Anytime Behaviour of Mixed CSP Solving

Neil Yorke-Smith and Christophe Guettier

IC–Parc, Imperial College London, SW7 2AZ, U.K.
{nys,cgue}@icparc.ic.ac.uk

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 solving 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 the anytime behaviour of solving a mixed CSP, an extension of classical CSP that accounts for uncontrollable parameters, using a benchmark problem from aerospace sub-system control. We propose two enhancements to the existing decomposition algorithm: heuristics for selecting the next uncertain environment to decompose, and solving of incrementally larger subproblems. We evaluate these enhancements empirically, showing that a heuristic on uncertainty analogous to 'first fail' gives the best performance. We also show that incremental subproblem solving provides effective anytime behaviour, and can be combined with the decomposition heuristics.

1 Introduction

The increasing desire for autonomy in aerospace systems, such as Uninhabited Aircraft Vehicles (UAVs), will lead to increasing complexity in planning, scheduling, and control problems [14]. Constraint programming techniques have proved effective for addressing such problems in the aerospace domain (e.g. [13, 15]). 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. [4, 5] for decision making with incomplete knowledge.

Our motivation comes from a problem in planning the control of aerospace equipment. In order to enhance autonomous behaviour, the plan produced must take account of environmental uncertainty the aerospace system may encounter. A constraint-based formulation as a mixed CSP is given in [20], where uncontrollable *parameters* are used to 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 the algorithm runs, the better the solution becomes [1]. If the algorithm is allowed to run to completion, a final solution is obtained. For the aerospace domain, with its deadlines on response time, anytime behaviour is highly desirable [14].

This paper presents an experimental study of anytime solving of mixed CSPs. Specifically, we study the performance of the existing decomposition algorithm of [5] 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’ gives the best performance. They also show that incremental subproblem solving provides effective anytime behaviour, and can be combined with the decomposition heuristics.

We begin by summarising the mixed CSP framework and the decomposition algorithm (Section 2), and then briefly summarise our motivation for anytime solving and outline our case study problem (Section 3). We present the two algorithmic extensions (Section 4) and empirically assess them (Section 5); and a discussion of the results concludes the paper (Section 6).

2 Mixed CSP and the Decomposition Algorithm

2.1 Mixed CSP

Fargier et. al. [4, 5] 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 extrogeneous 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 $\langle \mathcal{V}, \mathcal{D}, \mathcal{C} \rangle$:

Definition 1 (Mixed CSP [5]). *A* mixed CSP *is a 6-tuple* $\mathcal{P} = \langle \Lambda, L, V, D, \mathcal{K}, \mathcal{C} \rangle$ *where:*

- $\Lambda = \{\lambda_1, \ldots, \lambda_p\}$ *is a set of parameters*
- $L = L_1 \times \cdots \times L_p$, *where* L_i *is the domain of* λ_i
- $V = \{x_1, \ldots, x_n\}$ *is a set of decision variables*
- $D = D_1 \times \cdots \times D_n$, *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*

We say 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). We say 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 $\mathcal{P}$. We say 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 outcome sought from a mixed CSP model is a *robust solution*: intuitively, one solution that satisfies all constraints under as many realisations as simultaneously possible. However, the nature of this outcome depends on when knowledge about the state

[1] The earlier work [4] associates a probability distribution with each parameter; we follow the later work in which a (discrete) uniform distribution is assumed.

of the world will be acquired. If the realisation is observed before the decision must be made, we are in the case of *full observability* (FO). If the realisation is observed only after the decision must be made, we are in the case of *no observability* (NO). The intermediate cases are not considered in [4, 5].[2]

In the case of full observability, 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 $\mathcal{P}$; it is *complete* if further it covers all possible realisations. It is shown in [5] that deciding consistency of a binary mixed CSP is co-Σ_2 complete.

2.2 Decomposition Algorithm

We say an algorithm has an *anytime property* [1] if:

1. An answer is available at any point in the execution of the algorithm
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 output quality $Q(t)$ with execution time t [2]. The concept of an anytime algorithm, one that has an anytime property, was developed for time-critical planning and scheduling.[3] The original formulation imposed three additional requirements:

1. Ability to be preempted
2. Continuity of the function $Q(t)$ from time to quality
3. Diminishing marginal improvement of quality with time

An algorithm to find an optimal conditional decision for a mixed CSP under full observability is presented in [4, 5]. 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; for details, see [5].

Central to the method are sets of disjoint realisations called *environments*, and their judicious decomposition, which is achieved with a method called *sub-domain subproblem extraction* [6]. Formally, an *environment* is a Cartesian product $l_1 \times \ldots l_p$, where $l_i \subseteq L_i$. For example, if $L_1 = L_2 = \{a, b, c, d\}$, then an environment is $\{b, d\} \times \{c, d\}$.

[2] In temporal CSP, (FO) corresponds to weak controllability of STPUs and (NO) to strong controllability; dynamic controllability [12] is not considered for mixed CSPs.

[3] In the literature, two types of anytime algorithms have been defined. An *interruptible* algorithm is defined as above. A *contract* algorithm must be given in advance an upper limit on runtime; it will terminate within this limit with a partial solution. We consider interruptible algorithms because in the aerospace domain we do not necessarily have an estimate of available runtime before execution begins. Moreover, an interruptible algorithm can be converted to a contract algorithm with a constant factor overhead.

Algorithm 1 Decomposition for an optimal conditional decision

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

The result 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. However, the algorithm is approximate in that the conditional decision obtained is not guaranteed to have minimal cardinality.

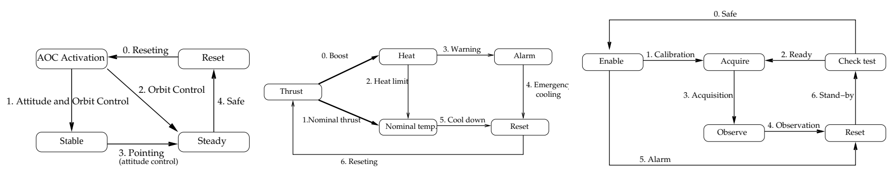
Pseudocode for `decomp` is given as Algorithm 1. In line 5 we pick an environment not yet covered. It is possible if at least one of its realisations is possible. If so, we find a decision that covers one of its realisations (line 9), compute the other realisations covered by the decision (line 11), and remove them from the environment (line 13). On the other hand, if the environment is bad (i.e. all its realisations are bad), we mark all its realisations so in line 15.

The consistency test in line 8 should be performed by instantiating the parameters Λ first.[4] In fact, the consistency test and subsequent search for one solution to the CSP in line 9 can be combined, since if P has a solution then it is by definition consistent.

The function $covers(d)$ in line 11 calculates the realisations covered by a decision d. Operationally, $covers(d)$ can be specialised for the constraints of the problem. In particular, it is simplified when each constraint contains at most one parameter. In this case, the set of realisations R covered by d is a Cartesian product of subsets of the parameter domains L. Hence we can build R by considering each parameter independently; moreover, R is an environment with no further computation.

The function $decompose(e',R)$ in line 13 implements sub-domain subproblem extraction to decompose e' by R, returning a set of distinct environments [6]; the details are unnecessary for this paper. Decomposing an environment e by an environment R

[4] This is because we must know whether the CSP $\langle \Lambda, L, \mathcal{K} \rangle$ is consistent. If so, environment e contains at least one possible realisation; otherwise we do not proceed with e. This is a necessary condition for the correctness proof of the algorithm [5].



(a) AOCS executive (b) Thruster (c) Star tracker

Figure 1. Discrete automata representing the behaviour of three spacecraft sub-systems

means producing a set of distinct environments S that together cover all realisations in e not covered by R.

Using results about environments from [6], in [5] Algorithm 1 is proved 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.

3 Problem Domain

From the introduction, recall that our motivation for studying mixed CSPs comes from the aerospace planning problem described in [20]. The problem is called the *Sub-system Control Planning Problem* (SCPP); a detailed description is found in [20, 19].

As noted earlier, planned future autonomy in the aerospace domain brings strong anytime requirements. Autonomous systems are characterised by limited computational power and limited online response time. Moreover, due to contingent events that may unexpectedly occur, a safe course of action is required to be immediately available.

In this paper we focus on the anytime solving of the constraint model of the SCPP. This model, derived from a high-level specification of a problem instance as a finite state automaton, is a mixed CSP. Importantly, although the constraints may be complicated, each constraint involves at most one parameter. The parameters arise from uncertain environment conditions, such as temperature variation, in each state of the automaton.

The model includes linear summation constraints (arising from path conservation constraints), implication constraints, channelling constraints; and constraints describing evolution of physical quantities according to the environmental uncertainty, such as:

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

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 [19].

The outcome sought for the SCPP is a conditional plan that covers the anticipated environmental uncertainty.[5] This corresponds to the conditional decision of a full observability mixed CSP. For a given aerospace sub-system, an instance of the SCPP is

[5] Environmental uncertainty should be distinguished from the technical definition of an *environment* above as a set of realisations.

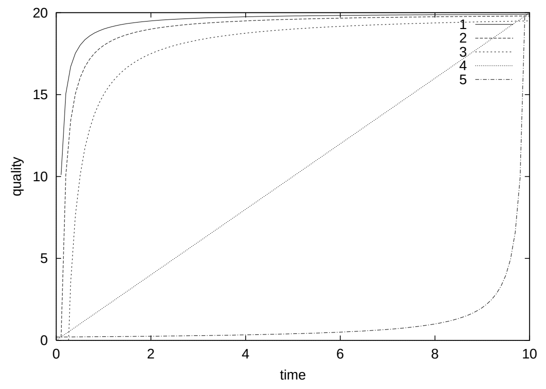


Figure 2. Performance profile curves of idealised anytime behaviour

parametrised by the planning horizon, $H \in \mathbb{N}$. Additionally, there is an minimum performance requirement on feasible solutions. This requirement corresponds to a percentage of the maximum possible performance (which can be computed a priori); it is imposed as an additional hard constraint in the model.

Figures 1(b)–1(c) show three discrete state automata (for illustration only). The automata represent the behaviour of three different, representative but simplified spacecraft sub-systems. These represent, respectively, an Attitude and Orbit Control System (AOCS), a thruster (Thruster), and a directional sensor (Tracker). We build a mixed CSP model of each automaton. The performance of solving these mixed CSPs will be the benchmark for our empirical study.

4 Enhancing the Anytime Behaviour of `decomp`

Summarising, we have recalled the algorithm we call `decomp` for a full observability mixed CSP (Algorithm 1), and described a model of our motivating problem as such a mixed CSP. We now introduce two orthogonal extensions of `decomp` designed to improve its anytime performance for the requirements arising in aerospace domain.

To see what we mean by improve anytime behaviour, consider the performance profiles shown in Figure 2. The horizontal axis depicts time t and the vertical axis solution quality $Q(t)$. The straight line 4 represents the anytime 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.

4.1 Environment Selection Heuristic

Recall that `decomp` is an anytime algorithm in terms of the number of realisations covered by the conditional decision it computes. If allowed to run to termination, it produces an optimal conditional decision; if stopped earlier, the conditional decision covers a subset of the good realisations.

In [5] it is noted 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, 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 maximise $\prod_{\lambda_i \in e} |L_i|$.
- ***least uncertainty***: pick the environment with the least uncertainty. That is, choose e to minimise $\prod_{\lambda_i \in e} |L_i|$.
- ***most restricting***: pick the environment that most constraints the variables' domains. That is, for each e, impose the constraints $\mathcal{E}$ in line 6 of Algorithm 1, and compute $\prod_i |D_i|$. Choose e to minimise this quantity.
- ***least restricting***: pick the environment that least constraints the variables' domains. That is, impose the constraints $\mathcal{E}$, compute $\prod_i |D_i|$, and choose the maximising e.

These heuristics are analogous to variable selection heuristics in finite domain CSP solving. Pursuing this link, we also considered a heuristic to pick the most or least constraining environment. That is, the environment whose realised CSPs are the most or least constrained (precisely, maximise or minimise the sum of a constrainedness metric, summed over all the realised CSPs corresponding to realisations in the environment). However, preliminary experiments indicated that such a heuristic has poor performance. This seems to be caused by only a weak correlation between the constrainedness of the realised CSPs that an environment leads to, and the difficulty of solving the whole mixed CSP. Thus we did not consider such a heuristic further.

4.2 Incremental Horizon

The SCP problem is naturally parametrised by the planning horizon, H. For a given horizon, running `decomp` to completion provides the sought optimal conditional plan for horizon H. Interrupting the algorithm at any point provides a partial plan.

A second means of ensuring anytime behaviour is to iteratively plan for longer horizons, $h = 1, \ldots, H$. We permit the algorithm to be interrupted at the completion of any horizon h. The resulting complete condition plan 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 $h-1$.

Algorithm 2 Anytime 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 $S$
```

More specifically, the time interval $[0 \ldots h],\ h \leq H$ defines a subproblem which is a subpart of the original SCP problem instance. The subproblem is obtained by ignoring decision variables and parameters in the interval $[h+1, H]$, and relaxing associated constraints. 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 $h-1$. Hence, the solution quality is measured by the (cumulative total) time to complete construction of the plan for horizon $h-1$.

Algorithm 2 summarises 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 horizon-$(H-1)$ decision. Thus we have an optimal conditional plan and, as we begin its execution, we can undertake further computation to extend the horizon-$(H-1)$ decisions to horizon-H decisions.

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 thus will evaluate the behaviour of the incremental horizon method both with the default *random* heuristic, and with the others proposed above.

5 Experimental Results

In this section we report an empirical assessment of the `decomp` algorithm on the SCP problem. The aim of the experiments was to evaluate: (1) the impact of the environment selection heuristics on anytime behaviour; and (2) the effectiveness of incremental horizon for producing anytime behaviour.

The results reported were obtained on a 2GHz linux PC with 1GB of memory, using ECL^{i}PS^{e} version 5.7 [3]; timings are in seconds. The SCPP instances were the three au-

Table 1. Characteristics of the benchmark problem instances

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

tomata given in Figure 3. Table 1 summarises their characteristics. 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 Section 3). This gives two parameters for each problem instance. We imposed a timeout on any single run of Algorithm 1, depending on the complexity of the automaton; the values are also given in Table 1.

5.1 Environment Selection Heuristic

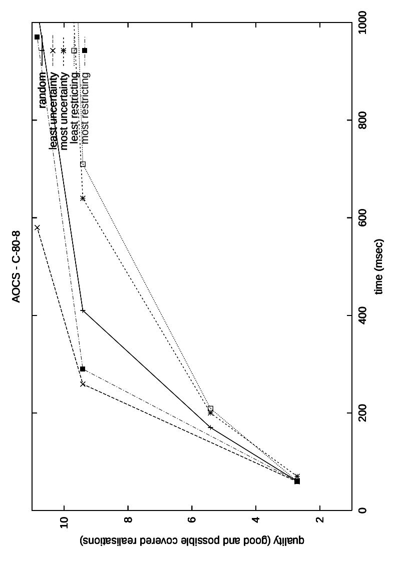
We first consider the five environment selection heuristics described in Section 4.1. We measure quality by the number of good and possible realisations covered by the conditional decision, plus the number of bad realisations marked as bad, after a given computation time. That is, the quality is $Q_1(t) = |D| + |B|$, where D and B are as in the notation of Algorithm 1.

Figures 3(a)–5(b) show the quality (realisations covered) versus solving time (ms). The vertical axis is shown on a log scale, i.e. $\log Q_1(t)$. Figure 3(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.

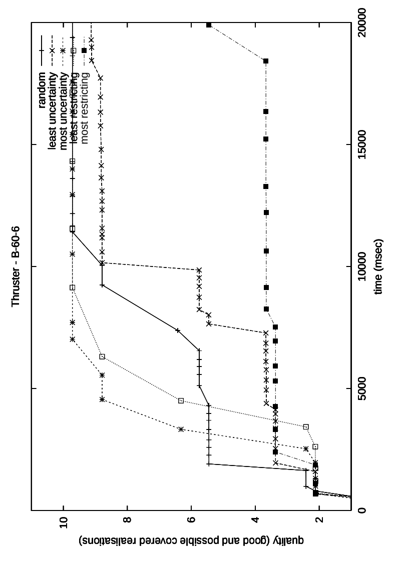
Figures 3(b)–4(b) demonstrate the performance of the heuristics for Thruster is more varied. For most instances of uncertainty, performance, and horizon, *least uncertainty* is the best heuristic and *random* is second or third. However, for some instances, *least uncertainty* does not have maximal $Q_1(t)$ for all t. First look at Figures 4(a)–4(b). 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. *most restricting* exhibits poor behaviour.

Next look at the rare result shown in Figure 3(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 classical CSPs, the choice of heuristic is itself heuristic.

The results for Tracker confirm those for AOCS. Figures 5(a)–5(b) show *least uncertainty* as the best heuristic. Note how it not only has a better performance profile, but also achieves a much earlier termination time than the other heuristics.

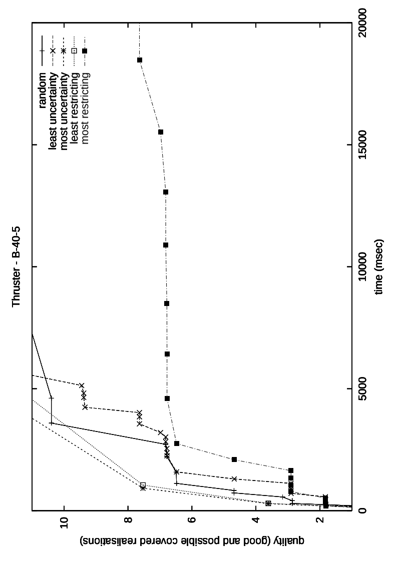


(a) AOCS C 80% horizon 8

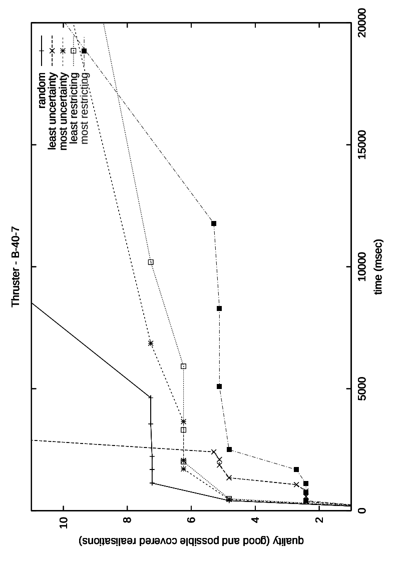


(b) Thruster B 60% horizon 6

Figure 3. Anytime behaviour of environment selection heuristics (I)

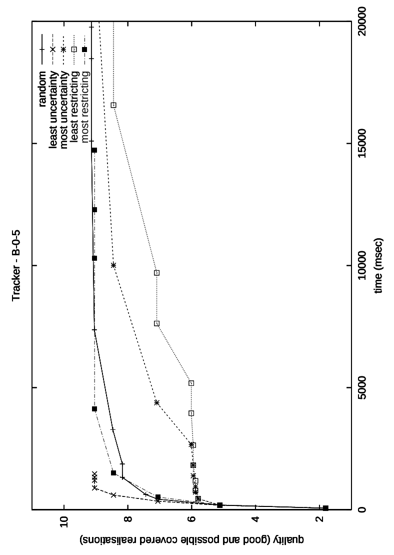


(a) Thruster C 40% horizon 5

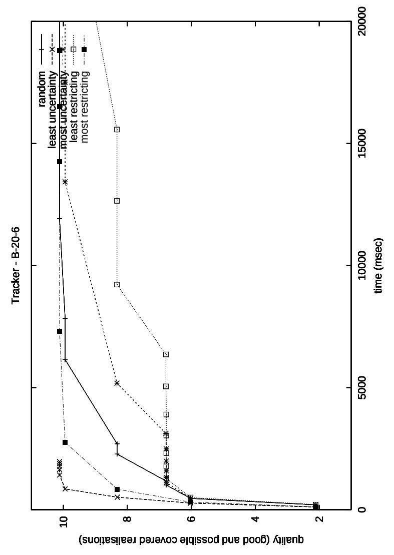


(b) Thruster B 40% horizon 7

Figure 4. Anytime behaviour of environment selection heuristics (II)



(a) Tracker B 0% horizon 5



(b) Tracker B 20% horizon 6

Figure 5. Anytime behaviour of environment selection heuristics (III)

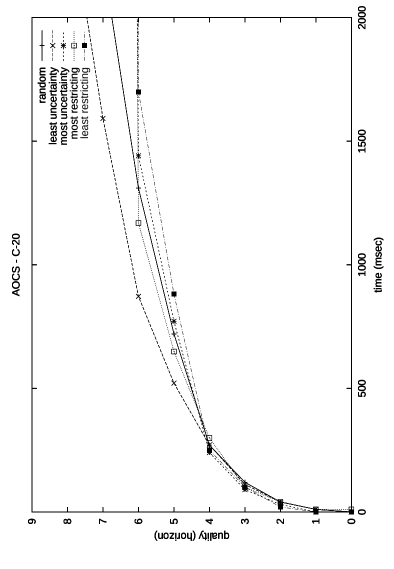
5.2 Incremental Horizon

We now consider the method described in Section 4.2. 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 $Q_2(t) = \max\{h | t_h \leq t\}$.

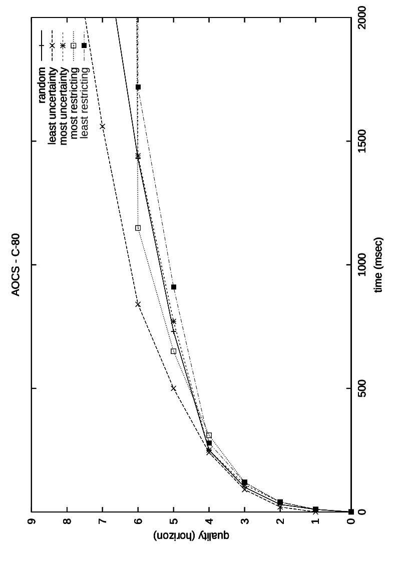
Figures 6(a)–8(b) show the quality (horizon attained) versus solving time (ms). The shape of the curves indicate that Algorithm 2 provides acceptable anytime behaviour. However, performance strongly depends on the environment selection heuristic. Since incremental horizon is built on decomp, this might 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 6(a)–6(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 7(a)–8(b)), the results are similar. The best heuristic is *least uncertainty*, and overall *random* is next best. For the Tracker instance A 20% (Figure 8(a)), beyond horizon 4, the remaining three heuristics struggle; *most uncertainty* is the best of them. For B 40% (Figure 8(b)), *random* and *least restricting* dominate about equally. The results for Thruster (Figures 7(a)–7(b)), while similar, show strongly that poor heuristics for environment selection give very poor

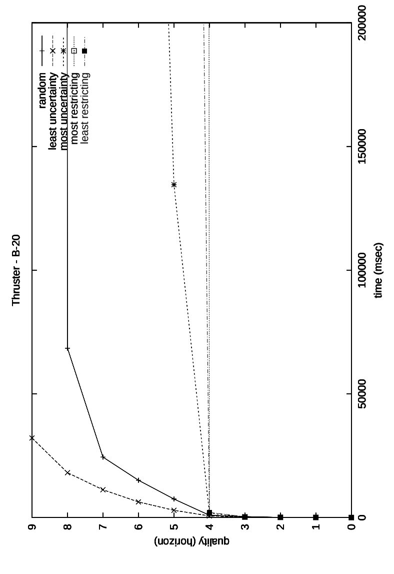


(a) AOCS C 20%

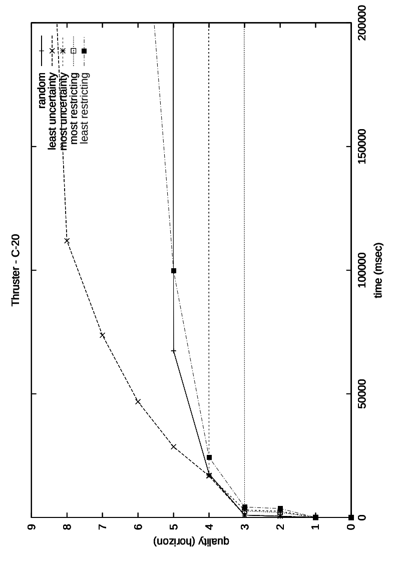


(b) AOCS C 80%

Figure 6. Anytime behaviour of incremental horizon (I)

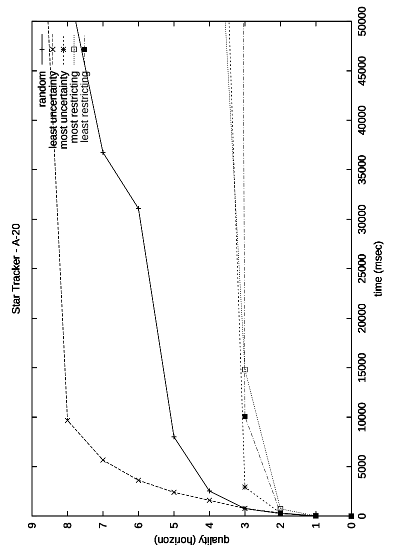


(a) Thruster B 20%

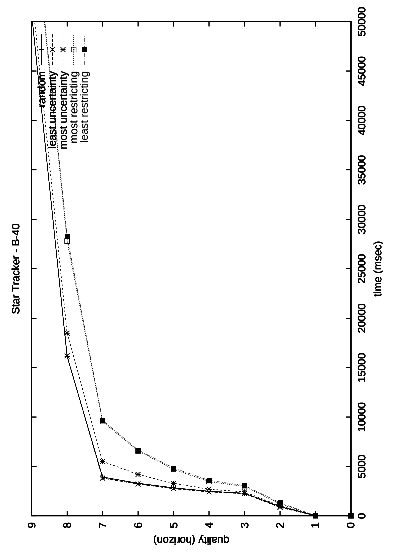


(b) Thruster C 20%

Figure 7. Anytime behaviour of incremental horizon (II)



(a) Tracker A 20%



(b) Tracker B 40%

Figure 8. Anytime behaviour of incremental horizon (III)

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*.

5.3 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(b)) 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* and the *first fail* (smallest domain first) variable selection heuristic for classical finite domain CSP. First fail is known as an often effective choice of variable selection heuristic [3, 11]. However, just as it is not the best heuristic for every CSP, so *least uncertainty* is not the best for every mixed CSP: Figure 3(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*. In particular, heuristics based on the size of variable domains (*most* and *least restricting*) vary in effectiveness between problem instances. For example, *most restricting* is acceptable in Figure 3(a) but very poor in Figure 4(a).

Thirdly, the results suggest that incremental horizon is effective in providing anytime behaviour, particularly for lesser horizons. When the subproblems 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.

6 Conclusion and Future Work

Anytime behaviour is an important requirement for the aerospace domain. Motivated by a planning problem for aerospace equipment control, this paper studied the anytime solving of 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.

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.

The incremental horizon method we considered is specialised for the SCP problem. However, the broader idea of problem 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.

Anytime algorithms for classical CSPs have been built by considering the CSP as a partial CSP, and using branch-and-bound or local search [17]. For finding robust 'super' solutions, anytime algorithms have also been built with branch-and-bound [8]. Anytime solving is related to incremental solving of CSPs (e.g. [16]). In the latter, however, the focus is to efficiently propagate the changes implied when a variable's domain changes.

In future work, we want to complete the investigation of incremental horizon by evaluating how often it produces plans for horizon H based on partial plans for a lower horizon, as described in Section 4.2. We would also like to evaluate the methods considered here on other SCPP instances (in particular, the hard Instrument instance described in [19]) and, importantly, on mixed CSPs arising from other problems.

The 'cross-over' between different heuristics over time suggest that meta-reasoning on the solving algorithm may yield the best anytime behaviour in practice. For instance, the hybrid approach suggested above. More generally, this reasoning can take into consideration [7, 9]: the current state of the solution (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, as noted earlier.

Driven by our motivational problem, in this paper we have considered only the full observability case; an interesting direction would be to consider anytime solving in the no observability case. Here, the outcome sought is a single robust solution that covers as many realisations as possible. As such, there are links not only to anytime methods for robust solutions to CSPs [8], but also to solving mixed CSPs with probability distributions over the parameters [4], which are an instance of the stochastic CSP framework [18]. For instance, the scenario sampling methods for stochastic CSPs give the opportunity for an anytime algorithm [10].

References

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

[2] M. S. Boddy and T. L. Dean. Solving time-dependent planning problems. In *Proc. of IJCAI'89*, pages 979–984, Detroit, MI, Aug. 1989.

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

[4] H. Fargier, J. Lang, R. Martin-Clouaire, and T. Schiex. A constraint satisfaction framework for decision under uncertainty. In *Proc. of UAI'95*, pages 167–174, Aug. 1995.

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

[6] E. C. Freuder and P. D. Hubbe. Extracting constraint satisfaction subproblems. In *Proc. of IJCAI'95*, pages 548–557, Montréal, Canada, 1995.

[7] E. A. Hansen and S. Zilberstein. Monitoring the progress of anytime problem-solving. In *Proc. of AAAI-96*, volume 2, pages 1229–1234, Portland, OR, Aug. 1996.

[8] E. Hebrard, B. Hnich, and T. Walsh. Super solutions in constraint programming. In *Proc. of CP-AI-OR'04*, pages 157–172, Nice, France, 2004.

[9] G. Le Lann. Time-utility scheduling and provably correct critical computer-based systems. In *Proc. of WPDRTS'04*, Santa Fe, NM, Apr. 2004.

[10] S. Manandhar, A. Tarim, and T. Walsh. Scenario-based stochastic constraint programming. In *Proc. of IJCAI'03*, pages 257–262, Acapulco, Mexico, Aug. 2003.

[11] K. Marriott and P. J. Stuckey. *Programming with Constraints: An Introduction*. MIT Press, Cambridge, MA, 1998.

[12] P. Morris, N. Muscettola, and T. Vidal. Dynamic control of plans with temporal uncertainty. In *Proc. of IJCAI'01*, pages 494–502, 2001.

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

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

[15] G. Verfaillie and M. Lemaître. Selecting and scheduling observations for agile satellites: Some lessons from the constraint reasoning community point of view. In *Proc. of CP'02*, pages 670–684, Sept. 2002.

[16] G. Verfaillie and T. Schiex. Solution reuse in dynamic constraint satisfaction problems. In *Proc. of AAAI-94*, pages 307–312, Seattle, WA, 1994.

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

[18] T. Walsh. Stochastic constraint programming. In *Proc. of ECAI-02*, Lyon, July 2002.

[19] N. Yorke-Smith. *Reliable Constraint Reasoning with Uncertain Data*. PhD thesis, IC-Parc, Imperial College London, Submitted Apr. 2004.

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