality shown, which is not the maximum attained. These graphs are for instances just before and just after infeasibility (which here occurs beyond horizon 6). In the former, *least uncertainty* is best at all times. In the latter, however, it is inferior to some other heuristics (in particular, to *random*) until about 2500ms, after which it strongly dominates. Heuristic *most restricting* exhibits poor behaviour.

Next look at the rare result in Figure 4(b). In this critically constrained problem, *random* is best at first, until overtaken by first *most uncertainty* then *least restricting*. Further, *least uncertainty* exhibits poor anytime performance. While exceptional, this instance indicates that no one heuristic always dominates. As in many algorithms that search through a plan or state space, the choice of heuristic is itself heuristic.

The results for Tracker confirm those for AOCS. Figures 4(e)–4(f) show *least uncertainty* as the best heuristic. Note it not only has a better performance profile, but also achieves much earlier termination than the other heuristics.

Incremental Horizon

We now consider the incremental planning method. Here, we measure quality by the horizon attained after a given computation time. That is, the problem is solved incrementally for horizons 1, 2, ..., and the times t_i recorded. The cumulative time for horizon h is computed as $t_h = \sum_{i=1, \dots, h} t_i$, and the quality is $Q_2(t) = \max\{h | t_h \leq t\}$.

Figures 5(a)–5(f) show log of the quality (horizon attained) versus solving time (ms). The shape of the curves indicate that Algorithm 2 provides acceptable anytime behaviour. However, performance strongly depends on the environment selection heuristic. Since incremental horizon is built on `decomp`, this might be expected.

Across the three automata, the performance of the *random* heuristic is broadly second or third of the five heuristics considered. For AOCS (Figures 5(a)–5(b)), the best heuristic

is *least uncertainty*, followed by *most restricting*; these are both better than *random*. The worst heuristic is *least restricting*; *most uncertainty* is slightly better. The performance of *most restricting* declines beyond horizon 6; beyond this point, *random* has better performance.

For Thruster and Tracker (Figures 5(c)–5(f)), the results are similar. The best heuristic is *least uncertainty*, and overall *random* is next best. For the Tracker instance A 20% (Figure 5(e)), beyond horizon 4, the remaining three heuristics struggle; *most uncertainty* is the best of them. For B 40% (Figure 5(f)), *random* and *least restricting* dominate about equally. The results for Thruster (Figures 5(c)–5(d)), while similar, show strongly that poor heuristics for environment selection give very poor performance. This appears to be due to the large number of environments that must be maintained by Algorithm 1; the algorithm suffers from a lack of memory, and the timeout is reached for Algorithm 2 while it is still considering a low horizon h .

Discussion

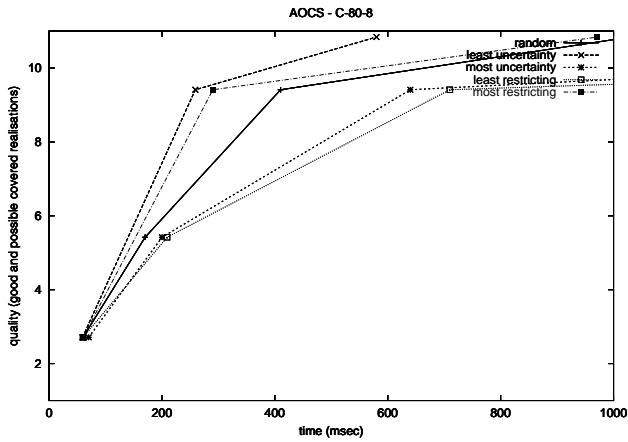
Of the environment selection heuristics, *least uncertainty* has the best overall performance, in terms of both metrics of quality. For the direct use of `decomp` (i.e. $Q_1(t)$), there are instances where other heuristics are better. In some instances, there is a ‘cross-over’ point (e.g. Figure 4(d)) prior to which another heuristic dominates, and after which *least uncertainty* dominates. For the incremental horizon use of `decomp` (i.e. $Q_2(t)$), *least uncertainty* dominates in almost all instances; we observe no cross-over behaviour.

We can make the analogy between *least uncertainty* (smallest environment first) and the *first fail* (smallest domain first) variable selection heuristic for classical CSP. ‘First fail’ is known as an often effective choice of variable selection heuristic (Cheadle *et al.* 2003). However, just as it is not the best heuristic for every CSP, so *least uncertainty* is not the best for every mixed CSP: Figure 4(b) shows a critically-constrained problem where the best heuristic is initially *random* then *most uncertainty*.

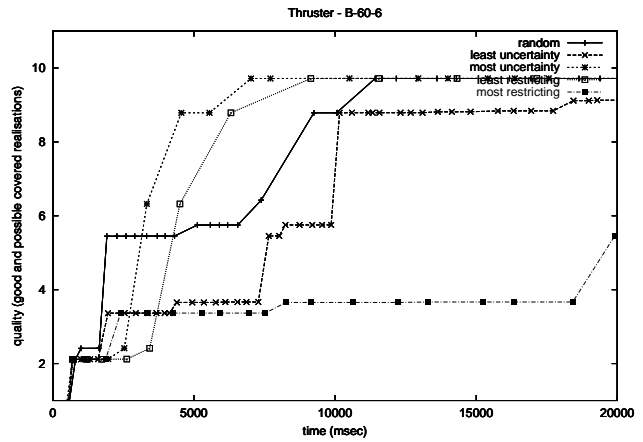
Secondly, overall *random* is consistently neither the best nor worst heuristic, as expected. On balance, its performance across the instances and across Algorithms 1 and 2 is second behind *least uncertainty*. Heuristics based on the size of variable domains (*most/least restricting*) vary in effectiveness between problem instances: e.g. *most restricting* is acceptable in Figure 4(a) but very poor in Figure 4(c). Note the size of variable domains loosely corresponds to the number of potentially feasible actions at a given plan state.

Thirdly, the results suggest that incremental horizon is effective in providing anytime behaviour with respect to plan executability, particularly for lesser horizons. When the sub-problems becomes hard (e.g. from $h = 4$ for Thruster), the rate of increase of solution quality declines. This is more marked when the performance requirement is higher, perhaps a result of the problem then being over-constrained.

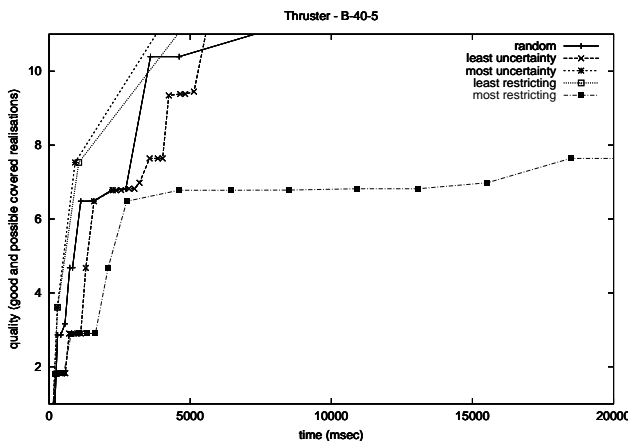
Since the SCPP is easy to solve for modest horizons, a possible approach might be: begin with Algorithm 2 and the *random* heuristic (which has no overhead to compute), and later switch to Algorithm 1 with the *least uncertainty* heuristic (the most effective overall). Further experimental work is needed to investigate this hybrid possibility.



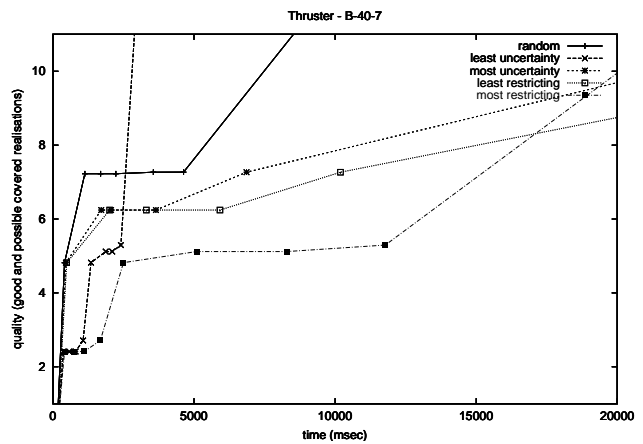
(a) AOCS C 80% horizon 8



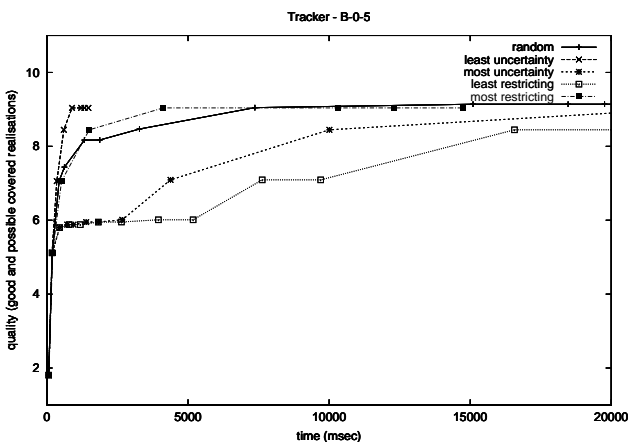
(b) Thruster B 60% horizon 6



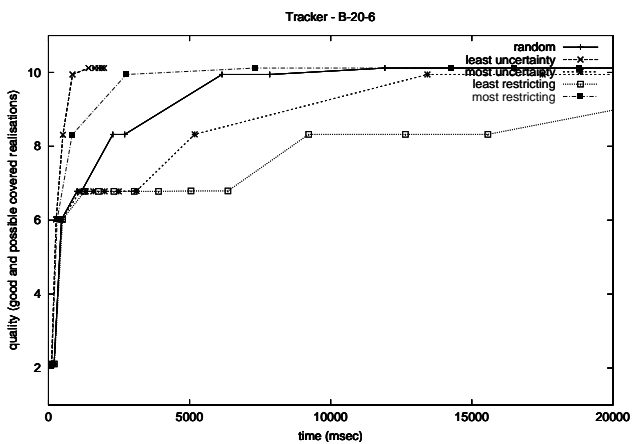
(c) Thruster C 40% horizon 5



(d) Thruster B 40% horizon 7

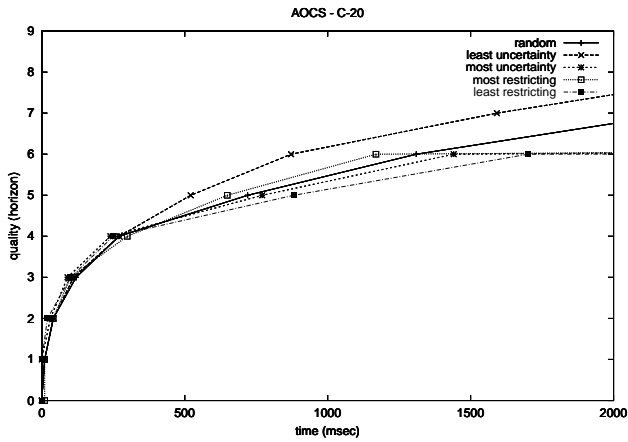


(e) Tracker B 0% horizon 5

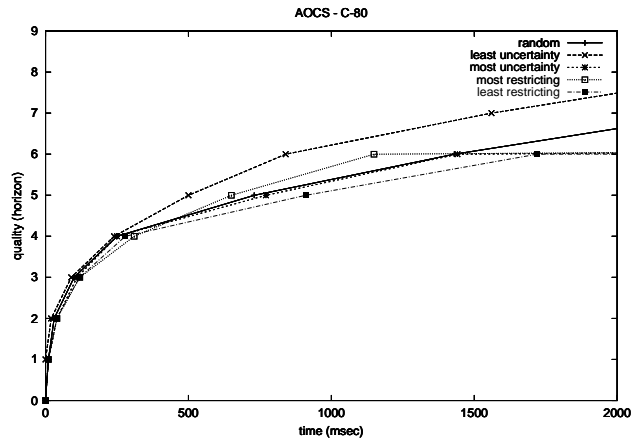


(f) Tracker B 20% horizon 6

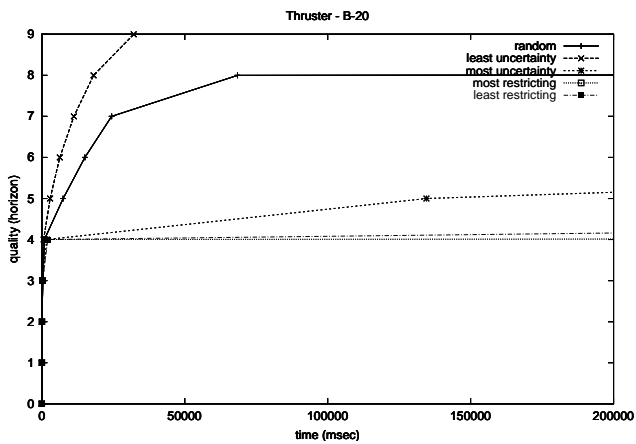
Figure 4: Anytime behaviour w.r.t. robustness of environment selection heuristics



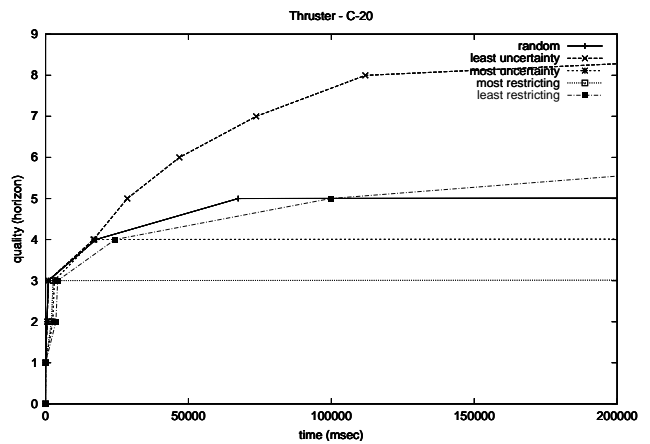
(a) AOCS C 20%



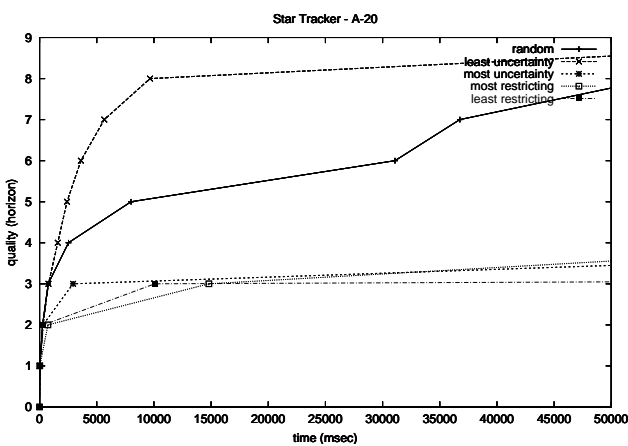
(b) AOCS C 80%



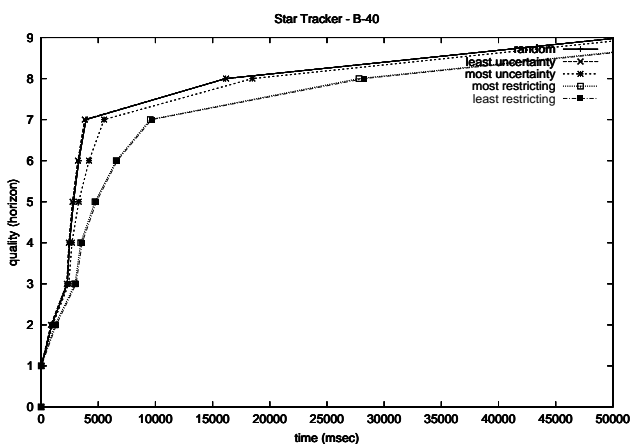
(c) Thruster B 20%



(d) Thruster C 20%



(e) Tracker A 20%



(f) Tracker B 40%

Figure 5: Anytime behaviour w.r.t. executability of incremental horizon

Related Work

One approach to deal with uncertainty in planning is to continuously adapt a plan according to the changing operating context. Plan adaptation is performed upon an unexpected event, system failure or goal (mission) update. Response time can be reduced by using, for example, iterative repair techniques (Chien *et al.* 2000). Rather than reacting, our approach here is based on proactive but online anticipation of the environment or other changes. In exchange for the space overhead of storing multiple plans, this has the operational advantage of enabling the system to reason more globally and react more quickly. However, plan monitoring, and possibly reactive plan repair, are important to handle unanticipated contingencies such as hardware failure. Further, online adaptation of conditional plans can reduce the size of the plan to be stored (Bresina & Washington 2001).

In practice, plan management and execution often adopts a hybrid proactive and reactive form (Verfaillie 2001). This is true for one of the most comprehensive approaches to uncertainty in the aerospace domain, *Reactive Model Programming Language* (RMPL) (Williams *et al.* 2003). RMPL and the approach used in this paper are both model-based and share the use of constraint-based automata. However, whereas the SCPP considers the control of a single component in view of anticipated uncertainty, RMPL is a control approach for a combined system, with the focus on managing the complexity of component interaction.

Our work is motivated by problems situated at the interface of system engineering, planning and control theory. Application of much classical work on planning to aerospace component control is difficult, due to challenging domain-specific requirements (Jónsson *et al.* 2000). Further, actions must be scheduled with respect to rich temporal constraints, the system must cope with large-scale problems, and for low-level components, behaviour must be guaranteed in the worst case. The latter point, together with the difficulty of estimating probabilities, also hampers the use of Markov Decision Processes (MDPs).

Our approach follows (Boddy & Dean 1994) in constructing flexible online solutions in the form of conditional plans. (Castillo, Fdez-Olivares, & González 2002) present a hybrid hierarchical and conditional planning approach for generating control of autonomous systems. Similar to our approach, during execution of the plan, a branch is selected according to the observed values of *runtime variables*. Our planning, however, is situated in an online context and also addresses time, resources, and uncertainty.

Similarities with several techniques in control theory exist mainly because environment assumptions apply to both the planning and control components of a system architecture, such as Figure 1, especially if the system must behave autonomously. However, the design of standard control components differs from our planning approach. In general, control techniques do not address the mission/operation level of granularity. Moreover, control properties like stability are expressed using an infinite horizon and a cost function defined over continuous domains.

Despite these differences, the vision of coexisting control and planning components in a unified system architecture motivates us to consider specific control techniques: hy-

brid control, model predictive control, and adaptive control (Levine 1996). These areas of control theory are applicable for autonomous systems evolving in uncertain and hostile environments. However, each only partially address the type of problems motivating this paper. The hybrid integration of planning, scheduling and control, supported by constraint solving techniques, is a potentially compelling avenue for the development of future autonomous systems.

An SCPP instance formulates a unique, rather than sequential, decision problem. Approaches to sequential decision making include MDPs and influence diagrams, and constraint-based frameworks, such as stochastic CSP (Manandhar, Tarim, & Walsh 2003), an extension of mixed CSP.

Lastly, anytime algorithms for classical CSPs have been built by considering a partial CSP and using branch-and-bound or local search (e.g. (Wallace & Freuder 1996)); (Cabon, de Givry, & Verfaillie 1998) address anytime computation of constraint violation lower bounds. For finding robust ‘super’ solutions, anytime algorithms have also been built with branch-and-bound (Hebrard, Hnich, & Walsh 2004). While anytime solving is related to incremental solving of CSPs, the focus there is on efficiently propagating the changes implied when a variable’s domain changes.

Conclusion and Future Work

Anytime behaviour is an important requirement for the aerospace domain. Motivated by an online planning problem for aerospace component control, this paper studied anytime planning under uncertainty with full observability mixed CSPs. We proposed two enhancements to the existing decomposition algorithm: heuristics for selecting the next environment to decompose, and solving of incrementally larger subproblems. Our empirical results indicate that incremental horizon planning provides effective anytime behaviour with respect to plan executability, and that it can be combined with the decomposition heuristics.

The heuristics we considered are applicable to solving any mixed CSP by the decomposition algorithm. Overall, the heuristic *least uncertainty*, which is analogous to ‘first fail’ for finite domain CSPs, gives the best performance. We have yet to consider heuristics based on analysis of priorities (e.g. criticality) from a model of system operation.

The incremental horizon method is specialized for the SCPP. By replacing the decomposition algorithm with an incremental version, we ensure anytime behaviour in terms of plan execution. However, the broader idea of decomposition into incremental subproblems, as a means of anytime solving, applies to any mixed CSP for which a suitable sequence of subproblems can be identified.

Incremental horizon is a baseline approach with similarities to iterative deepening. In future work, we want to complete our investigation by evaluating how often it produces plans for horizon H based on partial plans for a lower horizon, as described earlier. Initial results indicate a trade-off according to the problem hardness: critically-constrained instances have fewer feasible actions (so greater overlap between subproblems) but are harder to solve (so fewer realisations covered in any given time). Because of the complicated mapping between high-level planning decisions and variables in the constraint model, we want to eval-

uate the methods considered here on other and larger SCPP instances (Yorke-Smith 2004), besides mixed CSP models arising from other planning problems.

Algorithm 2 produces a succession of limited horizon plans. In our current implementation, planning at horizon h makes no reuse of the horizon $h - 1$ plan. We thus want to explore solution reuse as a planning heuristic between different horizons, and to compare our proactive approach to reactive solving of the single horizon control problem at different times. Both reactive and limited horizon proactive planning may lack completeness. In particular, if in Algorithm 2 we base planning for the next horizon on the solution for the last, the plan generated to horizon $h - 1$ may not be extendible to horizon h : for example, if it uses a resource (such as energy) that any plan for the next horizon may need. Thus this idea is more suited to managing a system whose global return must be optimized (for example, an observation system that must make the most useful observations over all its life) rather than when managing a system in order to lead it to a designated goal state.

The ‘cross-over’ between different heuristics over time suggest that meta-reasoning on the solving algorithm may yield the best anytime behaviour in practice. More generally, this reasoning can take into consideration (Hansen & Zilberstein 1996): the current state of the plan (such as what percentage of realisations it presently covers); the expected computation time remaining, if an estimate is available; the cost of computing the different heuristics; and the opportunity of switching between algorithms during solving. Such reasoning can be consolidated by analyzing the complexity of decomposition algorithm, thus providing metrics to understand how problem difficulty is balanced between the set of realisations and the core planning problem.

Driven by our motivational problem, in this paper we have considered only the full observability case, an instance of contingent planning; an interesting direction would be to consider anytime solving in the no observability case. Here, the outcome sought to a mixed CSP is a single robust solution that covers as many realisations as possible, i.e. a conformant plan. As such, there are links to anytime methods for robust solutions to CSPs and to solving mixed CSPs with probability distributions over the parameters, and to probabilistic planning (e.g. (Onder & Pollack 1999)). In particular, now scenario sampling methods for stochastic CSPs give the opportunity for an anytime algorithm (Manandhar, Tarim, & Walsh 2003).

Acknowledgments. We thank K. Myers, T. Winterer, and the anonymous referees for helpful comments, and the participants of the Changes’04 Workshop for suggestions on an earlier version. This work was undertaken when the second author was at IC-Parc, partially supported by the EPSRC under grant GR/N64373/01.

References

Allo, B.; Guettier, C.; Legendre, V.; Poncet, J.; and Strady-Lecubin, N. 2002. Constraint model-based planning and scheduling with multiple resources and complex collaboration schema. In *Proc. of AIPS’02*.

Arkin, R. C. 1998. *Behavior-Based Robotics*. Cambridge, MA: MIT Press.

Bertoli, P., and Cimatti, A. 2002. Improving heuristics for planning as search in belief space. In *Proc. of AIPS’02*, 143–152.

Boddy, M. S., and Dean, T. L. 1989. Solving time-dependent planning problems. In *Proc. of IJCAI’89*, 979–984.

Boddy, M., and Dean, T. L. 1994. Deliberation scheduling for problem solving in time-constrained environments. *Artificial Intelligence* 67(2):245–285.

Bresina, J. L., and Washington, R. 2001. Robustness via run-time adaptation of contingent plans. In *Proc. of AAAI 2001 Spring Symposium on Robust Autonomy*.

Cabon, B.; de Givry, S.; and Verfaillie, G. 1998. Anytime lower bounds for constraint violation minimization problems. In *Proc. of CP’98*, LNCS 1520, 117–131.

Castillo, L.; Fdez-Olivares, J.; and González, A. 2002. Shifting AI planning technology from automated manufacturing to autonomous operation and control in space missions. In *Proc. of AI Planning and Scheduling For Autonomy in Space Applications*.

Cheadle, A. M.; Harvey, W.; Sadler, A. J.; Schimpf, J.; Shen, K.; and Wallace, M. G. 2003. ECLiPSe: An Introduction. Technical Report IC-Parc-03-1, IC-Parc, Imperial College London.

Chien, S.; Knight, R.; Stechert, A.; Sherwood, R.; and Rabideau, G. 2000. Using iterative repair to improve responsiveness of planning and scheduling. In *Proc. of AIPS’00*.

Fargier, H.; Lang, J.; Martin-Clouaire, R.; and Schiex, T. 1995. A constraint satisfaction framework for decision under uncertainty. In *Proc. of UAI’95*, 167–174.

Fargier, H.; Lang, J.; and Schiex, T. 1996. Mixed constraint satisfaction: A framework for decision problems under incomplete knowledge. In *Proc. of AAAI-96*, 175–180.

Frank, J., and Jónsson, A. 2004. Constraint-based attribute and interval planning. *Constraints* 8(4):339–364.

Hansen, E. A., and Zilberstein, S. 1996. Monitoring the progress of anytime problem-solving. In *Proc. of AAAI-96*.

Hebrard, E.; Hnich, B.; and Walsh, T. 2004. Super solutions in constraint programming. In *Proc. of CP-AI-OR’04*, 157–172.

Jónsson, A.; Morris, P.; Muscettola, N.; Rajan, K.; and Smith, B. 2000. Planning in interplanetary space: Theory and practice. In *Proc. of AIPS-00*.

Levine, W. S., ed. 1996. *The Control Handbook*. CRC Press.

Manandhar, S.; Tarim, A.; and Walsh, T. 2003. Scenario-based stochastic constraint programming. In *Proc. of IJCAI’03*.

Muscettola, N.; Nayak, P. P.; Pell, B.; and Williams, B. 1998. Remote agent: To boldly go where no AI system has gone before. *Artificial Intelligence* 103(1–2):5–48.

Onder, N., and Pollack, M. E. 1999. Conditional, probabilistic planning: A unifying algorithm and effective search control mechanisms. In *Proc. of AAAI-99*.

Verfaillie, G. 2001. What kind of planning and scheduling tools for the future autonomous spacecraft? In *Proc. of the ESA Workshop on On-Board Autonomy*.

Wallace, R. J., and Freuder, E. C. 1996. Anytime algorithms for constraint satisfaction and SAT problems. *SIGART Bulletin* 7(2).

Williams, B. C.; Ingham, M.; Chung, S. H.; and Elliott, P. H. 2003. Model-based programming of intelligent embedded systems and robotic explorers. *IEEE Proceedings* 9(1):212–237.

Yorke-Smith, N., and Guettier, C. 2003. Towards automatic robust planning for the discrete commanding of aerospace equipment. In *Proc. of 18th IEEE Intl. Symposium on Intelligent Control (ISIC’03)*, 328–333.

Yorke-Smith, N. 2004. *Reliable Constraint Reasoning with Uncertain Data*. Ph.D. Dissertation, Imperial College London.