Changes'04

International Workshop on Constraint Solving under Change and Uncertainty

A CP2004 workshop, Toronto, Canada
September 27th, 2004

Problem solving under change and uncertainty is a significant issue for many practical applications. Solutions must be obtained before the full problem is known, and often must be executed as the environment changes. Many of the application areas have been tackled by constraint based methods, but current constraint solving tools offer little support for uncertain dynamic problems. Possible enhancements could include rapid reaction to problem changes, robust solutions, prediction of future changes and contingent solutions, time guarantees or exploitation of known time limits.

This workshop will continue the series of CP workshops on online solving, change and uncertainty, and in particular following on from Online-2003 held at Kinsale during CP2003. The workshop aims to bring together researchers interested in the general topic, to consider how existing techniques can be enhanced, and to explore combinations of the different techniques.

All submissions to the workshop were reviewed by at least two referees. Four papers were selected for full presentation, and are contained in these working notes. In addition, the notes contain a number of short position papers, which will be presented briefly in discussion sessions. We welcome you to the workshop, and look forward to a fruitful discussion.

Workshop Organisers

Chris Beck	University of Toronto, Canada
Ken Brown	Cork Constraint Computation Centre, Ireland
Gérard Verfaillie	RIA Research Group, LAAS-CNRS, France

Program Committee

Roman Barták	Charles University, Czech Republic
Amedeo Cesta	ISTC-CNR, Italy
Markus Fromherz	Parc, USA
Simon de Givry	INRA, France
Bill Havens	Simon Fraser University, Canada
Narendra Jussien	Ecole de Mines de Nantes, France
Ian Miguel	University of York, UK
Jon Spragg	Vidus Limited, UK
Thierry Vidal	ENIT, France
Toby Walsh	Cork Constraint Computation Centre, Ireland

Contents

Solver Learning for Predicting Changes in Dynamic Constraint Satisfaction Problems

Arnaud Lallouet[†*], Andreï Legtchenko[†], Éric Monfroy[‡], AbdelAli Ed-Dbali[†]

[†] Université d'Orléans – LIFO
BP 6759 – F-45067 Orléans – France
Firstname.Name@lifo.univ-orleans.fr
[*] Contact author

[‡] Université de Nantes – LINA
BP 92208 – F-44322 Nantes – France
Eric.Monfroy@irin.univ-nantes.fr

Abstract. We present a way of integrating machine learning capabilities in constraint reasoning systems by the use of partially defined constraints called *Open Constraints*. This enables a form of constraint reasoning with incomplete information: we use a machine learning algorithm to guess the missing part of the constraint and we put immediately this knowledge into the operational form of a solver. This approaches extends the field of applicability of constraint reasoning to problems which are difficult to model using classical constraints, and also potentially improves the efficiency of dynamic constraint solving. We illustrate our framework on online constraint solving applications which range from mobile computing to robotics.

1 Introduction

In the classical CSP framework, a constraint is a relation, a CSP is a conjunction of constraints and a solution is an element of the relation defined by the conjunction. The usefulness and the difficulty of this problem has deserved the huge amount of work which has been and remains to be done. But many problems do not perfectly fit in this framework and have been the origin of many extensions. Among them is the necessity to admit that a problem is not always perfectly defined. Some parts may be unknown for several good reasons: lack of information, incompleteness of human knowledge, no sufficient experience, uncertainty, belief, ... Also some part may change according to the environment, actions of external agents or movement in a mobile system.

One possibility is to accept belief revision when faced with a new piece of information. The most studied problem in this field is the one of constraint retraction in dynamic CSPs [8, 20] where one constraint can be deleted from the CSP. This gives more freedom to the problem but may invalidate the current computation state. Depending on the problem, the system may recompute a suitable consistent state or maintain solutions to be able to react to further modifications.

In this paper, we tackle a different way of dealing with incomplete information: we leave the possibility for a constraint to be partially defined. Clearly, this contradicts the Closed World Assumption made in the usual definition of

a constraint where what is not stated as true is mandatorily false. An *Open Constraint* is composed of two disjoint parts: one is positive and contains the tuples for which we are certain that they are true, and the other is negative and contains the tuples for which we are certain that they are false. Other tuples are simply unknown.

Since an open constraint actually represents a hidden reality, there is many ways to classify unknown tuples. But, if the well-defined part is sufficient, there is a chance that an automatic algorithm can detect regularities which can help to reconstitute the hidden constraint. We argue that this definition allows to integrate Machine Learning [17] or data-mining capabilities inside constraint reasoning. Among possible uses, we can cite the possibility of building reasoning systems from incomplete knowledge, predicting changes in dynamic constraint solving, self-improving constraint system, intelligent constraint reasoning modules in multi-agent systems or web-based distributed reasoning systems.

In section 2, we present the basics of open constraints. Section 3 proposes a general framework for open constraint solver construction and two methods for building solvers for open constraints. But let first start by a motivating example.

A smart agenda for PDAs

Let us introduce the usefulness of open constraints on a mobile computing application. In organizations in which different peoples collaborate, meetings can be planned in a non-hierarchical way by any member of the group according to the needs. The process is easy when people are at voice distance or in a centralized setting (groupware) but becomes tricky when the information is distributed on nomadic applications, for example hosted in PDAs. Then, many information exchanges (synchronizations) are needed in order to get a consistent solution since only couples of PDAs are able to synchronize. Moreover, synchronization is costly since the PDAs are supposed to be close to each other and people in the same room at the same time. For privacy reasons, we suppose that PDAs do not exchange information about third parties when synchronizing.

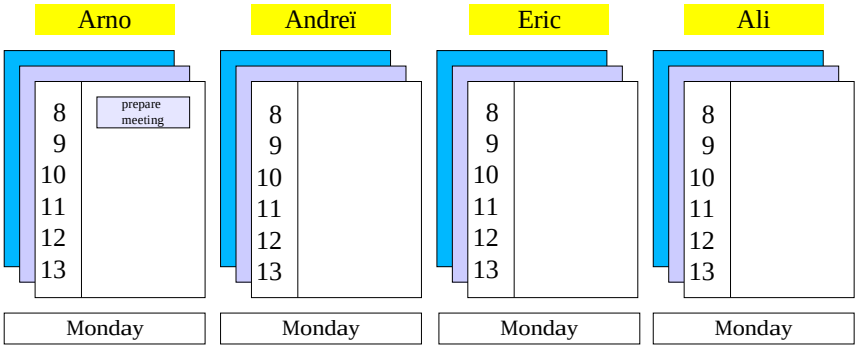


Fig. 1. Constraints in Arno's PDA before synchronization.

Let us consider a group of four persons: Arno, Andreï, Eric and Ali, each one holding a digital assistant. One day, Arno decides to organize a meeting with the three others on monday. Every PDA holds a representation for the other's schedule. Since the PDAs are not synchronized, Arno holds no constraint for the other's schedules (see figure 1). If a schedule for monday morning is represented by a constraint `schedule(hour,affected)`, it is equivalent to have a constraint with all tuples allowed, i.e. $\{(8,0),(8,1),(9,0),(9,1),\dots\}$. In other words, we do not know if an hour is affected or not. At the beginning, all known tuples for the constraints come from the past weeks. The current week tuples are not yet in the constraint database since no synchronization has occured. A closed-world Constraint Programming system would propose 9:00 for the starting time of the meeting by finding the first available solution.

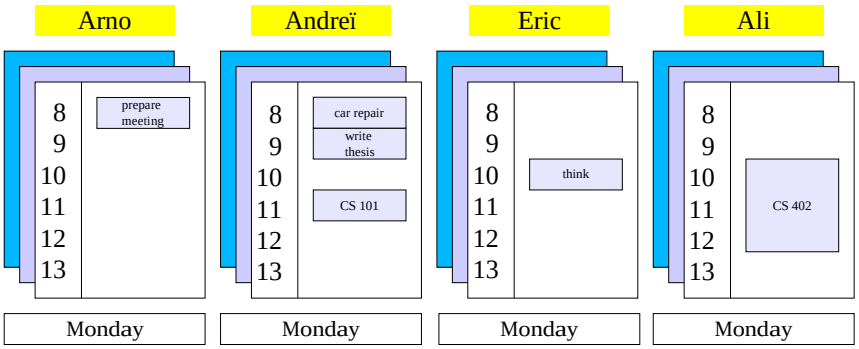


Fig. 2. Constraints after synchronization with Andreï, Eric and Ali.

When synchronizing with Andreï, Eric and Ali, conflicts will possibly appear, like for example thoses of figure 2 where actual schedules are described. A conflict yields a solver's revision and the need for re-synchronizing with previously synchronized PDAs. A possible scenario for this tentative meeting is the following:

Synchronization	**First solution found**
no	9:00
Arno - Andreï	10:00
Arno - Eric	12:00
Arno - Ali	13:00
Arno - Andreï	13:00
Arno - Eric	13:00

Five synchronizations and three revisions are needed in order to get a globally consistent schedule. For n peoples, if downloaded schedules remain stable until the meeting hour is eventually discovered, it needs up to $2n$ synchronizations for a single meeting. In a real-life situation, the n peoples may initiate their own meeting and change their timetable unexpectedly, thus increasing the whole number of synchronizations.

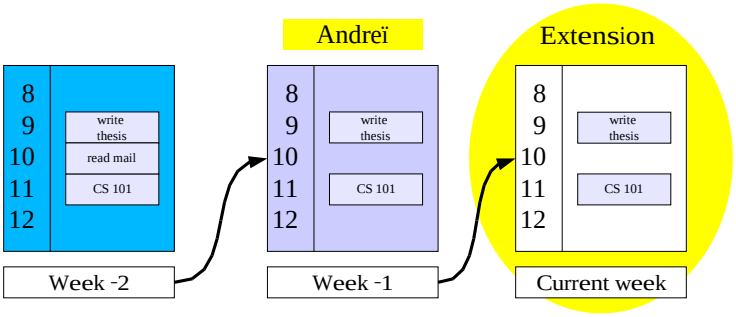


Fig. 3. Extension of Andreï's constraint as guessed by Arno's PDA.

But it is commonsense knowedge that other people's schedules are not empty but simply unknown. In order to improve this mechanism, let us consider the unknown schedules as open constraints. Our purpose is to find a suitable extension (guess of the missing tuples) which can be more informative than using the closed-world assumption. Over time, we do have informations about the habits of people we work with, as it goes in real life. A very simple extension of the unknown constraints may be found by reproducing the schedule of the last week, or by a simple analysis of recent past. In figure 3 is represented Andreï's schedule for the last two monday mornings. Here, for simplicity, the extension is built by carrying forward the events which occured in the two previous weeks. By doing the same with Eric and Ali's constraint, few or no revisions may appear. This clearly improves the system with respect to its most critical part, synchronization.

2 Open Constraints

Modeling a problem with constraints involves three different levels:

- *The model* which consists in constraints definitions. Usually the user is provided a constraint language in which each constraint is a building block with its own precise definition. A model is built by putting together the constraints in order to describe the problem. Sometimes, some constraints (for example compatibility constraints in frequency allocation or in configuration problems) which are impossible to describe in the language or have a clumsy formulation are given by a table. Theses constraints usually affect the performances of the resolution.
- *The solver* which is the definition of how the constraints react to events caused by search or other constraints and propagate information across the network. Following [3], we describe a solver by a set of operators whose chaotic iteration enforce a given consistency[1]. Many solvers implement only

[1] There are many other algorithms for solving CSP (see [9]) but the chaotic iterations framework allows to easily define an individual solver for a constraint.

the set of built-in constraints but some of them propose a language to express propagators for user-defined constraints. Among them, two languages have received a great attention: Indexicals [19] and Constraint Handling Rules (or CHRs) [12].

– *The computation state* which describes the current state of variable domains. Some states for which propagation is exhausted are said consistent and are starting points for the search mechanism.

Let V be a set of variables and $D = (D_X)_{X \in V}$ their (finite) domains. For $W \subseteq V$, we denote by D^W the set of tuples on W, namely $\Pi_{X \in W} D_X$. For $A \subseteq D^W$, $\overline{A}$ denotes its complementary in D^W.

A constraint c can be defined as a couple $c = (W, T)$ where $W = var(c) \subseteq V$ is its *arity* and $T = sol(c) \subseteq D^W$ is its *solution*. A constraint is supposed to be fully known by completing its definition with the so-called *Closed World Assumption* (or CWA) stating that what is not explicitely declared as true is false. Hence the complementary $\overline{T}$ is the set of tuples which do not belong to c. Alternatively, a constraint can be described by giving its forbidden tuples, with the same meaning. In the following, we call ordinary constraints under CWA *closed* or *classical* constraints. When dealing with incomplete information, it may happen that some parts of the constraint are unknown. We call such a partially defined constraint an *open constraint*:

Definition 1 (Open constraint).
An open constraint *is a triple* $c = (W, c^+, c^-)$ *where* $c^+ \subseteq D^W, c^- \subseteq D^W$ *and* $c^+ \cap c^- = \emptyset$.

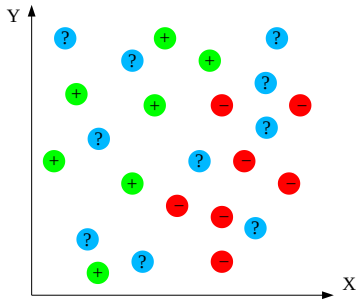


Fig. 4. An open constraint.

In an open constraint $c = (W, c^+, c^-)$, c^+ represents the tuples which are certainly allowed and c^- the ones which are certainly forbidden. The rest of the tuples, i.e. $\overline{c^+ \cup c^-}$ are simply unknown. An open constraint is depicted in figure 4 where the (green) dots labeled "+" represent tuples of c^+, the (red) dots labeled "-" represent tuples of c^- and the (blue) dots labeled "?" represent unknown tuples. We denote by $\mathcal{OC}$ the set of open constraints.

Remark 2. Note that a classical constraint $c = (W, T)$ is a particular open constraint for which the negative part is the complementary of the positive part. Thus, in the following, the notation (W, T) can be viewed as a shorthand for $(W, T, \overline{T})$.

Example 3 (Examples of open constraints).

- a bank customer database includes a relation:
  `good_customer(revenue,balance,#incident)`
  which defines what is a good customer. This constraint is not known in extension because it would need to define all the values for the fields. From one hand, giving this definition could be cumbersome and error-prone. Moreover, real customer databases include many more fields like professional activity, number of cars owned ..., making thus the whole relation intractable in extension.
- in picture processing, recognition of a figure in a picture coming from a camera can be seen as a high arity constraint (for example 100×100). For example, the constraint `circle`$(X_1, \ldots, X_{10000})$ can be used by a robot to detect if the current picture contains a circle.
- time-series data are by definition incomplete since they keep growing over time. They can be represented by an open constraint which can be used by a planning system in order to decide actions guided by experience.
- logs obtained by monitoring a system may be analyzed in order to find some patterns of interest like a probable attack.

Example 4 (Sales assistant).
Consider a robot sales assistant whose task is to give back change to customers. We model its knowledge by a constraint which states how many coins of different types have to be given to make a given sum, for example:

€/100	1 ¢	5 ¢	10 ¢	25 ¢
100	5	2	6	1

At the beginning, this constraint which models all different possibilities to give change for a given sum is unknown ($c = (W, \emptyset, \emptyset)$) and the assistant has to be supervised during a little training period before its first interactions with customers.

An open constraint can be viewed as to represent a set of closed constraints which are compatible with its positive and negative parts. Each of these constraint "extends" the partial definition given by the open constraint. We call such a compatible closed constraint an *extension* of the open constraint:

Definition 5 (Extension).
Let $c = (W, c^+, c^-)$ be an open constraint. A (classical) constraint $c' = (W, T)$ is an extension *of c if $c^+ \subseteq T$ and $c^- \subseteq \overline{T}$.*

In general, many extension can be considered, and let us introduce three of them. Among all possible extensions lies the real closed constraint which is associated

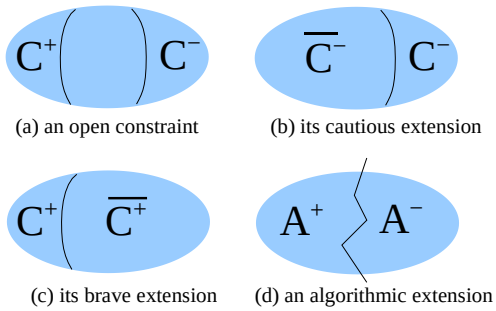


Fig. 5. An open constraint and some of its extension.

to the real world problem. In most cases, the knowledge of this constraint is impossible to get and all that can be done is computing an approximation of it. Let $c = (W, c^+, c^-)$ be an open constraint. We denote an extension of c by $[c]$.

- *Cautious* extension: $[c]_c = (W, \overline{c^-})$. All unknown tuples are assumed to be true (figure 5b).
- *Brave* extension: $[c]_b = (W, c^+)$. All unknown tuples are assumed to be false (figure 5c).
- *Algorithmic* extension $[c]_{\mathcal{A}}$: let $\mathcal{A} : D^W \rightarrow \{0, 1\}$ be a tuple classification algorithm such that $t \in c^+ \Rightarrow \mathcal{A}(t) = 1$ and $t \in c^- \Rightarrow \mathcal{A}(t) = 0$. Then $[c]_{\mathcal{A}} = (W, \{t \in D^W \mid \mathcal{A}(t) = 1\})$ (figure 5d).

The algorithmic extension of an open constraint allows a great freedom because a machine learning algorithm can be used to classify the missing tuples. We propose first to study the impact of the choice of the extension on the solver's behavior. Depending on the chosen extension, the solver will prune more or less agressively the search space:

- *Cautious extension*: a solver generated according to this extension is cautious in the sense that it will not prune the search space for any unknown tuple.
- *Brave extension*: a solver generated according to this extension will prune the search space as soon as possible. Actually, it behaves exactly as for a closed constraint for which all non-allowed tuples are disallowed.
- *Algorithmic extension*: this last kind of extension will be the ideal host for a learning algorithm. Tuples from c^+ and c^- are respectively positive and negative examples which are used to feed the learning algorithm. Note that the two preceding extensions are particular cases of this one for constant functions. The main challenge is to be able to generate the best possible solver: the one which has a strong pruning power and is not subject to many weakening revisions. As a learning task, there is no universal solution for every problem and the user has to carefully choose and tune his/her learning algorithm in order to obtain good results. For example, in figure 6 is depicted 2 possible extensions for the open constraint of figure 4. Note that two unknown points are put in different classes in these two extensions

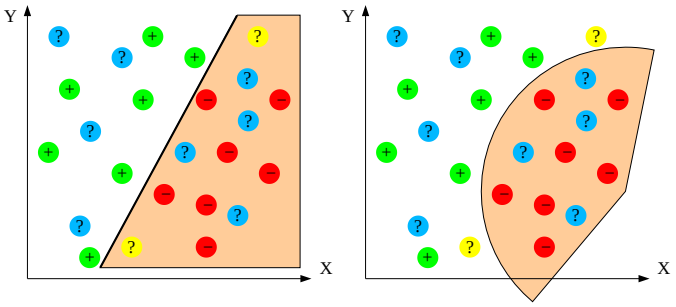


Fig. 6. Two possible algorithmic extensions.

(they are colored in yellow for those who can print in colors, but are easy to find otherwise).

3 Solvers for Open Constraints

In this section, we propose two techniques for building generalizing solvers from an open constraint.

3.1 Overview of open constraint solver construction

The notion of solver we use in this paper comes directly from the framework of chaotic iterations [3] applied to domain reduction. In this approach, a solver is modeled by a set of reduction operators for each constraint. In order to model a given consistency, these operators must have the following properties:

- monotonicity.
- contractance: they should reduce the domains.
- correction, meaning that no solution tuple could be rejected at any time anywhere in the search space.
- singleton completeness, meaning that the operator is a satisfiability test for the positive examples and rejects the counter-examples.

In some implementations (like indexicals [19]), each variable has its own reduction operator. Singleton completeness holds for the set of operators associated to a constraint. Note that singleton completeness for an open constraint is an extension of singleton completeness for closed constraints [14]. With this property on singletonic states, the consistency check for a candidate tuple can be done by the propagation mechanism itself.

Besides that, there is many ways to classify the unknown part while preserving examples and counter-examples in their respective category. The following proposition allows to limit the efforts made to verify that an operator is a consistency. It is sufficient to verify if the operator preserves the defined part of c (c^+ and c^-):

Proposition 6.
If f is monotonic, contractant and singleton complete, then f is correct.

The proof of this proposition can be derived from a similar result for closed constraints presented in [14]. When completed, an open constraint can be given a solver, but even if correct with respect to the open constraint, this solver may be incorrect or incomplete with respect to the real hidden constraint. This depends on the precision of the extension.

Building a solver for an open constraint is not an easy task since it needs to build propagators for a partially unknown constraint. This can be done by hand and the propagators will reflect the extension intended by the programmer. But given a large constraint arity and a large set of examples and counter-examples, it is likely that most regularities which could be useful for propagation will remain unnoticed, thus leading to poor performances. A better way is to find how to use a learning algorithm in order to build propagators.

One could imagine two methods for building automatically a solver for an open constraint: the first one consists in finding the desired extension of the constraint, and then to build the solver using one of the previous techniques. But this process may be cumbersome since examples and counter-examples are likely to come from a large database. Extending it would result in a larger set of data which may soon become intractable. Hence a better solution is to build the solver and the extension on the fly:

> *Our proposition consists, by using learning techniques, in building a set of reduction operators such that its associated constraint covers the positive examples and rejects the counter-examples.*

In other terms, we build the solver instead of building the constraint because a constraint is fully defined by its consistency operator (or its set of domain reduction operators). In the following, we propose two techniques aiming at building solvers for open constraints. The first one is an extension of our solver construction technique: the Solar system [14]. The second one proposes a novel technique which uses a classifier like an artificial neural network in order to build an operator enforcing a consistency.

3.2 Indexical solver construction

Indexicals have been introduced in [19] in order to define finite domains constraint solvers. The Solar system automatically finds an indexical expression for a constraint given in extension. The possible expressions are restricted by a language bias and we refer to [14] for a more complete description of the system.

Let $c = (W, c^+, c^-)$. Here we present the automatic construction of one linear indexical for a variable $X \in W$. The general form of the indexical is `X in min`l_X `..` `max`l_X, both bounds being linear and defined as follows:

$$a + \sum_{Y \in W \setminus \{X\}} (b_Y . \min(Y) + c_Y . \max(Y))$$

With such a fixed form, the learning problem amounts to find the best coefficients for the template expression. Since there is no need for completeness, we just use the reduction indexicals provided by the system and not the reparation ones which are built only to ensure completeness.

Example 7 (Example 4 running, the sales assistant). At first, the constraint is empty and thus its solver does nothing and accepts all tuples. Let us provide to the system a first training example in order to build an effective solver. For conveniency, we give name to the variables, we restrict the arity of the constraint to 3 and we compute the solver only for the variable A:

A = €/100	B = 10 ¢	C = 25 ¢
60	1	2

With this only tuple provided, our system outputs the following solver:

```
A in -100 -100*min(B) -100*max(B) +280*min(C) -100*max(C) ..
     -100 -100*min(B) -100*max(B) +280*min(C) -100*max(C)
```

This is not the expected knowledge for a change assistant but the tuple belongs to this hyperplane. Since the hypothesis space has 5 dimensions, it needs 6 tuples to find the hyperplane. Let the next five customers ask for the following change: $(10, 1, 0)$, $(75, 0, 3)$, $(75, 5, 1)$, $(25, 0, 1)$ and $(185, 1, 7)$. This time the expected solver is produced:

```
A in 10*min(B) +25*min(C) .. 10*max(B) +25*max(C)
```

Since the actual constraint of this example is linear, our system will eventually find it, as it does for built-in constraints. The trained solver may now be put in a CSP, for example with resource constraints which limit the amount of coins of different types in the cashier's office: for example $B \leq 100, C \leq 80$.

3.3 Classifier-based solver

Given a relation, a classifier [17] is a machine learning algorithm which aims at finding a class for an attribute knowing the value of the others. For example, in a mushroom database with many observational attributes like hat, foot or color, the purpose may be to guess if the mushroom is toxic or can be eaten. Classifiers have nice properties: they are tolerant to noise, they generalize properties and can be used to predict the class of new items and they are often fast to execute. Several knowledge representations may be used to implement classifiers, like for example artificial neural networks (ANN) or binary decision trees. As is, a classifier is pretty much different from a solver. But a solver for open constraints would take advantage of these properties if the idea underlying a classifier could be used to build it. We present here an example of classifier-based solver which uses an ANN.

Let $r = (W, T)$, $X \in W$ and $a \in D_X$, we denote by $r[x(a)] = \{t|_{W-\{X\}} \mid t \in r \bowtie (\{X\}, \{a\})\}$ the subtable composed of those tuples of $sol(r)$ for which their value on X is a.

Example 8. Let r be the relation $(\{X, Y, Z\}, \{(0, 0, 1), (0, 1, 1), (1, 1, 0)\})$. The selection $r[x(0)] = (\{Y, Z\}, \{(0, 1), (1, 1)\})$.

In order to build a reduction operator for X for the constraint $c = (W, c^+, c^-)$, the idea is to associate a boolean classifier to each value a of X's domain which says if a can be pruned. The classifier takes as input the domain of the other variables. It is trained with $c^+[x(v)]$ as positive examples and $c^-[x(v)]$.

Example 9. Let $c^+ = (0, 0, 1)$ and $c^- = (0, 1, 0)$. For this example, we use a single perceptron to classify the tuples supporting the value $x(0)$. The result of training is depicted in figure 7a and its extension in figure 7b. By adding the tuple $(0, 1, 1)$ to c^-, it shifts the line as in figure 7c, yielding the extension given in figure 7d.

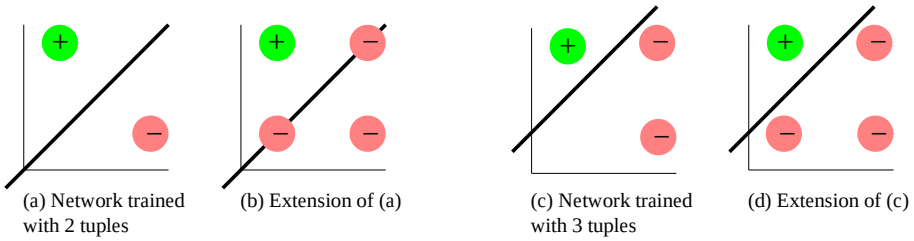


Fig. 7. An ANN propagator.

The full solver is obtained by taking the union of the operators defined by a classifier for each value of each variable. The space complexity obtained by such a solver is comparable to the one of AC4 [18], each of the classifier having a constant size. Once trained, the ANN can be used as follows: for checking one tuple, for example $(0, 0, 0)$, 0 is presented on Y and Z's neurons and the answer is 0 meaning that 0 is not a supported value (see figure 7d and figure 8a).

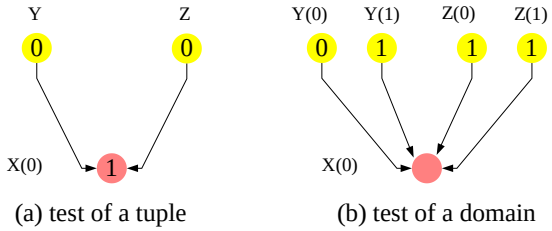


Fig. 8. Using nn ANN propagator.

For checking intervals or subsets in order to implement bound- or arc-consistency, it is needed to have a different network composed of one neuron per value of the

domain of the input variables. Then, in order to possibly remove the value 0 from X's domain, all domain values for Y and Z are simultaneously presented on input variables (figure 8b). The neuron is simply feeded with a representation of the input variables domain. If the network is monotonic and if $x(0)$ is supported by one tuple, the output value is 1. Conversely, if $x(0)$ is not supported, the output is 0. Thus propagation is done in a safe and efficient way.

If the operator coded by the network is not monotonic, then all tuples in the Cartesian product of the domains of the input variables have to be checked while their output is 0 (which means that they do not support the value). Once a 1 is produced, this means that the value is supported and the enumeration may be shorten. Of course this technique could be applied only for very small domains. Note that this technique applies even if the reduction operator is defined as usual and not by learning — by a programmer for example. Hence we insist on using monotonic operators.

We have implemented a larger prototype using ANN constraint-based planning of a robot in a virtual environment. To be more expressive, perceptrons have to be replaced by multilayer ANN in order to express non-linearly separable functions. One problem with ANNs is that they are non-monotonic by nature since some connexion weights may be negative. The way we chose to overcome this problem is to constrain the weight to be positive. It has the effect of sometimes preventing the retropropagation learning algorithm to converge towards the optimum. This is why we set the weight by using a genetic algorithm. Convergence is slower but the quality of solutions is satisfactory.

The robot dwells in a grid environment, can perform actions like make a step forward, turn left or eat energy pill. It has state variables representing location and energy. Each time the robot bumps a wall, it looses energy. The robot's representation of the world is modeled internally by a constraint `step(OldState, Movement, NewState)` which relates its perception before and after its actions. The known part of this constraint changes at every move and the robot learns from its past actions a probable description of what it should do next. The robot uses this constraint for a 3-steps simple planning (see figure 9). It searches in the space of possible moves the best one according to the current extension of the constraint and performs this move. Each time the move leads to a bad outcome (lower the energy, for example), the robot revise its solver for the constraint by re-running the learning algorithm. In other terms, it changes the computed extension of this constraint. We observed that after a first period of intense revision (when only a few tuples are known), the robot improves itself and is eventually able to show an emerging "intelligent" behavior by avoiding obstacles. This is done without having an internal iconic representation.

4 Related work and open issues

Related work. Many approaches have been tackled to handle changes in CSPs, especially in the so-called Dynamic CSP framework [8]. We are indebted to [20] for their excellent survey on dynamic constraint solving in which the reader can

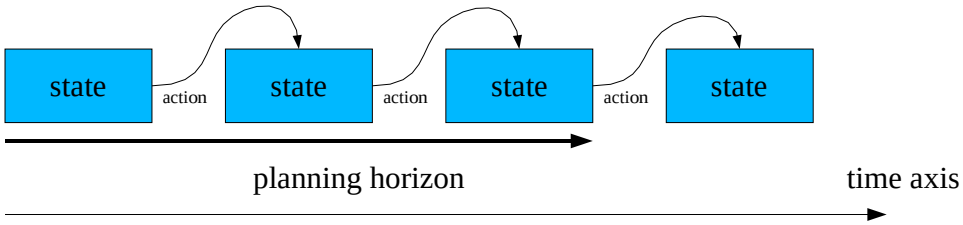


Fig. 9. Robot planning.

find many more references than it is possible to include in this paper. Interestingly, the $AC|DC$ algorithm [5] can cope with the enlargement of the domain of variables or adding of tuples to constraints. However, the algorithm works at the computation state level and outputs a new consistent search state after revision and no consideration on solvers revision is made.

The name "open constraint" has been coined by [11] but with a rather different treatment as it is proposed in this paper. They consider CSPs for which the solving algorithm may query variable values when the search space is exhausted and no solution is found. With the artifact of hidden variable encoding, the solving algorithm *fo-search* gathers tuple by tuple the definition of the constraint and the main consideration is to limit the number of gathered tuples. The same intention can be found in the Interactive Constraint Satisfaction framework proposed in [15]. In contrast, our concern is to build a solver and not to provide an algorithm for distributed CSPs.

Other works introduce machine learning in Constraint Programming like the Adaptative Constraint Engine [10]. The purpose is different of ours since learning is used to infer strategies for a class of problems.

Several techniques have been considered for the automatic construction of a constraint solver since the pioneering work of [4]. The main difference between approaches is the choice of the representation language for operators. In [1] is introduced the system PropMiner which constructs as CHR all relevant propagation rules for a constraint given a language for the left- and right-hand sides of the rules. We believe that this framework could be extended for open constraint in the context of association rules mining [2]. In [7, 14] is presented a general framework for consistency approximation, instantiated by the automatic construction of indexicals-based solvers for bound-consistency. It has been extended for arc-consistency by using clustering techniques [13] and delay of expensive operators [16]. Our current approach mainly focuses on this framework in order to extend it for open constraints.

In [6], the goal is to find a CSP made out of built-in constraints like $\leq$, $=$, $\neq$ which accepts examples and rejects counter-examples. The purpose and methods are different since they want to help a CSP designer to build the model he/she wants to solve but it can also be understood as finding a model for an open constraint. Since they use (combination of) symbolic constraints, they can use a powerful version space learning algorithm. As with many learning algorithms,

they have to put strong languages biases in order to get a tractable problem. In a solver learning perspective, this representation of an open constraint as a CSP has the advantage of reusing existing solvers for built-in constraints.

Open issues. Building a native solver for open constraints is related to the treatment of uncertainty. For example, the constraint $X < Y \pm 1$ is composed of a positive part $X < Y + 1$, a negative part $X \geq Y - 1$ and an unknown part in between. Such a solver could provide information about the number of uncertain constraints used in a solution. It could be also useful to express preferences and/or to address over-constrained problems. The unknown area of the constraint has to be small in order to find some certain solutions. Uncertainty in Constraint Programming has been tackled in [21] among others.

Also open constraints pose the problem of the solver's correctness, which is one of the most interesting feature of the closed-world approach. In order to generalize, a learning algorithm should be incomplete by accepting more tuples than only the set of positive examples. But it can also be incorrect and this behavior is useful to be tolerant to noisy or incomplete data. We believe that applications which need open constraints also need the full power of learning tools even if some solutions found are false positive or negative.

5 Conclusion

Future knowledge-based system will require reasoning capabilities, reactivity to the external world and self-improvement. We propose partially defined or *open constraints* as a model for constraint reasoning with incomplete information. We have proposed a way of building a solver for an open constraint by combining a learning algorithm and a solver generation mechanism. Eventually, open constraints will play for machine learning techniques the same role as global constraints do for specialized graph or mathematical programming algorithms.

Acknowledgements. This project is supported by French CNRS grant 2JE095. The authors would like to thank Lionel Martin for valuable suggestions.

References

1. Slim Abdennadher and Christophe Rigotti. Automatic generation of rule-based constraint solvers over finite domains. *Transaction on Computational Logic*, 5(2), 2004.
2. Rakesh Agrawal, Tomasz Imielinski, and Arun Swami. Mining association rules between sets of items in large databases. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 22(2):207–216, June 1993.
3. K.R. Apt. The essence of constraint propagation. *Theoretical Computer Science*, 221(1-2):179–210, 1999.
4. K.R. Apt and E. Monfroy. Automatic generation of constraint propagation algorithms for small finite domains. In Joxan Jaffar, editor, *International Conference on Principles and Practice of Constraint Programming*, volume 1713 of *LNCS*, pages 58–72, Alexandria, Virginia, USA, 1999. Springer.

5. P. Berlandier and B. Neveu. Maintaining arc-consistency through constraint retraction. In *International Conference on Tools with Artificial Intelligence*, pages 426–431, New Orleans, LA, USA, 1994. IEEE.
6. R. Coletta, C. Bessière, B. O'Sullivan, E. C. Freuder, S. O'Connell, and J. Quinqueton. Semi-automatic modeling by constraint acquisition. In Francesca Rossi, editor, *International Conference on Principles and Practice of Constraint Programming*, number 2833 in LNCS, pages 812–816, Kinsale, Ireland, 2003. Springer.
7. Thi-Bich-Hanh Dao, Arnaud Lallouet, Andrei Legtchenko, and Lionel Martin. Indexical-based solver learning. In Pascal van Hentenryck, editor, *International Conference on Principles and Practice of Constraint Programming*, volume 2470 of *LNCS*, pages 541–555, Ithaca, NY, USA, Sept. 7 - 13 2002. Springer.
8. R. Dechter and A. Dechter. Belief maintenance in dynamic constraint networks. In *7th National Conference on Artificial Intelligence*, pages 37–42, St Paul, MN, USA, 1988. AAAI Press.
9. Rina Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.
10. Susan L. Epstein, Eugene C. Freuder, Richard Wallace, Anton Morozov, and Bruce Samuels. The adaptative constraint engine. In Pascal Van Hentenryck, editor, *International Conference on Principles and Practice of Constraint Programming*, number 2470 in LNCS, pages 525–540, Ithaca, NY, USA, 2002. Springer.
11. Boi Faltings and Santiago Macho-Gonzalez. Open constraint satisfaction. In Pascal van Hentenryck, editor, *International Conference on Principles and Practice of Constraint Programming*, volume 2470 of *LNCS*, pages 356–370, Ithaca, NY, USA, Sept. 7 - 13 2002. Springer.
12. Thom Frühwirth. Theory and practice of Constraint Handling Rules. *Journal of Logic Programming*, 37(1-3):95–138, 1998.
13. Arnaud Lallouet, Andreï Legtchenko, Thi-Bich-Hanh Dao, and AbdelAli Ed-Dbali. Intermediate (learned) consistencies. In Francesca Rossi, editor, *International Conference on Principles and Practice of Constraint Programming*, number 2833 in LNCS, pages 889–893, Kinsale, County Cork, Ireland, 2003. Springer.
14. Arnaud Lallouet, Andreï Legtchenko, Thi-Bich-Hanh Dao, and AbdelAli Ed-Dbali. Learning approximate consistencies. In Krzysztof R. Apt, Francois Fages, Francesca Rossi, Péter Szeredi, and József Váncza, editors, *CSCLP'03: Recent Advances in Constraints*, LNAI 3010, pages 87–106. Springer, 2004.
15. E. Lamma, P. Mello, M. Milano, R. Cucchiara, M. Gavanelli, and M. Piccardi. Constraint propagation and value acquisition: Why we should do it interactively. In *International Conference on Artificial Intelligence*, Stockholm, Sweden, 1999.
16. Andreï Legtchenko, Arnaud Lallouet, and AbdelAli Ed-Dbali. Intermediate consistencies by delaying expensive propagators. In Valerie Barr and Zdravko Markov, editors, *Flairs'04, International Florida Artificial Intelligence Conference*, South Beach Miami, FL, USAi, 2004. AAAI Press.
17. Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.
18. Roger Mohr and Thomas C. Henderson. Arc and path consistency revisited. *Artificial Intelligence*, 28(2):225–233, 1986.
19. P. van Hentenryck, V. Saraswat, and Y. Deville. Constraint processing in cc(fd). Technical report, Computer Science Department, Brown University, 1992.
20. Gérard Verfaillie and Narendra Jussien. Dynamic constraint solving, 2003. CP'2003 Tutorial.
21. Neil Yorke-Smith and Carmen Gervet. Certainty closure: A framework for reliable constraint reasoning with uncertainty. In Francesca Rossi, editor, *9th International Conference on Principles and Practice of Constraint Programming*, number 2833 in LNCS, pages 769–783, Cork, Ireland, 2003. Springer.

Feedback Control for Real-Time Solving

Ying Lu [1], Lara S. Crawford [2*], Wheeler Ruml [2] and Markus P.J. Fromherz [2]

[1] Department of Computer Science, University of Virginia
Charlottesville, VA 22903
ying@cs.virginia.edu

[2] Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304
{lcrawford, ruml, fromherz}@parc.com

Abstract. Numerous solvers have been proposed to solve constraint satisfaction problems (CSPs) or constrained optimization problems (COPs). Research has demonstrated that solvers' performance is instance-dependent and that no single solver is the best for all problem instances. In this paper, we further demonstrate that solvers' relative performance is time-dependent and that, given a problem instance, the best solver varies for different solving time bounds. We investigate an on-line feedback control paradigm for solver or problem reconfiguration so that the solver can reach the best possible solution within a specified time bound. Our framework is unique in specifically considering the time constraint in the feedback control of solving. With this augmented time-adaptivity, our paradigm improves solver performance for real-time applications. As a case study, we apply the feedback control paradigm to real-time performance control of a multidimensional knapsack problem solver.

1 Introduction

Given a problem instance, some solvers or solver configurations perform vastly better than others. In the literature, much research has tried to provide guidance for matching the right solver to a problem instance. Minton [24] pointed out that the performance of solvers is instance-dependent, i.e., for a given problem class a solver can perform well for some instances, but poorly for others, which makes the matching very difficult. Many authors have used off-line analysis (based on statistics or problem characteristics available before actually beginning to solve an instance) or probing to optimize algorithms or heuristics for a particular class of problems or even a particular instance. Others have used information acquired during a solving run to iteratively tune the solving process in a type of feedback loop.

For many real-world problems, a hard or soft time constraint is imposed on the solving process. The solvers have to be terminated within certain time

* Corresponding author.

bounds in order to provide acceptable service. Although previous work has always attempted to improve solving efficiency, it has almost never explicitly taken the time bound into account when selecting solvers, heuristics, or parameter values. We demonstrate in this paper that the best solver choice is dependent on the time constraint and propose an on-line feedback control framework that will adapt the solving process to the problem instance as well as to the real-time application requirement.

2 Related Work

There is a large body of literature on off-line adaptive problem solving. A number of systems (see Minton [24], Gratch and DeJong [15, 14], and Caseau, Laburthe, and Silverstein [8]) have used off-line analysis to optimize algorithms or heuristics for a particular class of problems. This approach can be seen as analogous to designing an open-loop controller, in the sense that the selection and tuning of algorithms, heuristics, and problem transformations are done in advance of the solving and are not responsive to the on-line performance of the system. The same is true for approaches such as that in Flener, Hnich, and Kızıltan [12], in which a model is built off-line defining the relationship between the problem instance and the best set of heuristics to use. There are several similar approaches to on-line algorithm or heuristic selection (see Allen and Minton [3] and Lobjois and Lemaître[23]). Although these approaches probe the problem instance on-line to determine the best algorithm or heuristics to use, and thus take performance feedback into account during this stage of solving, once the selection is made, no further feedback is used. A similar approach can be applied to algorithm parameter selection: in their *Auto-WalkSAT* algorithm, Patterson and Kautz [27] use probing to identify the best noise parameter for a particular algorithm/solver pair. Finally, instance-based solver or parameter selection need not depend on probing. Nudelman and his co-authors [22, 26] use a statistical regression approach to learn which problem features can be used to predict the run time of different solvers. They then use this prediction to select the fastest solver in a portfolio for each instance.

There are a number of approaches that make more use of feedback-type information for algorithm or parameter selection or for search control. Borrett, Tsang, and Walsh [5] use on-line performance feedback to switch between algorithms. Ruan, Horvitz, Kautz, and their coauthors [20, 29, 30] use it as part of a dynamic restart policy. Hoos [19] uses stagnation monitoring to dynamically adjust the noise parameter in WalkSAT algorithms. In the evolutionary algorithms community, a variety of techniques have been used to adapt genetic operators and parameters based on various performance measures (Eiben, Hinterding, and Michalewicz [11]). Similar methods have been used with simulated annealing (Wah and Wang [35]). There are also a variety of approaches that dynamically build up estimates of value or cost functions to guide the search (see, for example, Baluja, *et al.* [4], Boyan and Moore [6], Narayek [25], Ruml [31], and Lagoudakis and Littman [21]. These functions are measurements of the

"goodness" of particular states or action choices, and are developed on-line using accumulated performance data.

Adaptive techniques have also been used to modify problem representations. An "open-loop" off-line design approach for problem reformulation has been proposed by Hnich and Flener [17]. Feedback approaches have been used as well. For example, Pemberton and Zhang [28] have used (open-loop) phase transition information and on-line branching estimation to identify complex search problems and transform them into easier searches producing suboptimal solutions. Modification of penalty weights or chromosome representations in response to performance has also been explored in the evolutionary algorithms community (Eiben, Hinterding, and Michalewicz [11]).

In real time systems, though, time deadlines are a fact of life. None of the approaches described above explicitly takes this time bound into account when selecting solvers, heuristics, or parameter values. Some of these techniques represent anytime algorithms that can be stopped when a time bound is reached, but the time bound is not considered earlier. The solver thus does not take advantage of the known stopping time in order to make appropriate performance/speed trade-offs. Techniques also exist to monitor anytime algorithms and stop them when the solution improvement no longer justifies the additional time expenditure (Hansen and Zilberstein [16]), but again, this approach does not take a time bound explicitly into account in selecting or tuning the solver. Very recently, Carchrae and Beck [7] demonstrated a feedback-based system that switches among algorithms to get the best solution at a deadline. Their approach has a number of similarities to ours. Our goal in the example described here, however, is to use feedback to fine-tune algorithm or problem parameters rather than to select the algorithms themselves, though both applications fit within our general framework.

3 System Design

3.1 Solver Control

The generic framework for the feedback control of solving is shown in Figure 1 (see Crawford, *et al.* [10]). The control module is built upon a model or set of rules reflecting the relationship between problem solving dynamics (y), the real-time application requirement (i.e. the deadline T) and the choices for solvers, solver configurations, and problem transformations. These choices define the control parameters (u) of the solver. The model or rule base enables the prediction of the solver behavior defined by these control parameters. Based on the predicted behavior, the control module updates the control parameters (u), in order to achieve the best suboptimal solution at the specified time bound.

3.2 Control Parameters

Control parameters u that could be used to change the solving performance include the choice of solvers, solver configurations or problem representations. For

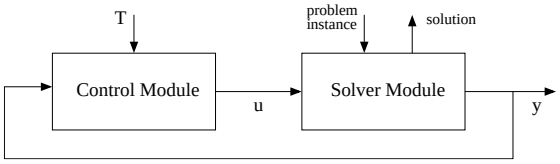


Fig. 1. Feedback control of solving framework.

example, some solvers may work better on under-constrained problems, while others are better choices for over-constrained problems (Shang and Fromherz [32]). In this case, the control parameter could be the choice of solvers. Another example is a solver with a restart policy, for which it is useful to adopt different restart cutoffs for problem instances with varied hardness levels. Previous research by Horvitz, Ruan, Kautz, and their co-authors [20, 29, 30] has proposed dynamic restart policies where the choice of cutoff is instance-based. In those mechanisms, the control parameter would be the restart cutoff configuration. Another possible control parameter is one defining a problem representation. For example, Pemberton and Zhang's ϵ-transformation [28] makes use of a parameter ϵ to define an off-line transformation of a tree search problem to one of lower complexity (with a loss of optimality, of course). The value of ϵ defines the severity of the reformulation.

In this paper, we investigate problem representation as a key control parameter for solving constrained optimization problems under a time bound. A linear constrained optimization problem is typically of the form:

$$\text{maximize} \quad c_1x_1 + c_2x_2 + \cdots + c_nx_n$$

respecting the constraints:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n + b_1 &\leq 0 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n + b_2 &\leq 0 \\ &\cdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n + b_m &\leq 0 \end{aligned} \tag{1}$$

where x_i, $i \in 1, 2, \cdots, n$ are variables whose value lies in some permissible set $\{X\}$.

To simplify problem solving, sometimes it is desirable to reduce the problem scale by problem transformation. The new problem representation should satisfy the following two requirements. First, it is deduced from the original problem but has smaller number of variables or constraints. Second, a solution to the new problem can be transformed into a solution to the original problem.

Many approaches exist for such a problem transformation. Three simple possibilities are variable grouping, constraint grouping and variable removal. In variable grouping, two or more variables are grouped together and considered as one new variable. This type of problem transformation leads to the same assignments for the variables in one group. For example, assume that we group

variables x_1 and x_2 in Equation 1 together and consider the group as a new variable g_1. Then the original problem becomes

$$\text{maximize} \quad (c_1 + c_2)g_1 + c_3x_3 + \cdots + c_nx_n$$

such that

$$\begin{aligned}(a_{11} + a_{12})g_1 + a_{13}x_3 + \cdots + a_{1n}x_n + b_1 &\leq 0\\ (a_{21} + a_{22})g_1 + a_{23}x_3 + \cdots + a_{2n}x_n + b_2 &\leq 0\\ &\cdots\\ (a_{m1} + a_{m2})g_1 + a_{m3}x_3 + \cdots + a_{mn}x_n + b_m &\leq 0\end{aligned}$$

Constraint grouping combines two or more constraints together. That is, a new constraint will be used to replace the old constraints in such a way that the satisfaction of the new constraint leads to the satisfaction of all the old constraints. For instance, the grouping of the first two constraints in Equation 1 may result in the following new constraint (assume the permissible values of the variables are positive),

$$max(a_{11}, a_{21})x_1 + \cdots + max(a_{1n}, a_{2n})x_n + max(b_1, b_2) \leq 0$$

Variable removal fixes the assignments of some variables so that the number of unknown variables is smaller after the problem transformation.

Other, more complex approaches to problem transformation along these lines can also be envisioned; for example, hierarchical approaches to variable grouping in which the problem is decomposed into a number of smaller subproblems. In all the grouping approaches, whether of variables or constraints, care must be taken when choosing how to define the groups, as this choice can affect solver performance significantly.

3.3 Control Module

In the feedback control framework, the control module leverages a model or set of rules to predict the solving behavior with different control parameter choices. Therefore, a good model that accurately reflects the solving dynamics is the key to good control parameter selection. In this paper, we propose a general modeling framework. Given a solver with a defined set of parameter choices $U = u_1, u_2, \ldots, u_N$, we will first apply it to a group of representative benchmark problem instances $P_1, P_2, \ldots, P_M$. The solving performance on those instances will then be used to generate the model (see Figure 2). This approach is similar to that taken by Nudelman *et al.* [26], but with the differences that we are aiming to predict solution quality at a time bound rather than solver run time (to the optimal solution), and that we use data acquired during the solving process in the predictions.

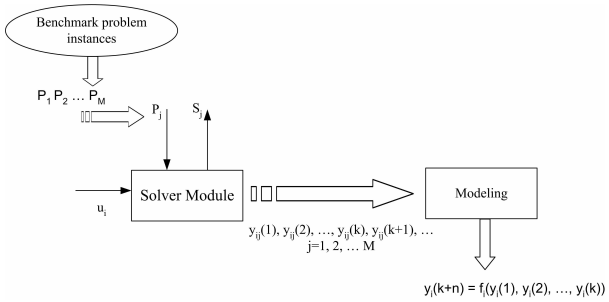


Fig. 2. Modeling framework.

As shown in Figure 2, a model (represented by f_i) is generated for each solver configuration (defined by control parameter u_i)[1]. The control module will then use the models ($f_1, f_2, \cdots, f_N$, assuming there are N different solver configuration choices) to predict the solving behavior on any given problem instance and try to choose the best parameters for solving the problem instance. As the control module will use the dynamic solving information $y(k)$, such as suboptimal solutions obtained, to make the best parameter choice, parameter changes may be required on-line. For example, the control module could choose some $u(1) \in U$ at the first sampling interval and later switch to other configurations $u(2), u(3), \cdots, u(k) \in U$ if switching is predicted to produce better solving performance.

When changing parameters from one configuration to another, one of two types of switch strategies can be applied. These are constructive and restart strategies. A constructive strategy uses the dynamic information obtained with one configuration to adapt or modify the starting point of the solving with the new configuration, while with a restart strategy, when the configuration switches from u_1 to u_2, say, the solving process for u_1 will pause and the process for u_2 will start from scratch or restart from where it was paused. An intelligent constructive strategy has the potential to improve the performance of the new configuration, but may be difficult to build, depending on the type of parameters u being used, and may require a solver-specific approach. With the simple restart strategy, on the other hand, the solving process of each configuration is independent, which leads to a simpler control system whose dynamics will be much easier to model. Hence, in our general feedback control framework, we have chosen the restart switching strategy.

For prediction and control in this framework to be feasible, two assumptions must be made. First, we assume that the solver performance on the modeling problem instances can be used to predict the performance on new instances.

[1] This paper only considers the case where there is a limited discrete number of control parameter choices. The modeling framework for control with a continuous choice of parameters requires a somewhat different modeling approach or a method for interpolating among models and is a topic for future work.

Second, if the solver behavior on two problem instances are the same from the beginning of the solving, then the solver is assumed to be very likely to continue performing similarly on the two for the rest of the solving process.

Based on these assumptions, a brute force prediction and control algorithm can be designed from a simple model. Let S be the set of M modeling problem instances. The modeling process solves them until the specified time bound (T) with each solver (defined by the parameter configuration $u_i \in U$). The suboptimal solutions reached by each solver at every sample interval are recorded. We use $y_{ij}(k)$ to represent the suboptimal solution reached by the i^{th} solver at the k^{th} sampling interval for modeling problem P_j. Note that the same parameter configuration is used throughout a solving run (there is no switching).

When solving a new problem instance p, we will use a distance metric to evaluate how similar the performance on p is to that on each modeling instance P_j. The distance to P_j at time interval k, $D_j(p,k)$, is calculated based on the performance on p by the solvers (defined by $u_1, u_2 \cdots, u_N$) and their known performance on P_j. Assume that at time interval k, the algorithm has spent k_i sampling intervals on the solving process with configuration u_i, where $\sum_{i=1}^{N} k_i = k$, and generated outputs $y_i(1), y_i(2), \ldots, y_i(k_i)$. Then we calculate $D_{ij}(p,k_i)$, the distance between p and P_j with solver u_i at time interval k as follows:

$$D_{ij}(p,k_i) = \sum_{n=1}^{k_i} |y_{ij}(n) - y_i(n)| \tag{2}$$

All experience is weighted equally. From 2 the distance metric $D_j(p,k)$ is derived:

$$D_j(p,k) = \sum_{i=1}^{N} D_{ij}(p,k_i) \tag{3}$$

At each sampling interval, the algorithm calculates $D_j(p,k)$ for every $P_j \in S$ and finds the modeling problem that is the smallest distance from p. That is, it finds the problem P_{opt} that yields the minimum distance $D_{opt}(p,k) = min(\{D_j(p,k)|P_j \in S\})$. Then, the algorithm uses the solvers' performance on instance P_{opt} to predict how they would perform on p if allowed to run for the rest of the sampling intervals. Then, for the next sampling interval, the algorithm chooses the solver configuration $u(k+1)$ whose predicted final result at deadline T is best. The above process is repeated at each interval. At the beginning of a solving run, a minimum number of sampling intervals of each configuration will be run to ensure accurate prediction. For the sake of reducing switching in difficult cases, the feedback process continues until a specified time, at which point one solver configuration will be chosen for the rest of the solving.

4 Case Study

As a case study, we apply the proposed feedback control paradigm (Section 3) to the real-time performance control of a multidimensional knapsack problem

solver. The real-time solving performance of this type of problem is important because many practical real-time problems such as resource allocation in distributed systems, capital budgeting, and cargo loading can be formulated as multidimensional knapsack problems.

4.1 Multidimensional Knapsack Problem Solver

The 0-1 multidimensional knapsack problem (MKP01) can be stated as:

$$MKP01 = \begin{cases} \text{maximize } c \cdot x \text{ subject to} \\ Ax \leq b \text{ and } x \in \{0,1\}^n \end{cases}$$

where $c \in \mathbb{N}^n$, $A \in \mathbb{N}^{m \times n}$ and $b \in \mathbb{N}^m$.

There are a number of approaches to solving the MKP01 problem in the literature. A standard was set by Chu and Beasley [9], who obtained good results using a genetic-algorithm-based heuristic method. Their problem set was made publicly available in the OR-Library [2], and has been a benchmark problem set for testing other algorithms. Their paper also provides a useful survey of MKP01 solvers at that time. Many other heuristic solvers have been proposed since then (for example, Holte [18], Fortin and Tsevendorj [13], and Vasquez and Hao [33]), some placing emphasis on solution quality and some focusing on solving speed.

We will here apply the feedback control paradigm to a MKP01 solver developed by Vasquez, Hao, and Vimont [33, 34]. This solver takes a hybrid approach that combines linear programming with an efficient tabu search algorithm. It gave results on the OR-Library benchmarks [2] that the authors claim were the best known at the time. Other solvers could of course be used as targets for real-time performance control; we chose this one based on its final solution quality.

The main idea of the hybrid solver is to perform a search around a solution of the fractional relaxed MKP01 problem with additional constraints. Starting from the obvious statement that each solution of MKP01 satisfies the property: $1 \cdot x = \sum_1^n x_j = k$, where k is a positive integer, they add this constraint to the fractional relaxed MKP01 to obtain a series of problems like:

$$MKP[k] = \begin{cases} \text{maximize } c \cdot x \text{ s.t.} \\ Ax \leq b \text{ and } x \in [0,1]^n \text{ and} \\ 1 \cdot x = k \in \mathbb{N} \end{cases}$$

These $MKP[k]$ are solved and their fractional solutions, $\bar{x}_{[k]}$, are used as starting points for tabu searches. They are solved in order of most promising to least promising k. In Vasquez and Vimont [34], k_0, the first value used, is the rounded sum of the elements of the optimal solution $\bar{x}$ of the relaxed MKP01. That is, $k_0 = [1 \cdot \bar{x}]$, where

$\bar{x} = \arg\max c \cdot x$ s.t.

$Ax \leq b, x \in [0,1]^n$

The region around $\bar{x}_{[k_0]}$ is the first to be tabu searched. Next, regions around $\bar{x}_{[k]}, k = k_0 - 1, k_0 + 1, k_0 - 2, k_0 + 2, \cdots$ are sequentially tabu searched. According to Vasquez and Vimont [34], this search order ensures the exploration of the hyperplanes $1.x = k$ in the decreasing order of $\bar{z}_{[k]} = c.\bar{x}_{[k]}$, the optimal values of $MKP[k]$.

To reduce the search space, the algorithms impose a geometric constraint on the tabu search neighborhood, so that the local search is limited to a sphere of fixed radius around the fractional optimum $\bar{x}_{[k]}$. Therefore, each binary configuration x reached by the local tabu search satisfies the following two constraints:

1) $1 \cdot x = k$

2) $|x, \bar{x}_{[k]}| = \sum_{j=1}^{n} |x_j - \bar{x}_{[k]j}| \leq \delta_{max}$

4.2 Control Approach

To control the performance of the aforementioned hybrid solver in the context of a fixed time bound, several control parameters could be effective. First, the choice of the geometric constraint δ_{max} on the tabu search neighborhood will have an important impact on its performance. We believe that the best choice of δ_{max} is problem instance dependent and application requirement dependent. Therefore, on-line feedback tuning may be required to reach the optimal configuration. Allocating the time that the algorithm will run on each hyperplane is another place where the feedback control could be beneficial. Instead of strictly following the search order k_0, $k_0 - 1$, $k_0 + 1$, $k_0 - 2$, ..., it would be desirable to have some dynamic mechanism to identify unpromising hyperplanes and switch to possible better hyperplanes, so that a better suboptimal solution can be reached within the time bound.

In this paper, we consider the case where the time bounds imposed on solving are quite tight, and we have therefore chosen problem representation as the control parameter. To reduce the scale of the problem, we changed the problem representation by grouping several variables together. The solver performance on the new problem representation will be quite different with different grouping strategies. Although any grouping strategy might speed the solving process, some will cause big performance degradations in terms of the suboptimal solutions for the transformed problem. Instead of grouping variables together randomly, our algorithm groups pairs of variables with small values of the relative price heuristic:

$$\frac{c_j}{\sum_{i=1}^{m} \frac{a_{ij}}{b_i}}$$

Grouping with this heuristic allows for trading off between solution quality and solving speed.

To demonstrate that the optimal solver configuration is time-dependent, we describe experiments with the mk_gk11.dat benchmark (proposed by Glover and Kochenberger [1]). This benchmark problem instance includes 2500 variables and 100 constraints. Through variable grouping, we generated four new problem representations, in which the number of variables are 2000, 1500, 1000 and 500 respectively. For each representation, we used a reimplementation of the hybrid solver to solve the problem. Solution qualities (as fractions of the relaxed MKP01 solution) over time for one particular problem instance are shown in Figure 3. One can see that different time bounds lead to different optimal problem representation choices. For instance, if the solver is only allowed 50 seconds to solve the problem, the optimal problem representation should be the one with

1000 variables. For a 100-second time bound, the optimal choice becomes the representation with 2000 variables.

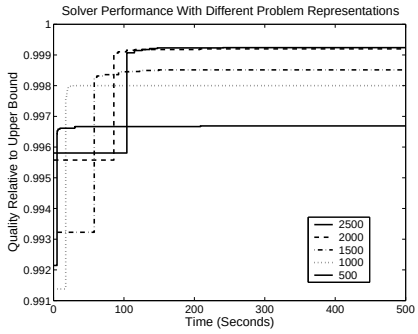


Fig. 3. The hybrid solver performance with different problem representations.

Previous work (Vasquez and Hao [33]) has demonstrated that the required solving time for a problem instance depends on its size, the number of variables and constraints in the instance. We further investigated whether other static features have an impact on the solver performance and can thereby provide a hint on selecting the control parameter (the problem representation, in this case). If offline analysis can be helpful in optimizing the solving process, the dynamic reconfiguration and switching overhead will be reduced. Several features of the knapsack problem were analyzed for their ability to predict how well the solver will perform (at the deadline) with each of the two problem representations. Example features included the tightness ratio $\frac{b_i}{\sum_{j=1}^{n} a_{ij}}$ where $i = 1, 2, ..., m$, and the relative price $\frac{c_j}{\sum_{i=1}^{m} \frac{a_{ij}}{b_i}}$. Linear regression results indicate that only the tightness ratio of the problem instance has an obvious effect on the solver performance, while other static features do not seem to yield any consistent indication of how the solver will perform.

This static feature analysis further demonstrates that static information alone is not enough for guiding the parameter choice. Experiments also show that for problem instances with the same tightness ratio the solver performance can still be quite different. Therefore, in this example our focus is on problem instances with the same tightness ratio and size (the number of variables and constraints in the instance). For such a group of problem instances, we build a dynamic model that can be used as a basis for making problem representation choices solely using solver runtime information.

We propose to use the solver runtime information (y) and the real-time application requirement (deadline T) to determine the right problem representation choice, using the feedback control paradigm. The best suboptimal solutions $z^*(k)$ reached at each sampling interval (i.e. the system performance outputs $y(k)$) are used for predicting the system performance and choosing the problem repre-

sentation [2]. We consider two different problem representations as our control parameter choices, $U = \{u_1, u_2\}$. One representation requires no problem transformation and presents the hybrid solver with the original problem, while the other representation transforms the problem through variable grouping.

4.3 Experimental Results

We implemented a benchmark generator according to the algorithm described in Chu and Beasley [9]. We then used it to generate problem instances that have $n = 500$ objects, $m = 30$ constraints and 0.25 tightness ratio. For these problem instances, we investigate two different problem representations. One representation will change the number of objects to $n = 420$ (chosen for demonstration purposes) by grouping 80 pairs of small relative price objects together. The other representation will retain the original problem formulation with $n = 500$ objects. The objective is to achieve the best suboptimal solution at a specified time bound with the Vasquez and Hao hybrid solver by solving the given problem instance with the feedback-chosen problem representation.

1100 problem instances were generated, with 1000 of them used as modeling instances to guide the feedback solving of the final 100 validation problem instances. The modeling and feedback control were done as described in Section 3. The sampling interval (feedback update period) was set to 10 iterations. The minimum number of intervals for each problem formulation was set to 40 (400 iterations) and the number of intervals after which the feedback was turned off and a single formulation was selected was 400 (4000 iterations). Two experiments were carried out, where different modeling and validation problem instances were chosen from the 1100 problem instance set. For reasons of simplicity, we chose 10000 solving iterations as the time bound, where a solving iteration is defined as the time interval between two successive moves of the local tabu search for the hybrid solver. Sample control traces, showing the controller's choice of problem representation at each iteration, are given in Figure 4.

To assess the the performance of the solver with and without feedback, we performed a paired comparison between the feedback-controlled solver and the no-feedback solver using each of the two problem formulations (420 and 500 variables). For this comparison, the two experiments (with different modeling and validation sets) were combined. Figure 5 shows the distribution of the differences in quality at the deadline between the feedback and no-feedback cases. Quality is measured as a fraction of the upper bound provided by the solution to the relaxed problem. The boxes indicate the central 50% of the data, while the whiskers denote the extent of the data. The gray bars show the 95% confidence interval around the mean.

The figure shows that with feedback control of the choice of problem representation, the solver performed better on average at the deadline than with

[2] Other possible dynamic information that could be used for feedback control includes the constraint violation $v_b = \sum_{i|a_i.x>b_i}(a_i.x - bi)$ and the time since the last improvement in the solution (stagnation).

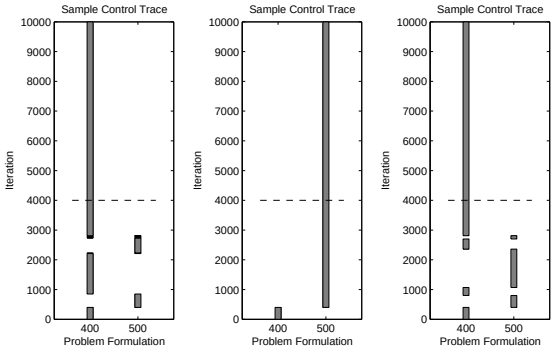


Fig. 4. Typical sample traces showing the feedback control switching between the two problem representations. The horizontal dashed line indicates the point beyond which no further switching was permitted.

either of the fixed problem representations. The improvement was small, but statistically highly significant ($p < 0.01$ in a binomial test for equal or better performance).

5 Conclusions and Future Work

In conclusion, we have here demonstrated an approach for using feedback control to improve the quality of the solution obtained when solving a problem under a strict time bound. The algorithm makes use of a model based on solution profiles to dynamically predict solver performance and choose solver control parameters accordingly. We have presented a case study application of this approach to solving the 0-1 multidimensional knapsack problem with the hybrid solver described by Vasquez, Hao, and Vimont [33, 34]. In this application, small solution quality improvements at the time bound were seen.

There are several issues still to be addressed with the feedback method described here. The modeling and prediction technique used can incur a large overhead if the size M of the modeling set is big. Thus, in the future it will be essential to carry out research on how to reduce the size of the modeling problem set by, for example, removing redundant similar-performance problems. Another future research topic is how to generate the modeling problem set so that it includes "representative" problems. The modeling set should be able to self-evolve during the solving process when new performance unique problem instances are identified. Further, analyzing the underlying reasons why two problems have similar performance profiles is a very interesting research topic. It would also be useful to explore how the models used here could transfer to broader problem classes. Other modeling and prediction methods are of course possible, as well, and bear further investigation.

The choice of problem granularity as control variable should also be explored further. Though a performance improvement was observed with the knapsack

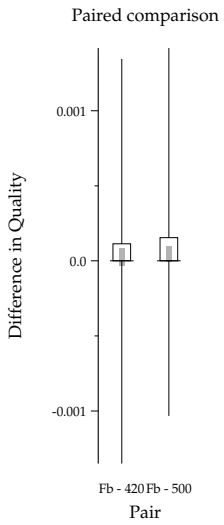


Fig. 5. Performance of the solver under feedback control, as compared to using each of the problem representations without control. The data plotted is the difference between the solution quality for the feedback case and the no-feedback case. The boxes indicate the region where 50% of the data points lie, and the whiskers show the extent of the data. The gray bars show the 95% confidence interval around the mean.

problem, it was not a very large one. It is possible that other classes of problems might lend themselves better to the approach of controlling the granularity by aggregating variables. Additionally, though we explored different ways to do the variable grouping, there may be better heuristics to use than relative price, and some of the other approaches to reducing granularity might be more fruitful than variable grouping.

Here we explored the use of on-line feedback control to choose solver and problem parameters. It would be interesting to integrate this technique with an off-line modeling and prediction approach based on, for example, problem instance features, such as that used by Nudelman and his co-authors [26, 22]. Such a combined approach might further improve performance at the deadline.

Feedback control of solving in response to a time deadline is a complex problem with many interacting variables, including choice of solver, choice of control parameters, choice of modeling and prediction techniques, and choice of control logic. If these obstacles can be overcome, however, the benefits of being able to provide high-quality solutions in real-time settings would be many.

6 Acknowledgments

The authors would like to thank Yi Shang, Hai Fang, Tarek Abdelzaher, John Stankovic, and Gang Tao for suggestions and helpful discussions. This work was partially supported by DARPA under contract F33615-01-C-1904.

References

1. Large benchmark, http://hces.bus.olemiss.edu/tools.html.
2. OR-library, http://mscmga.ms.ic.ac.uk/jeb/orlib/mknapinfo.html.
3. J. A. Allen and S. Minton. Selecting the right heuristic algoithm: runtime performance predictors. In *Advances in Artificial Intelligence. 11th Biennial Conference of the Canadian Society for Computational Studies of Intelligence*, pages 41–53, Toronto, Ontario, May 1996.
4. S. Baluja, A.G. Barto, K.D. Boese, J. Boyan, W. Buntine, T. Carson, R. Caruana, D.J. Cook, S. Davies, T. Dean, T.G. Dietterich, P.J. Gmytrasiewicz, S. Hazlehurst, R. Impagliazzo, A.K. Jagota, K.E. Kim, A. McGovern, R. Moll, A.W. Moore, E. Moss, M. Mullin, A.R. Newton, B.S. Peters, T.J. Perkins, L. Sanchis, L. Su, C. Tseng, K. Tumer, X. Wang, and D.H. Wolpert. Statistical machine learning for large-scale optimization. *Neural Computing Surveys*, 3:1–58, 2000.
5. J. E. Borrett, E. P.K. Tsang, and N. R. Walsh. Adaptive constraint satisfaction: the quickest first principle. Technical Report CSM-256, University of Essex Department of Computer Science, 1995.
6. J. A. Boyan and A. W. Moore. Learning evaluation functions to improve optimization by local search. *Journal of Machine Learning Research*, 1:77–112, 2000.
7. Tom Carchrae and J. Christopher Beck. Low-knowledge algorithm control. In *Proceedings of AAAI-04*, pages 49–54. AAAI Press / The MIT Press, 2004.
8. Y. Caseau, F. Laburthe, and G. Silverstein. A meta-heuristic factory for vehicle routing problems. *Constraint Programming*, 1999.
9. P.C. Chu and J.E. Beasley. A genetic algorithm for the multidimensional knapsack problem. *Journal of Heuristics*, 4:63–86, 1998.
10. Lara S. Crawford, Markus P.J. Fromherz, Christophe Guettier, and Yi Shang. A framework for on-line adaptive control of problem solving. In *CP'01 Workshop on On-line Combinatorial Problem Solving and Constraint Programming*, December 2001.
11. A. E. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in evolutionary algorithms. *IEEE transactions on evolutionary computation*, 3:124–141, 1999.
12. P. Flener, B. Hnich, and Z. Kızıltan. A meta-heuristic for subset problems. In *Practical Aspects of Declarative Languages. Third International Symposium, PADL 2001*, pages 274–287, Las Vegas, NV, March 2001.
13. Dominique Fortin and Ider Tsevendorj. Global optimization and multi knapsack: a percolation algorithm. Technical Report 3912, Institut National de Recherche en Informatique et en Automatique (INRIA), 2000.
14. J. Gratch and G. DeJong. COMPOSER: a probabilistic solution to the utility problem in speed-up learning. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 235–240, San Jose, CA, July 1992.
15. J. Gratch and G. DeJong. A decision-theoretic approach to adaptive problem solving. *Artificial Intelligence*, 88(1-2):101–142, 1996.
16. E. A. Hansen and S. Zilberstein. Monitoring the progress of anytime problem-solving. In *13th National Conference on Artificial Intelligence*, Portland, OR, August 1996.
17. B. Hnich and P. Flener. High-level reformulation of constraint programs. In *Proceedings of the Tenth International French Speaking Conference on Logic and Constraint Programming*, pages 75–89, 2001.
18. Robert C. Holte. Combinatorial auctions, knapsack problems, and hill-climbing search. In *Proceedings of AI'2001, the Fourteenth Canadian Conference on Artificial Intelligence*. Springer, 2001.

19. Holger H. Hoos. An adaptive noise mechanism for WalkSAT. In *Proceedings of AAAI-02*, pages 655–660. AAAI Press / The MIT Press, 2002.
20. E. Horvitz, Y. Ruan, C. Gomes, H. Kautz, B. Selman, and M. Chickering. A Bayesian approach to tackling hard computational problems. In *Proceedings of the Seventeenth Conference on Uncertainty and Artificial Intelligence*, Seattle, WA, August 2001.
21. M. G. Lagoudakis and M. L. Littman. Learning to select branching rules in the DPLL procedure for satisfiability. In *LICS 2001 workshop on theory and appliations of satisfiability testing (SAT 2001)*, 2001.
22. Kevin Leyton-Brown, Eugene Nudelman, and Yoav Shoham. Learning the empirical hardness of optimization problems: the case of combinatorial auctions. In *Constraint Programming 2002*, 2002.
23. L. Lobjois and M. Lemaître. Branch and bound algorithm selection by performance prediction. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 353–358, Madison, WI, July 1998.
24. S. Minton. Automatically configuring constraint satisfaction programs: a case study. *Constraints*, 1(1-2):7–43, 1996.
25. A. Narayek. An empirical analysis of weight-adaptation strategies for neighborhoods of heuristics. In *Proceedings of the fourth metaheuristics international conference*, pages 211–216, 2001.
26. Eugene Nudelman, Alex Devkar, Yoav Shoham, and Kevin Leyton-Bown. Understanding random SAT: Beyond the clauses-to-variables ratio. In *Proceedings of Constraint Programming 2004*, 2004. to appear.
27. D. J. Patterson and H. Kautz. *Auto-walksat*: a self-tuning implementation of *walksat*. *Electronic Notes in Discrete Mathematics*, 9, 2001.
28. J. C. Pemberton and W. Zhang. ϵ-transformation: exploiting phase transitions to solve combinatorial optimization problems. *Artificial Intelligence*, 81(1-2):297–325, 1996.
29. Y. Ruan, E. Horvitz, and H. Kautz. Restart policies with dependence among runs: a dynamic programming approach. In *Proceedings of the Eighth International Conference on Principles and Practice of Constraint Programming (CP-2002)*. Springer-Verlag, 2002.
30. Y. Ruan, E. Horvitz, and H. Kautz. Hardness-aware restart policies. In *IJCAI-03 Workshop on Stochastic Search Algorithms*, Alcapulco, Mexico, 2003.
31. W. Ruml. Incomplete tree search using adaptive probing. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pages 235–241, Seattle, WA, August 2001.
32. Yi Shang and Markus P.J. Fromherz. Experimental complexity analysis of continuous constraint satisfaction problems. *Information Sciences*, 153:1–36, 2003.
33. Michel Vasquez and Jin-Kao Hao. A hybrid approach for the 0-1 multidimensional knapsack problem. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pages 328–333, Seattle, WA, August 2001.
34. Michel Vasquez and Yannick Vimont. Improved results on the 0-1 multi dimensional knapsack problem. In *Sixteenth Triennial Conference of the International Federation of Operational Research Societies (IFORS 2002)*, 2002.
35. B. W. Wah and T. Wang. Tuning strategies in constrained simulated annealing for nonlinear global optimization. *International Journal of Artificial Intelligence Tools*, 9(1), 2000.

Belief and Desire Networks for Answering Complex Queries

Cédric Pralet[1], Gérard Verfaillie[1], and Thomas Schiex[2]

[1] LAAS-CNRS, Toulouse, France, {cpralet,gverfail}@laas.fr
[2] INRA, Castanet Tolosan, France, tschiex@toulouse.inra.fr

Abstract. Constraint networks have been designed to reason about constraints between variables, whereas probabilistic networks have been separately designed to reason about conditional probabilities or local joint probabilities between random variables. In this working paper, we propose a framework which allows constraints and probabilities, and more generally desires and beliefs, to be represented and processed together. This framework encompasses numerous existing frameworks such as classical and valued constraint satisfaction networks, bayesian networks, influence diagrams, stochastic boolean satisfiability and stochastic constraint satisfaction problems. It allows numerous reasoning and decision problems in which desires and beliefs are imbricated to be represented. It also allows various queries, involving diversely quantified variables, to be formulated. Because this framework is based on simple algebraic properties of combination and elimination operators, we think that it will be then possible to develop on top of it powerful generic inference and search algorithms able to handle these queries.

1 Existing constraint and probabilistic frameworks

In the last decades, several representation frameworks have been designed in the AI community. *Constraint satisfaction problems* (CSP [1,2]), also referred to as *Constraint networks* (CN), have been introduced to represent problems in which local constraints between discrete variables express local requirements or facts. These constraints implicitly define a global constraint over all the variables, that is a partition of the set of the complete variable assignments between solutions and non solutions. The CSP framework has been extended to represent, in addition to the usual hard constraints, soft constraints which express soft requirements or uncertain facts. These constraints implicitly define a global soft constraint over all the variables, that is a satisfaction or violation degree distribution over the set of the complete variable assignments. See for example the *Valued* and *Semiring-based* CSP frameworks (VCSP and SCSP [3]), which encompass classical, fuzzy, additive, lexicographic and probabilistic CSP.

Concurrently, *Bayesian networks* (BN [4–6]), have been introduced to represent problems in which local conditional probability distributions between discrete random variables implicitly define a joint probability distribution over all

the variables. *Influence diagrams* [7] extend bayesian networks by adding random utility variables and non random decision variables to the usual random variables. In another direction, *Dynamic bayesian networks* [8] allow dynamic stochastic processes to be modelled.

Markov decision processes (MDP [9–11]) have been devised to represent sequential decision problems under uncertainty. Unlike dynamic bayesian networks or influence diagrams, they do not reason on variables, but directly on states and decisions, that is on aggregate large domain variables. *Factored MDP* [12] extend MDP concepts and methods to a variable-based representation of states and decisions. In another direction, *Partially observable Markov decision processes* (POMDP [13]) take into account the fact that the state of the dynamic process is generally only indirectly observable via potentially erroneous observations.

Mixed CSP [14] introduce a distinction between controllable and uncontrollable variables in the CSP framework, where all the variables are usually assumed to be controllable *i.e.*, one can decide on the value they take. *Stochastic constraint satisfaction* (SCSP [15, 16]) and *Stochastic satisfiability* (SSAT [17]) extend the CSP and SAT frameworks by introducing randomly and universally quantified variables in addition to the usually existentially quantified variables.

Hybrid or *mixed networks* [18, 19] allow the deterministic part of bayesian networks to be represented and manipulated more efficiently as constraints.

2 Modelling requirements

Taking into account all these frameworks, their scopes, and the requirements of various application domains, it is possible to list what should be expected from a framework which would combine *constraints* and *probabilities*, and more generally *desires* and *beliefs*.

First, we need *variables* to represent states as well as decisions. Each variable is equipped with a *domain* of possible values, which may be discrete or continuous, finite or infinite. Although some problems require continuous or infinite domains, we restrict ourselves to *discrete* and *finite* domains, mainly for the needs of the future algorithms. Values in these domains may be *symbolic* or *numeric*.

Then, we need to represent *local relations* between variables. The word *local* means that each relation links a subset of the existing variables (typically a small number of them). We assume the existence of two kinds of relation. Relations of the first type are denoted as *belief* relations. They are intended to represent beliefs or knowledge about combinations of values. More precisely, each belief relation associates with each combination of values of the variables it links a *belief* degree, which represents to which extent this combination is possible or probable. Relations of the second type are denoted as *desire* relations. They are intended to represent desires on combinations of values. More precisely, each desire relation associates with each combination of values of the variables it links a *desire* degree, which represents to which extent this combination is desirable, useful, or required. Special values are associated, on the one hand, with com-

pletely unbelieved (impossible) or undesired (inadmissible) combinations and, on the other hand, with completely believed (necessary or certain) combinations. These notions of *belief* and *desire* could be compared with the ones used in the *Belief-Desire-Intention* (BDI [20]) framework.

Finally, we need *combination* (or aggregation) operators to combine degrees of belief from different local belief relations, degrees of desire from different local desire relations, and degrees of belief and degrees of desire together, so as to associate a degree with any complete variable assignment. We need also *elimination* (or marginalisation) operators to extract synthetic information from any subset of variables, so as finally to answer any query about the network[3].

Typical queries — motivated by various tasks, such as situation assessment, situation explanation, situation prediction, property verification, decision assessment, or decision making, in settings that may be sequential or not, multi-agent or not — will involve arbitrary sequences of variable eliminations.

Finally, some sensible *properties* should be assumed about combination and elimination operators. These properties should ideally enable *global* computation to be performed by combining results of more *local* computations, as for example in *dynamic programming*-style algorithms [21].

3 An example

The following example will be used throughout this paper to illustrate the main features of the proposed framework.

Peter wants to create a start-up and requires help. He needs the participation and the investment of some of his friends, Paul, John, Luke, and Matthew. In order to get their agreement, he decides to organise a dinner. Things will be discussed at the end of the dinner. It is therefore very important to get most of his friends present at that time.

Peter knows that if Paul is present at the end of the dinner, he will agree to participate and invest 10 *k€ in the company. The same kind of guess stands for John, Luke, and Matthew with respectively* 100, 20, *and* 50 *k€. In order to launch his start-up, Peter needs the participation of Paul, John, or Luke (at least one of them). Participation of Matthew is not required.*

Peter knows that John will come. He knows that Paul will come with a probability of 0.7, *but that, if Paul comes, John, who cannot stand him, will leave the*

[3] Given a function f of the variables in V, the elimination from f of a variable $v \in V$ using an operator o results in a new function f_v of the variables in $V - \{v\}$. The value of f_v for a specific tuple t of values of the variables in $V - \{v\}$ is the result of the application of o on the values of f for all the possible extensions of t on v. Computation of the minimum, the maximum, or the mean value of desires, beliefs, or combinations of both over the cartesian product of domains of variables are typical examples of such operations and min, max, and + are the associated elimination operators. Variable quantifiers used in logical frameworks can be viewed as particular cases of elimination. In fact, if we assume that $f \prec t$, then max (resp. min) is the elimination operator associated with the $\exists$ (resp. $\forall$) quantifier in boolean domains.

dinner immediately. Peter knows that Luke and Matthew come with the same car, so that they will either come together, or none will come. He knows that they will come with a probability of 0.6. *Peter also knows that the presence of Paul and John and the presence of Luke and Matthew are independent.*

Considering the menu, Peter must choose between fish and meat for the main course, and between white and red for the wine, but he does not want fish with red wine. Peter knows that John does not like fish. If fish is chosen, he will leave the dinner. He also knows that Luke does not like meat with white wine and that Matthew does not like red wine. If the menu does not suit them, they will also leave the dinner.

4 A belief and desire-based formalism

4.1 Problem definition

We define a problem instance Pb as a tuple $\langle V, B, D, S_b, S_d \rangle$ where:

- V is sequence of n *variables*; V_i is the i^{th} variable; with each variable V_i, is associated a finite set Dom_i, which defines its *domain* of possible values;
- B is a sequence of b *belief relations*; B_j is the j^{th} belief relation; with each belief relation B_j, are associated:
  - a subsequence Scb_j of V, which defines its *scope*, that is the variables it links;
  - a function ϕ_j, which associates with any element of the cartesian product of the domains of the variables in Scb_j a *belief degree* in E_b (see below);
- similarly, D is a sequence of d *desire relations*; D_k is the k^{th} desire relation; with each desire relation D_k, are associated:
  - a subsequence Scd_k of V, which defines its *scope*;
  - a function ψ_k, which associates with any element of the cartesian product of the domains of the variables in Scd_k a *desire degree* in E_d (see below)[4];
- S_b is a quintuple $\langle E_b, \preceq_b, \bot_b, \top_b, \otimes_b \rangle$, which defines the *belief structure*; E_b is the set of possible belief degrees; $\preceq_b$ is a total order on E_b; smaller belief degrees are associated with less believed facts; $\bot_b$ is the minimum element of E_b; it is associated with completely unbelieved (impossible) facts; $\top_b$ is its maximum element; it is associated with completely believed (necessary or certain) facts; $\otimes_b$ is a binary closed operator on E_b; it allows the belief degree of any conjunction of facts to be computed from the belief degrees of its components (combination operation);
- S_d is a quintuple $\langle E_d, \preceq_d, \bot_d, u_d, \otimes_d \rangle$, which defines the *desire structure*; E_d is the set of possible desire degrees; $\preceq_d$ is a total order on E_d; smaller desire degrees are associated with less desirable facts; $\bot_d$ is the minimum element of E_d; it is associated with completely undesired (inadmissible) facts; u_d is a special element, associated with indifferent facts, at the frontier between undesirable and desirable facts; $\otimes_d$ is a binary closed operator on E_d; it

[4] Belief and desire functions can be either analytically, or enumeratively defined.

allows the desire degree of any conjunction of facts to be computed from the desire degrees of its components (combination operation); note that the structure of S_d is different from the one of S_b, since we assume the existence of completely undesired facts, but not the one of completely desired ones; this can be obviously assumed if needed.

4.2 Problem semantics

A problem $\langle V, B, D, S_b, S_d \rangle$ allows a belief degree $b(A)$ and a desire degree $d(A)$ to be computed from any complete assignment A of the variables. Denoting $A[V']$ the projection of A on a sub-sequence $V' \subseteq V$, these degrees are defined by:

$$b(A) = (\otimes_b^{j=b}{}_{j=1} \phi_j(A[Scb_j])) \otimes \top_b \quad (1)$$

$$d(A) = (\otimes_d^{k=d}{}_{k=1} \psi_k(A[Scd_k])) \otimes u_d \quad (2)$$

Informally, the belief (resp. desire) degree of A is the result of the combination, via the belief (resp. desire) combination operator, of the belief (resp. desire) relations applied to the projection of A on their respective scope. Said otherwise, we assume that the global belief (resp. desire) relation is factorised. If there is no belief (resp. desire) relation, every fact is considered to be certain (resp. indifferent).

4.3 Required algebraic properties

Several algebraic properties on the belief and desire structures S_b and S_d can be considered as sensible.

First, the combination operators $\otimes_b$ and $\otimes_d$ should be *commutative* and *associative*, because we do not want the result of these combination operations to be dependent on the way they are performed.

They should be *monotonic*, since if a fact F_1 is less believable (resp. desirable) than a second fact F_2, then the conjunction of F_1 with any third fact F_3 is less believable (resp. desirable) than the conjunction of F_2 with the same fact F_3:

$$\forall b_1, b_2, b_3 \in E_b,\ (b_1 \preceq_b b_2) \rightarrow (b_1 \otimes_b b_3 \preceq_b b_2 \otimes_b b_3) \quad (3)$$

$$\forall d_1, d_2, d_3 \in E_d,\ (d_1 \preceq_d d_2) \rightarrow (d_1 \otimes_d d_3 \preceq_d d_2 \otimes_d d_3) \quad (4)$$

Then, the minimum elements $\bot_b$ and $\bot_d$ in E_b and E_d should be *annihilator*, because the conjunction of any fact with an impossible (resp. inadmissible) fact is impossible (resp. inadmissible) too.

$$\forall b \in E_b,\ b \otimes_b \bot_b = \bot_b \quad (5)$$

$$\forall d \in E_d,\ d \otimes_d \bot_d = \bot_d \quad (6)$$

Finally, the maximum element $\top_b$ in E_b should be a *neutral element* for $\otimes_b$: the combination of any fact with a certain fact yields an unchanged belief degree.

Similarly, the element u_d in E_d should be a *neutral element* for $\otimes_d$ because the conjunction of any fact with an indifferent fact yields an unchanged desire degree.

$$\forall b \in E_b,\ b \otimes_b \top_b = b \tag{7}$$
$$\forall d \in E_d,\ d \otimes_d u_d = d \tag{8}$$

4.4 Modelling our example

Variables The problem described in Section 3 can be modelled using ten variables. Four of them, denoted as bp_P, bp_J, bp_L and bp_M, represent the presence of Peter's friends at the beginning of the dinner. Four more variables, denoted as ep_P, ep_J, ep_L, and ep_M, represent their presence at the end of the dinner. Each of these variables has two possible values (t or f). The other two variables represent Peter's choices about the menu: mc with two possible values ($fish$ or $meat$) and w with two possible values ($white$ or red). From Peter's point of view, mc and w are non random controllable variables (Peter can set their value), whereas the other eight variables are random uncontrollable variables (Peter cannot decide on their value).

Belief relations Considering beliefs, a first unary belief relation B_1 on bp_P expresses that Paul will come with a probability of 0.7: $P(bp_P = t) = 0.7$, and thus $P(bp_P = f) = 0.3$. A second one B_2 on bp_J expresses that John will certainly come: $P(bp_J = t) = 1$ [5]. This can be equivalently expressed by the unary constraint $bp_J = t$. In fact this random variable, the value of which is certain, could be removed from the problem definition. A third one B_3 on bp_L expresses that Luke will come with a probability of 0.6: $P(bp_L = t) = 0.6$. Moreover, a binary belief relation B_4 on bp_L and bp_M expresses that either Luke and Matthew will both come, or none of them will: $P(bp_M = t|bp_L = t) = 1$ and $P(bp_M = t|bp_L = f) = 0$. This can be equivalently expressed by the binary constraint $bp_L = bp_M$.

The same way, four other belief relations (B_5, B_6, B_7, B_8) can be expressed to model the other specifications of the problem in terms of beliefs:
$B_5 : bp_P = ep_P$
$B_6 : (ep_J = t) \leftrightarrow ((bp_J = t) \wedge (bp_P = f) \wedge (mc = meat))$
$B_7 : (ep_L = t) \leftrightarrow ((bp_L = t) \wedge \neg((mc = meat) \wedge (w = white)))$
$B_8 : (ep_M = t) \leftrightarrow ((bp_M = t) \wedge (w = white))$

B_5, B_6, B_7 and B_8 are expressed here as constraints, but could be expressed as previously as conditional probabilities[6].

[5] Because all the variables in this example have only two possible values, we will express the (conditional) probabilities only for one of the values.

[6] For example, the conditional probability distribution associated with the belief relation B_8 is the following one: $P(ep_M = t|bp_M = t, w = white) = 1$, $P(ep_M = t|bp_M = t, w = red) = 0$, $P(ep_M = t|bp_M = f, w = white) = 0$, and $P(ep_M = t|bp_M = f, w = red) = 0$.

Desire relations Considering desires, a binary desire relation D_1 on mc and w expresses that Peter does not want fish with red wine: $U(mc = fish, w = white) = 0, U(mc = fish, w = red) = -\infty, \ldots$ This can be expressed by the binary constraint $\neg((mc = fish) \wedge (w = red))$.

A ternary desire relation D_2 on ep_P, ep_J, and ep_L expresses that the presence of Paul, John, or Luke at the end of the dinner is required. This can be expressed by the ternary constraint $\neg((ep_P = f) \wedge (ep_J = f) \wedge (ep_L = f))$.

Finally, four unary desire relations D_3, D_4, D_5, and D_6 on ep_P, ep_J, ep_L, and ep_M express the gains that Peter can expect from the presence of each of his friends at the end of the dinner: $U(ep_P = t) = 10$, $U(ep_J = t) = 100$, $U(ep_L = t) = 20$, $U(ep_M = t) = 50$, $U(ep_P = f) = U(ep_J = f) = U(ep_L = f) = U(ep_M = f) = 0$.

Problem structure All these relations are graphically drawn in Figure 1. As with influence diagrams, a square is associated with each non random variable and a circle with each random variable. A hyper-arc, drawn with plain lines, is associated with each belief relation, its arrow being directed to the variable on which conditional probabilities are defined. A hyper-edge, drawn with dotted lines, is associated with each desire relation. Note that belief and desire relations link indifferently controllable and uncontrollable variables, random and non random ones, with the only restriction that a hyper-arc cannot be directed to a non random variable.

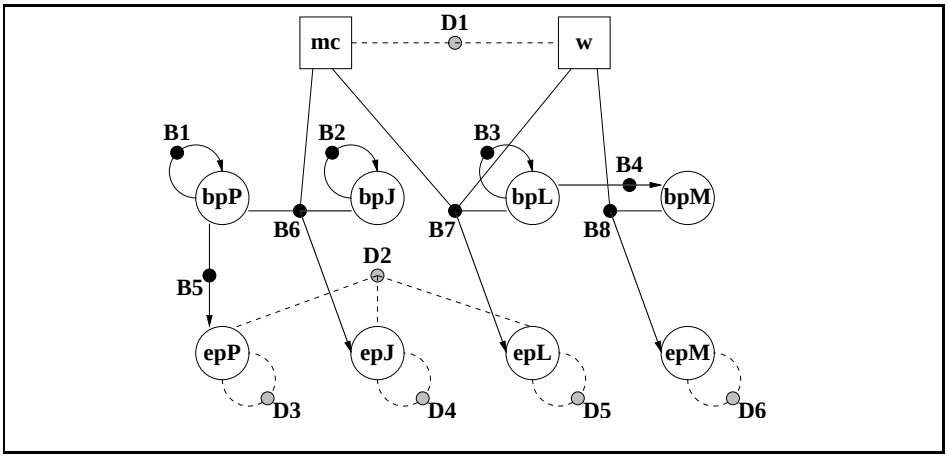


Fig. 1: Hybrid belief and desire network.

Belief and desire structures The belief structure S_b used is the usual structure associated with probabilities: $\langle [0,1], \leq, 0, 1, \times \rangle$. The desire structure S_d is $\langle \mathbb{Z} + \{-\infty\}, \leq, -\infty, 0, + \rangle$. Assumptions of Section 4.3 are satisfied by these structures provided that $+$ is extended as follows: $\forall x \in \mathbb{Z} + \{-\infty\}, x + -\infty = -\infty$.

Belief and desire functions Belief functions ϕ_j are given by local conditional probability distributions P and belief constraints. In particular, $\phi_j(t) = 0$ for all the tuples t that violate a belief constraint. Desire functions ψ_k are similarly given by local utilities U and desire constraints. In particular, $\psi_k(t) = -\infty$ for all the tuples t that violate a desire constraint. Note that, if constraints can be used to partially or completely specify belief and desire relations, these two kinds of relations will be handled separately and differently. This differs from the classical CSP case, where certain facts and hard requirements are represented and handled exactly the same way as constraints.

Semantics Let us consider the complete assignment A where $mc = fish$, $w = white$, $ep_J = f$, and all the other variables are set to t. Its belief degree is the following: $b(A) = \phi_1(bp_P = t) \times \phi_2(bp_J = t) \times \ldots \times \phi_8(w = white, bp_M = t, ep_M = t) = 0.7 \times 1 \times 0.6 \times 1 \times 1 \times 1 \times 1 \times 1 = 0.42$. On the other hand, its desire degree is: $d(A) = \psi_1(mc = fish, w = white) + \psi_2(ep_P = t, ep_J = f, ep_L = t) + \ldots + \psi_6(ep_M = t) = 0 + 0 + 10 + 0 + 20 + 50 = 80$.

4.5 Query definition

We define a query Q as a tuple $\langle Pb, A, Sov, \otimes_{bd}, S_c \rangle$ where:

- Pb is a *problem* $\langle V, B, D, S_b, S_d \rangle$ as defined in Section 4.1;
- A represents the *assignment* of some of the variables in V: the variables the value of which is known or fixed;
- Sov is a sequence of *operator-variable* pairs. Operators may be min, max, or a specific operator $\oplus_c$ (see below); the intersection between the variables in A (denoted as *assigned* variables) and the variables in Sov (denoted as *quantified* variables) is empty; some variables may be neither assigned, nor quantified; they are denoted as *free* variables;
- $\otimes_{bd}$ is a *combination operator* between belief and desire degrees; it associates an element of E_c (see below) with any pair made of an element of E_b and an element of E_d; it computes the combined belief-desire degree from the belief and desire degrees;
- S_c is a quintuple $\langle E_c, \preceq_c, \bot_c, u_c, \oplus_c \rangle$, which defines the combined *belief-desire structure*; E_c is the set of possible combined belief-desire degrees; we assume that, if $E_b \otimes_{bd} E_d = \{b \otimes_{bd} d \,/\, b \in E_b, d \in E_d\}$, E_c is the transitive closure of $E_b \otimes_{bd} E_d$ according to the $\oplus_c$ operator; moreover, we assume that a special element, denoted Δ, is added to E_c to represent facts that are ignored from the combined belief-desire point of view; $\preceq_c$ is a total reflexive binary relation on E_c, such that its restriction on $E_c - \{\Delta\}$ is a total order: facts that are not ignored are ordered and higher degrees are preferred; $\bot_c$ is the minimum element of $E_c - \{\Delta\}$, if such an element exists; it is associated with the least admissible facts from the combined belief-desire point of view; if such an element does not exist, we set that $\bot_c = \Delta$ by default; u_c is an element of $E_c - \{\Delta\}$ which is neutral for $\oplus_c$, if such an element exists; it is associated with indifferent facts from the combined belief-desire point of view; if such

an element does not exist, we set that $u_c = \Delta$ by default; $\oplus_c$ is a binary operator on E_c; as min and max, it will be used to eliminate variables in order to synthesise information about combined belief-desire degrees.

4.6 Query semantics

The answer $Ans(Q)$ to a query $Q = \langle Pb, A, Sov, \otimes_{bd}, S_c \rangle$ is defined as a function of the free variables in Q (those that appear neither in A, nor in Sov). If A' is an assignment of these variables, the value of $(Ans(Q))(A')$ is given by:

$$(Ans(Q))(A') = Qs\,(Pb, A \cup A', Sov, \otimes_{bd}, S_c) \tag{9}$$

Qs being recursively defined as follows:

$$Qs\,(Pb, A'', \emptyset, \otimes_{bd}, S_c) = b(A'') \otimes_{bd} d(A'') \tag{10}$$

$$Qs\,(Pb, A'', < o, i > .\, Sov', \otimes_{bd}, S_c) = \tag{11}$$
$$o_{a \in Dom_i}\,[Qs\,(Pb, A'' \cup \{< i, a >\}, Sov', \otimes_{bd}, S_c)]$$

Equation 10 expresses that, if all the problem variables are assigned, the answer to the query is the combination of the belief and desire degrees. Equation 11 expresses that, if the problem variables are not all assigned and i is the first quantified variable with o as operator, the answer to the query results from the elimination of i using o as elimination operator.

4.7 Required algebraic properties

Some algebraic properties on the combination operator $\otimes_{bd}$ and the combined belief-desire structure S_c are assumed.

Ignored facts

$$\forall c \in E_c,\ max(c, \Delta) = min(c, \Delta) = c \oplus_c \Delta = c \tag{12}$$

Equation 12 expresses that facts the combined belief-desire degree of which equals Δ are *ignored* in the computation of the answer to a query, whatever elimination operators are[7].

Belief-desire combination

$$\forall d \in E_d,\ \bot_b \otimes_{bd} d = \Delta \tag{13}$$

Equation 13 expresses that the combined belief-desire degree of *impossible* facts equals Δ, whatever their desire degrees are. These facts are consequently ignored from the combined belief-desire point of view and do not have any impact on the answer to a query. Naturally, the belief-degree combination operator $\otimes_{bd}$ should be in some sense *monotonic*:

[7] Note that assuming $max(c, \Delta) = min(c, \Delta) = c$ is not incorrect because $\preceq_c$ is not a total order on E_c. Only its restriction on $E_c - \{\Delta\}$ is.

$$\forall b \in E_b, \forall d_1, d_2 \in E_d, \quad (14)$$
$$(d_1 \preceq_d d_2) \rightarrow (b \otimes_{bd} d_1 \preceq_c b \otimes_{bd} d_2)$$

Equation 14 expresses that, if a fact F_2 is more desired than another fact F_1 and both are similarly believed, F_2 is preferred to F_1 from the combined belief-desire point of view. We then assume the existence of an element $\theta_d \in E_d$ which corresponds to a frontier between positively desired and negatively desired facts and satisfies the following equations:

$$\forall b_1, b_2 \in E_b, \forall d \in E_d, \quad (15)$$
$$((b_1 \preceq_b b_2) \wedge (\theta_d \preceq_d d)) \rightarrow (u_c \preceq_c b_1 \otimes_{bd} d \preceq_c b_2 \otimes_{bd} d)$$
$$\forall b_1, b_2 \in E_b, \forall d \in E_d, \quad (16)$$
$$((b_1 \preceq_b b_2) \wedge (d \prec_d \theta_d)) \rightarrow (b_2 \otimes_{bd} d \preceq_c b_1 \otimes_{bd} d \preceq_c u_c)$$

Equation 15 expresses that, if a fact F_2 is more believed than another fact F_1 and both are equally positively desired (desire degree greater than or equal to θ_d), then F_2 is preferred to F_1 from the combined belief-desire point of view. Conversely, Equation 16 expresses that, if a fact F_2 is more believed than another fact F_1 and both are equally negatively desired (desire degree smaller than θ_d), F_1 is preferred to F_2 from the combined belief-desire point of view[8]. We then assume that:

$$(u_c \neq \Delta) \rightarrow (\forall b \in E_b, \ ((b \otimes_{bd} \theta_d = \Delta) \vee (b \otimes_{bd} \theta_d = u_c))) \quad (17)$$

Equation 17 expresses that, provided there exists in $E_c - \{\Delta\}$ a neutral element for $\oplus_c$ ($u_c \neq \Delta$), facts that lie at the frontier between positively and negatively desired facts (desire degree equal to θ_d) are either *ignored* or *indifferent* from the combined belief-desire point of view.

Previous axioms entail that, if there exists a minimum element in $E_c - \{\Delta\}$ ($\bot_c \neq \Delta$) and if ($\bot_d \neq \theta_d$) $\wedge$ ($\top_b \otimes_{bd} \bot_d \neq \Delta$), then $\top_b \otimes_{bd} \bot_d = \bot_c$: if a fact is certain and inadmissible, it is the least admissible from the combined belief-desire point of view. Figure 2 offers a possible representation of a belief-desire space that respects this set of properties.

Belief-desire structure The elimination operator $\oplus_c$ should be *commutative* and *associative*, because we do not want the result of the elimination to be dependent on the way it is performed.

$\oplus_c$ should be *monotonic*, because, if a fact F_2 is preferred to another fact F_1, then the result of the synthesis of F_2 with any third fact F_3 is preferred to the synthesis of F_1 with the same fact F_3:

$$\forall c_1, c_2, c_3 \in E_c - \{\Delta\}, (c_1 \preceq_c c_2) \rightarrow (c_1 \oplus_c c_3 \preceq_c c_2 \oplus_c c_3) \quad (18)$$

[8] Note that we impose neither that θ_d is the only element with such a property, nor that u_d has this property: an indifferent fact from the desire point of view may be non indifferent from the combined belief-desire point of view.

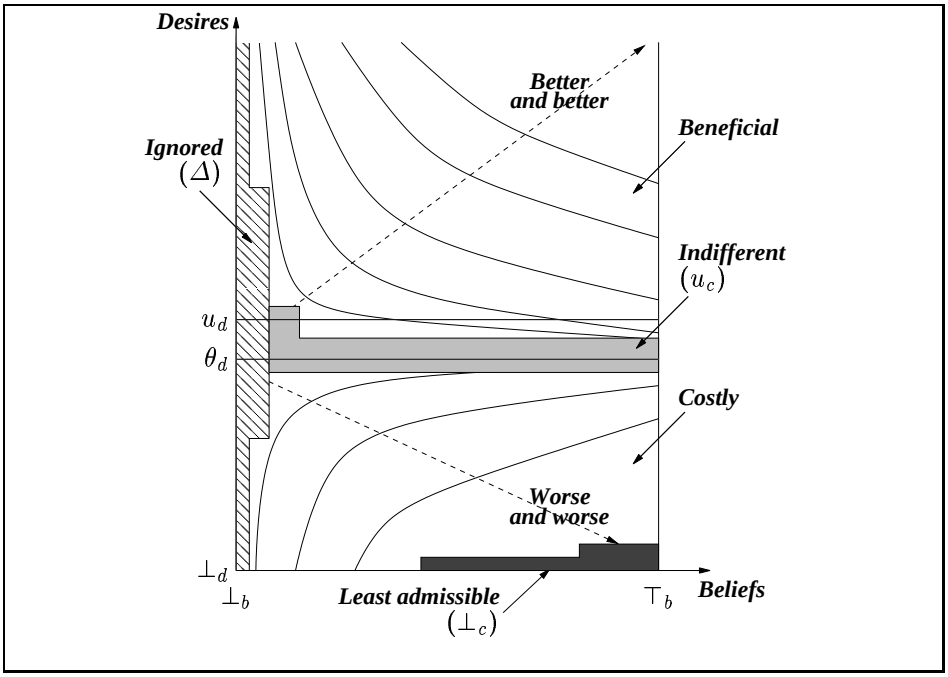


Fig. 2: Abstract representation of the belief-desire space.

Then, the element u_c in E_c should be a *neutral element* for $\oplus_c$, because the synthesis of any not ignored fact F with an indifferent fact does not change F's combined belief-desire degree.

$$\forall c \in E_c - \{\Delta\}\,,\ c \oplus_c u_c = c \tag{19}$$

Note that we do not impose that $\perp_c$ is *annihilator* for $\oplus_c$. This can be assumed if needed. In that case, the least admissible facts (combined belief-desire degree equal to $\perp_c$) become inadmissible.

4.8 Some query examples

Using the example of Section 4.4, the most obvious request consists in asking which decision(s) Peter should take about the menu in order to maximise the expected investment in his company. The associated query can be formulated as follows[9]:

$$< Pb, \emptyset, \{< max, \{mc, w\} > . \\ < \oplus_c, \{bp_P, bp_J, bp_L, bp_M, ep_P, ep_J, ep_L, ep_M\} >\}, \otimes_{bd}, S_c > \tag{20}$$

[9] The actual answer to this query is the maximum expected investment itself, but it can be assumed that the argument(s) of the answer is(are) also available. For all the queries, we assume that the answer contains both the value of the query and the associated value(s) of the variable(s) of interest.

with $\otimes_{bd} = \times$, $\theta_d = 0$, $S_c = \langle \mathbb{R} + \{-\infty\} + \{\Delta\}, \leq, -\infty, 0, + \rangle$, and $\times$ and $+$ extended and modified as follows: $\forall d \in \mathbb{Z} + \{-\infty\},\ 0 \times d = \Delta$, $\forall x \in]0, 1],\ x \times -\infty = -\infty$ and $\forall x \in \mathbb{R} + \{-\infty\},\ x + -\infty = -\infty$. Note that, in this query and in the following ones, successive variables quantified using the same operator are gathered under their common operator[10]. By enumerating all possible cases, one can verify that the answer is 67 k€ (maximum expected investment) with $\{< mc, meat >, < w, white >\}$ as associated decision.

Let us imagine that Peter knows beforehand that Matthew (and then Luke) will not come. The query associated with the same request, taking into account this additional knowledge, can be formulated as follows:

$$< Pb, \{< bp_M, f >\}, \{< max, \{mc, w\} > . \\ < \oplus_c, \{bp_P, bp_J, bp_L, ep_P, ep_J, ep_L, ep_M\} >\}, \otimes_{bd}, S_c > \quad (21)$$

The answer is 37 k€ (30 k€ less than with query 20) with $\{< mc, meat >, < w, white >\}$ or $\{< mc, meat >, < w, red >\}$ as associated decisions (wine does not matter in this case).

Let us imagine now that the restaurant can handle last minute changes for an additional cost of 250 €. In this case, Peter can choose the menu when he knows who is present. The query associated with such a situation can be formulated as follows:

$$< Pb, \emptyset, \{< \oplus_c, \{bp_P, bp_J, bp_L, bp_M\} > . < max, \{mc, w\} > . \\ < \oplus_c, \{ep_P, ep_J, ep_L, ep_M\} >\}, \otimes_{bd}, S_c > \quad (22)$$

The answer is 75.4 k€ (8.4 k€ more than with query 20; gain resulting from the use of the information about the presence of each one at the beginning of the dinner). Thus, it is worthwhile for Peter to pay the additional cost of 250 €.

In a more complex situation, Peter may be forced to choose the menu in advance, but with the knowledge that Paul is in fact a false friend and that he will wait until he knows who is present to decide to come or not with the aim of minimising the expected investment in Peter's company. The query associated with such a situation is:

$$< Pb, \emptyset, \{< max, \{mc, w\} > . < \oplus_c, \{bp_J, bp_L, bp_M\} > . \\ < min, \{bp_P\} > . < \oplus_c, \{ep_P, ep_J, ep_L, ep_M\} >\}, \otimes_{bd}, S_c > \quad (23)$$

The answer is 40 k€ (27 k€ less than with query 20; loss resulting from Paul's betrayal) with $\{< mc, meat >, < w, white >\}$ as associated decision (the same as with query 20).

Ignoring precise belief degrees and distinguishing only possible and impossible facts, it is possible to reason about worst (or best) cases. A possible request consists in asking which decision(s) about the menu maximise(s) the minimum investment. The associated query can be formulated as follows:

[10] For example, $< max, \{mc, w\} >$ is an abbreviation for $< max, mc > . < max, w >$. Such a gathering under a common operator is possible thanks to associativity and commutativity of min, max, and $\oplus_c$ operators.

$$< Pb, \emptyset, \{< max, \{mc, w\} > . \quad (24)$$
$$< min, \{bp_P, bp_J, bp_L, bp_M, ep_P, ep_J, ep_L, ep_M\} >\}, \otimes_{bd}, S_c >$$

with $\forall b \in]0,1], \forall d \in \mathbb{Z} + \{-\infty\},\ b \otimes_{bd} d = d$, $\forall d \in \mathbb{Z} + \{-\infty\},\ 0 \otimes_{bd} d = \Delta$, $\theta_d = 0$ and $S_c = \langle \mathbb{Z} + \{-\infty\} + \{\Delta\}, \leq, -\infty\ , 0, + \rangle$. The answer is 10 k€ with $\{< mc, meat >, < w, white >\}$ or $\{< mc, meat >, < w, red >\}$ as associated decisions (wine does not matter).

Conversely, ignoring desire degrees, it is possible to reason only about belief degrees. A possible request consists in asking which is(are) the most probable situation(s) if Peter chooses fish and white wine. The associated query can be formulated as follows:

$$< Pb, \{< mc, fish >, < w, white >\}, \quad (25)$$
$$\{< max, \{bp_P, bp_J, bp_L, bp_M, ep_P, ep_J, ep_L, ep_M\} >\}, \otimes_{bd}, S_c >$$

with $\forall b \in]0,1], \forall d \in \mathbb{Z} + \{-\infty\},\ (b \otimes_{bd} d = b)$, $\forall d \in \mathbb{Z} + \{-\infty\},\ 0 \otimes_{bd} d = \Delta$, $\theta_d = -\infty$ and $S_c = \langle R^+ - \{0\} + \{\Delta\}, \leq, \Delta, \Delta, + \rangle$. The answer is 0.42 (probability of the most probable situation) with the following associated situation: all Peter's friends are present at the beginning of the dinner and all of them but John are present at the end.

Taking into account the whole information again, Peter may be interested in a decision that avoids, as far as possible, undesirable situations. For example, Peter's request may consist in asking which decision(s) he should take about the menu in order to maximise the probability that the investment is greater than or equal to a given threshold $\theta > -\infty$. The associated query (query 26) is similar to query 20, with some changes related to $\otimes_{bd}$, θ_d, and S_c: $\forall b \in]0,1], \forall d \in \mathbb{Z} + \{-\infty\},\ ((d < \theta) \rightarrow (b \times_{bd} d = 0)) \wedge ((d \geq \theta) \rightarrow (b \otimes_{bd} d = b))$, $\forall d \in \mathbb{Z} + \{-\infty\},\ 0 \otimes_{bd} d = \Delta$, $\theta_d = -\infty$, $S_c = \langle [0,1] + \{\Delta\}, \leq, 0, 0, + \rangle$. With $\theta = 50$ k€, the answer is 0.72 (probability that the investment is greater than or equal to 50 k€) with $\{< mc, meat >, < w, white >\}$ as associated decision.

A related request, which ignores precise beliefs, consists in asking which decision about the menu guarantees that the investment will be greater than or equal to a given threshold $\theta > -\infty$. The associated query (query 27) is similar to query 24, with some changes related to $\otimes_{bd}$, θ_d, and S_c: $\forall b \in]0,1], \forall d \in \mathbb{Z} + \{-\infty\},\ ((d < \theta) \rightarrow (b \otimes_{bd} d = f)) \wedge ((d \geq \theta) \rightarrow (b \otimes_{bd} d = t))$, $\forall d \in \mathbb{Z} + \{-\infty\},\ 0 \otimes_{bd} d = \Delta$, $\theta_d = \theta$, $S_c = \langle \{t, f, \Delta\}, \preceq, f, t, \wedge \rangle$, with $f \prec t$. With $\theta = 50$ k€, the answer is f (no such decision), but with $\theta = 5$ k€, the answer is t, with two associated decisions: $\{< mc, meat >, < w, white >\}$ or $\{< mc, meat >, < w, red >\}$ (wine does not matter).

One can verify that all the assumptions of Section 4.7 are satisfied by the belief-desire combination operators and the combined belief-desire structures used in this section. The *shape* of the combined belief-desire spaces E_c associated with all the queries presented in this section is shown in Figure 3.

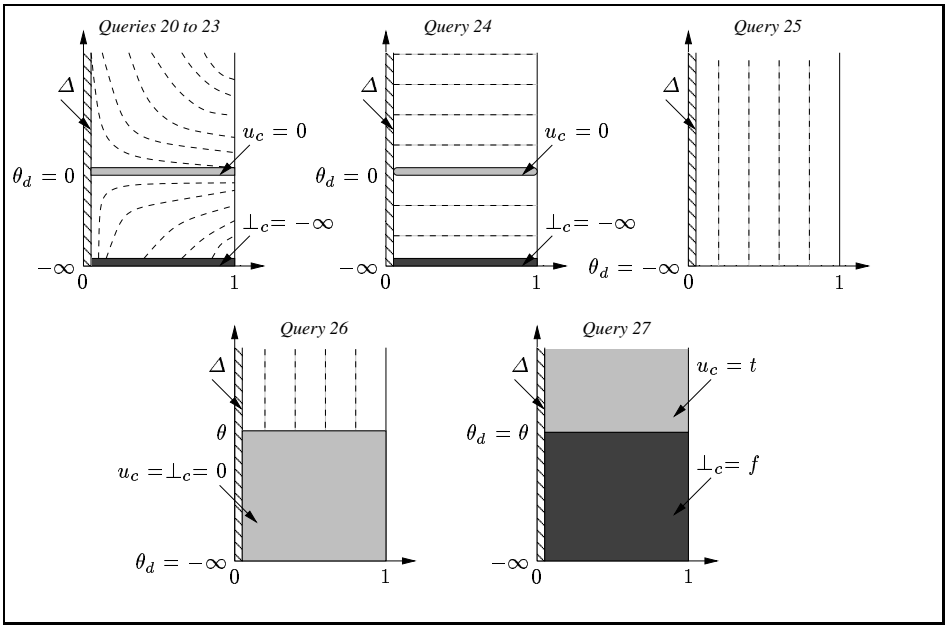


Fig. 3: Belief-desire spaces associated with the various queries.

5 Future work

It can be easily shown that the framework proposed in this paper encompasses classical constraint and probabilistic ones (see Section 1), except for those that admit infinite temporal horizons, such as Markov decision processes over infinite horizons. Moreover, classical requests, such as *belief updating*, *most probable explanation* (MPE), and *maximum expected utility* (MEU) in bayesian networks or influence diagrams, *solution finding* and *solution counting* in constraint satisfaction problems can be easily expressed in the proposed query language.

An important feature of the proposed framework is a clear separation between the *problem* definition, which contains only variables, domains, and belief and desire relations, and the *query* definition, which can be adapted to the actual situation to handle: observability and controllability of the problem variables by the agent itself or by other agents, objectives of the various agents. Section 4.8 showed the wide sprectrum of queries which can be formulated on top of the same problem definition.

But this framework is still in a working stage. Immediate work will consist in studying its main *properties*, as well as those of the numerous sub-frameworks that may result from the addition of extra structural properties.

The final definition of such a framework shall take into account, not only problem and query *representation* issues, but also *complexity* and *algorithmic* issues. It is however almost sure that the proposed framework will inherit the *PSPACE-hardness* of the most general frameworks it subsumes, such as quantified boolean formulae.

Considering algorithmic issues, *non serial dynamic programming* [22], also known as *variable* or *cluster tree elimination*, used in *valuation-based systems* [21] is a natural way to explore. The question of the link between the *structure* of the problem and of the request, on the one hand, and the spatial and temporal *complexity* of these algorithms, on the other hand, is one of the first ones to answer. But other directions such as *branch and bound*, *local search*, and *stochastic sampling* deserve to be explored too.

References

1. Mackworth, A.: Consistency in Networks of Relations. Artificial Intelligence **8** (1977)
2. Dechter, R.: Constraint Processing. Morgan Kaufmann (2003)
3. Bistarelli, S., Montanari, U., Rossi, F., Schiex, T., Verfaillie, G., Fargier, H.: Semiring-Based CSPs and Valued CSPs: Frameworks, Properties and Comparison. Constraints **4** (1999)
4. Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann (1988)
5. Lauritzen, S.: Graphical Models. Oxford University Press (1996)
6. Jensen, F.: Bayesian Networks and Decision Graphs. Springer-Verlag (2001)
7. Howard, R., Matheson, J.: Influence Diagrams. In: Readings on the Principles and Applications of Decision Analysis (1984)
8. Dean, T., Kanazawa, K.: A Model for Reasoning about Persistence and Causation. Computational Intelligence **5** (1989)
9. Bellman, R.: Dynamic Programming. Princeton University Press (1957)
10. Bertsekas, D.: Dynamic Programming: Deterministic and Stochastic Models. Prentice-Hall (1987)
11. Puterman, M.: Markov Decision Processes, Discrete Stochastic Dynamic Programming. John Wiley & Sons (1994)
12. Boutilier, C., Dearden, R., Goldszmidt, M.: Stochastic Dynamic Programming with Factored Representations. Artificial Intelligence **121** (2000)
13. Lovejoy, W.: A Survey of Algorithmic Methods for Partially Observed Markov Decision Processes. Annals of Operations Research **28** (1991)
14. Fargier, H., Lang, J., Schiex, T.: Mixed Constraint Satisfaction: a Framework for Decision Problems under Incomplete Knowledge. In: Proc. of AAAI-96 (1996)
15. Fargier, H., Lang, J., Martin-Clouaire, R., Schiex, T.: A Constraint Satisfaction Framework for Decision under Uncertainty. In: Proc. of UAI-95 (1995)
16. Walsh, T.: Stochastic Constraint Programming. In: Proc. of ECAI-02 (2002)
17. Littman, M., Majercik, S., Pitassi, T.: Stochastic Boolean Satisfiability. Journal of Automated Reasoning **27** (2001)
18. Dechter, R., Larkin, D.: Hybrid Processing of Beliefs and Constraints. In: Proc. of UAI-01 (2001)
19. Dechter, R., Mateescu, R.: Mixtures of Deterministic-Probabilistic Networks and their AND/OR Search Space. In: Proc. of UAI-04 (2004)
20. Bratman, M.: Intention, Plans, and Practical Reason. Harward University Press (1987)
21. Shenoy, P.: Valuation-based Systems for Discrete Optimization. Uncertainty in Artificial Intelligence **6** (1991)
22. Bertelé, U., Brioschi, F.: Nonserial Dynamic Programming. Academic Press (1972)

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. [5, 6] 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. During execution, the plan need only specify the immediate next control action: thus continuous, incremental planning is possible for the problem. 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].

* Present address: SRI International, Menlo Park, CA, USA, nysmith@ai.sri.com

This paper presents an experimental study of anytime solving of mixed CSPs. Specifically, we investigate the performance of the existing decomposition algorithm of [6] 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). A discussion of the results concludes the paper (Section 6).

2 Mixed CSP and the Decomposition Algorithm

2.1 Mixed CSP

Fargier et. al. [5, 6] 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 $\langle \mathcal{V}, \mathcal{D}, \mathcal{C} \rangle$, where $\mathcal{V}$ is a finite set of variables, $\mathcal{D}$ is the corresponding domains, and $\mathcal{C}$ is a set of constraints:

Definition 1 (Mixed CSP [6]). *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

[1] The earlier work [5] associates a probability distribution with each parameter; we follow the later work in which a (discrete) uniform distribution is assumed.

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 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 [5, 6].[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 [6] 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 (after some initial time, perhaps zero, required to provide a first valid solution)
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 additional requirements, such as preemption.

An algorithm to find an optimal conditional decision for a mixed CSP under full observability is presented in [5, 6]. 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 [6].

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* [7]. Formally, an *environment* is a Cartesian product $l_1 \times \cdots \times l_p$, where

[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 ← ∅                                                  {bad realisations}
 2: D ← ∅                                                  {decision–environment pairs}
 3: E ← L_1 × ··· × L_p                                    {environments still to be covered}
 4: repeat
 5:    Choose an environment e from E
 6:    let C_e be constraints that enforce e
 7:    let P be the CSP ⟨Λ ∪ V, L ∪ D, K ∪ C ∪ C_e⟩
 8:    if P is consistent then
 9:       let s be a solution of P
10:       let v be s projected onto the domain variables V
11:       R ← covers(v)                                   {realisations covered by v}
12:       Add the pair (v, R) to D
13:       E ← ⋃_{e'∈E} decompose(e', R)
14:    else                                               {all realisations in e impossible}
15:       Add e to B
16: until E = ∅                                           {all possible realisations covered}
17: return(B, D)
```

$l_i \subseteq L_i$. For example, if $L_1 = L_2 = \{a,b,c,d\}$, then an environment is $\{b,d\} \times \{c,d\}$. 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 [7]; the details

[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 [6].

are unnecessary for this paper. Decomposing an environment e by an environment R means producing a set of distinct environments S that together cover all realisations in e not covered by R.

Using results about environments from [7], in [6] 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 [19, 20].

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:

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

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 parametrised by the planning horizon, $H \in \mathbb{N}$. During execution, the plan need only specify the immediate next control action at the current horizon. However, there is an additional 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.

Three representative but simplified spacecraft sub-systems we consider as SCPP instances are a navigation 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.

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

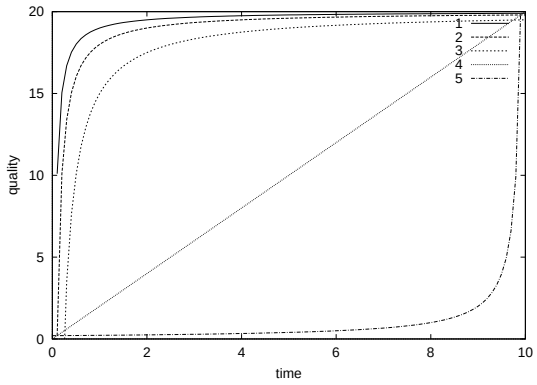


Figure 1. Performance profile curves of idealised anytime behaviour

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 1. 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 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 realisation actually observed is not covered by the conditional decision at the time of interruption, the algorithm does not provide a valid control action.

In [6] 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.

Algorithm 2 Anytime computation by incremental plan horizon

```
 1: S ← ∅
 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 ← 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
```

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 $\mathcal{C}_e$ 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 $\mathcal{C}_e$, 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 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.

4.2 Incremental Horizon

The SCP problem is naturally parametrised 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, $h = 1, \ldots, H$. 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

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

`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 $[0 \ldots h]$, $h \leq H$, 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 $h-1$.

Algorithm 2 summarises the method. As stated, conceptually it operates by solving incrementally larger subproblems. 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 we begin its execution, we can undertake further computation to extend the horizon-$(H-1)$ decisions to horizon-H decisions.

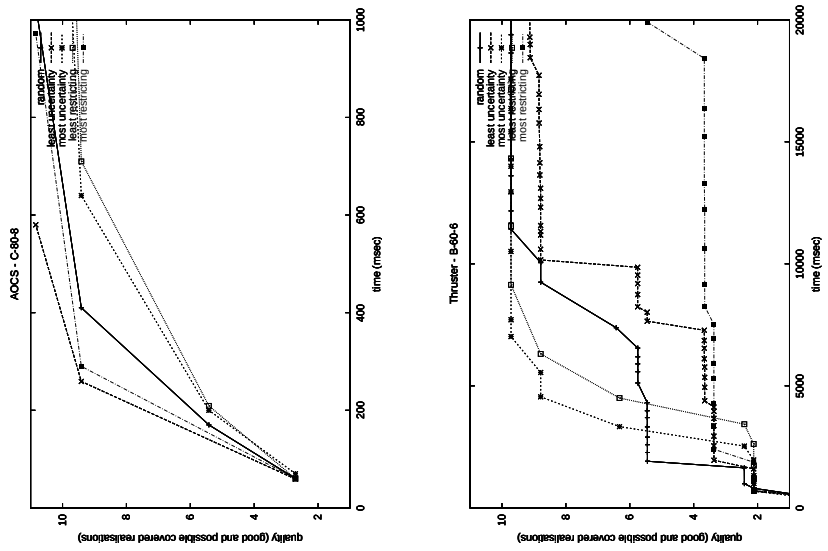
In terms of execution, Algorithm 2 thus has the advantage over Algorithm 1 that an initial, valid action is for certain available (once the problem is solved for horizon 1, which is expected to be rapid). Upon interruption, execution can proceed by checking whether the realisation r^* observed is covered 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.

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. Table 1 summarises the characteristics



(a) AOCS C 80% horizon 8 (b) Thruster B 60% horizon 6

Figure 2. Anytime behaviour of environment selection heuristics (I)

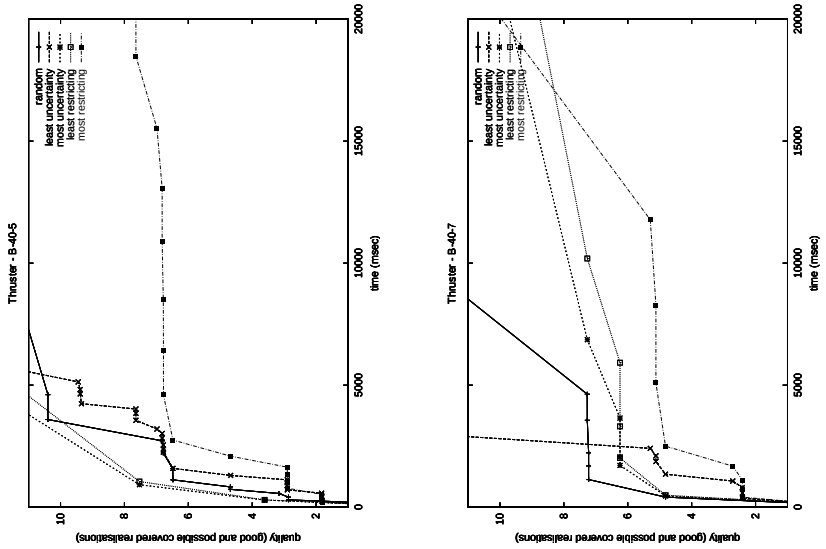
of the three SCPP instances. For each automaton, we considered three degrees of uncertainty: moderate, average and large, denoted A–C respectively. We also considered performance requirements between 20–80% (recall Section 3). This gives two parameters for each problem instance. We imposed a timeout on any single run of Algorithm 1, depending on the complexity of the automaton; the values are 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.

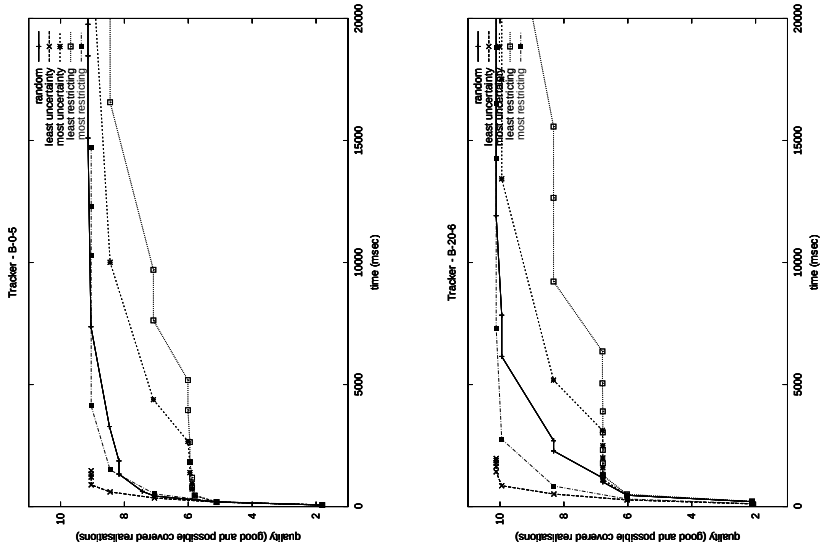
Figures 2(a)–4(b) show the quality (realisations covered) versus solving time (ms). The vertical axis is shown on a log scale, i.e. $\log Q_1(t)$. Figure 2(a) shows the typical result for the AOCS instance: the best heuristic is *least uncertainty*, followed by *most restricting*; these are both better than *random*. The worst heuristic is *least restricting*; *most uncertainty* is slightly better.

Figures 2(b)–3(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,



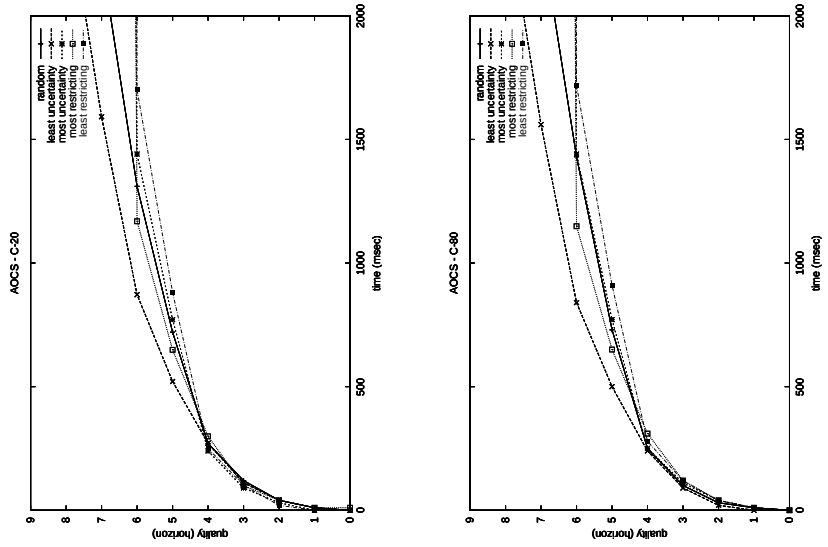
(a) Thruster C 40% horizon 5 (b) Thruster B 40% horizon 7

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



(a) Tracker B 0% horizon 5 (b) Tracker B 20% horizon 6

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



(a) AOCS C 20% (b) AOCS C 80%

Figure 5. Anytime behaviour of incremental horizon (I)

least uncertainty does not have maximal $Q_1(t)$ for all t. First look at Figures 3(a)–3(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.

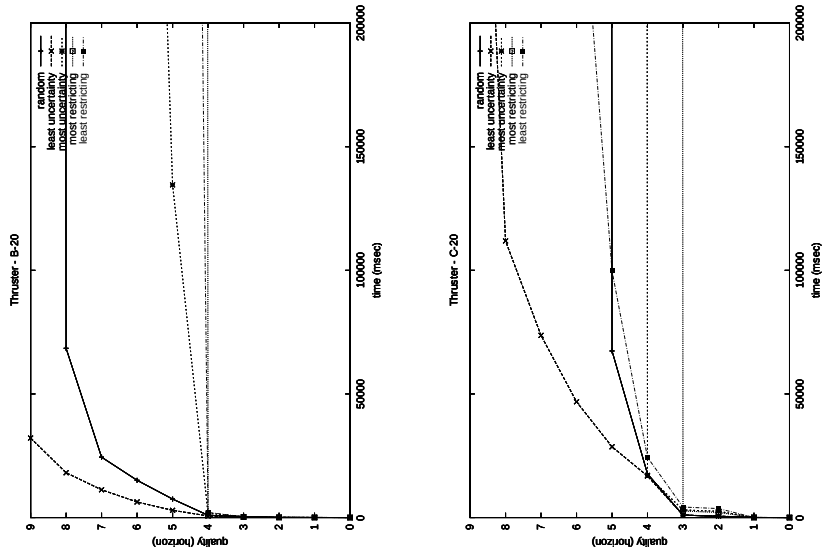
Next look at the rare result shown in Figure 2(b). In this critically constrained problem, *random* is best at first, until overtaken by first *most uncertainty* then *least restricting*. Further, *least uncertainty* exhibits poor anytime performance. While exceptional, this instance indicates that no one heuristic always dominates. As in classical CSPs, the choice of heuristic is itself heuristic.

The results for Tracker confirm those for AOCS. Figures 4(a)–4(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.

5.2 Incremental Horizon

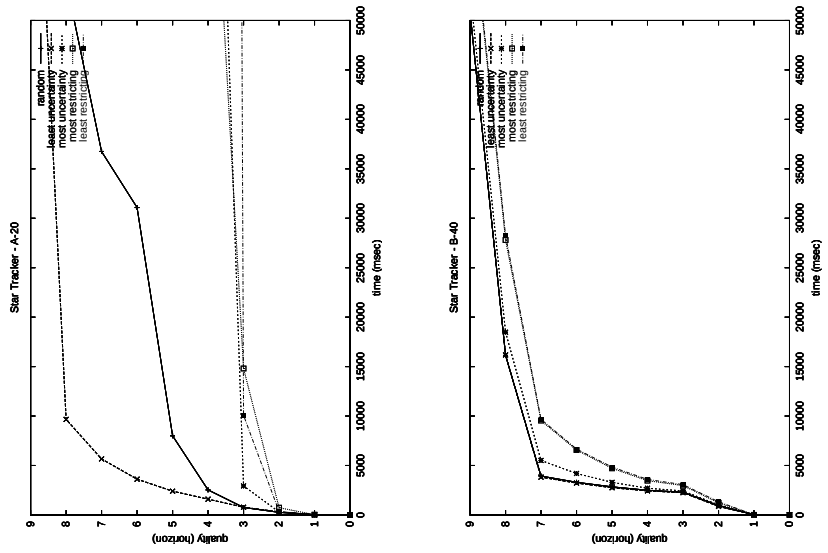
We now consider the method described in Section 4.2. Here, we measure quality by the horizon attained after a given computation time. That is, the problem is solved incrementally for horizons 1, 2, ..., and the times t_i recorded. The cumulative time for horizon h is computed as $t_h = \sum_{i=1,\ldots,h} t_i$, and the quality is $Q_2(t) = \max\{h | t_h \leq t\}$.

Figures 5(a)–7(b) show the quality (horizon attained) versus solving time (ms). The shape of the curves indicate that Algorithm 2 provides acceptable anytime behaviour.



(a) Thruster B 20% (b) Thruster C 20%

Figure 6. Anytime behaviour of incremental horizon (II)



(a) Tracker A 20% (b) Tracker B 40%

Figure 7. Anytime behaviour of incremental horizon (III)

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 6(a)–7(b)), the results are similar. The best heuristic is *least uncertainty*, and overall *random* is next best. For the Tracker instance A 20% (Figure 7(a)), beyond horizon 4, the remaining three heuristics struggle; *most uncertainty* is the best of them. For B 40% (Figure 7(b)), *random* and *least restricting* dominate about equally. The results for Thruster (Figures 6(a)–6(b)), 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.

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 3(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 2(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 2(a) but very poor in Figure 3(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. For the SCPP, by replacing the decomposition algorithm with an incremental version, we ensure anytime behaviour in terms of plan execution.

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 [9]. 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.

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. It can be done with reasonable response time, using for example iterative repair techniques (e.g. [4]). Rather than reacting, our approach here is based on proactive anticipation of the environment or other changes. This has the operational advantage of enabling the system to react more quickly. However, not all environmental uncertainty can be anticipated, and precomputing plans has a computational cost. Thus in practice plan management and execution adopts a hybrid proactive and reactive form [14].

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 in the last section. More generally, this reasoning can take into consideration [8]: 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 [9], but also to solving mixed CSPs with probability distributions over the parameters [5], which are an instance of the stochastic CSP

framework [18]. For instance, scenario sampling methods for stochastic CSPs give the opportunity for an anytime algorithm [10].

Acknowledgement. We thank T. Winterer and the anonymous referees for helpful comments.

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] S. Chien, R. Knight, A. Stechert, R. Sherwood, and G. Rabideau. Using iterative repair to increase the responsiveness of planning and scheduling for autonomous spacecraft. In *Proc. of IJCAI'99 Workshop on Scheduling and Planning meet Real-time Monitoring in a Dynamic and Uncertain World*, Stockholm, Sweden, Aug. 1999.

[5] 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.

[6] 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.

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

[8] 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.

[9] E. Hebrard, B. Hnich, and T. Walsh. Super solutions in constraint programming. In *Proc. of CP-AI-OR'04*, pages 157–172, Nice, France, 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, Seattle, WA, Aug. 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, Ithaca, NY, 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, June 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.

The 2-Expert Approach to Online Constraint Solving

William S. Havens and Bistra N. Dilkina

Intelligent Systems Lab
Simon Fraser University
Burnaby, British Columbia Canada V5A 1S6
{havens, bnd}@cs.sfu.ca

Modern constraint programming (CP) applications are inherently both online systems and mixed-initiative systems. Or at least they should be so. Applications such as satellite tasking, day of operations planning, maintenance scheduling and vehicle dispatching exhibit these requirements. The constraint satisfaction problems (CSPs) they solve are not static but change unpredictably during the schedule generation process. Likewise, these applications usually involve mixed-initiative reasoning where the user and the system work together to solve the CSP. The user is also an expert in the problem domain who embodies unstated constraints and may know good heuristics for finding solutions. Acknowledging this, the so-called *2-expert approach* attempts to support collaboration between the user and system instead of completely automating the solution process.

Traditional CP algorithms do not serve online and mixed-initiative applications well. New architectures are required which support: 1) dynamic constraint propagation methods; and 2) semi-systematic search algorithms. We believe that these two areas should be complementary foci of CP research. Our rationale is as follows.

Online systems confront CSPs that change dynamically. Changes in the problem (environment) are seen as non-monotonic constraint addition, deletion and revision in the CSP model. Likewise, mixed-initiative reasoning views the user as an induced k-ary constraint on the CSP. Some k variables in the model are controllable (in part) by the user as unary constraints on these variables. This induced user constraint is also non-monotonic, changing spontaneously via user interaction and guidance of the solution process. Thus constraint propagation algorithms need to be incremental, reversible and efficient. Incremental propagation methods perform variable domain revision by considering only the changed constraints and not recomputing consistency from scratch [2, 4]. Propagation must be reversible such that non-monotonic constraint retraction can also be computed incrementally. And dynamic constraint propagation needs to be efficient to be used with non-systematic local search techniques. Existing algorithms for dynamic arc consistency are probably too slow.

As well, dynamic CSPs put strong demands on search algorithms. We require that the CP system accommodate to the dynamics of online problem changes and user choices. The system must continue searching from its current (partial) solution and not restart after every change. Traditional constructive (backtrack) methods are inappropriate because they search a tree of partial assignments

which assumes that the CSP is static. Branches pruned by dynamic constraints need to be reconsidered when these constraints are retracted or revised. Purely local search algorithms avoid this problem because they relinquish systematicity in order to follow solely the heuristic gradient (hillclimbing) in the CSP. But local search does not exploit the benefits of systematically by exhausting search subspaces before moving onto other regions. No long term memory (of nogoods) is maintained of previously visited regions. The memory requirements in general are too large.

We believe, however, that new semi-systematic search algorithms are appropriate for online and mixed-initiative CSPs. These algorithms combine desirable aspects of both constructive and local search (*e.g.* [3]). They operate on a (nearly) complete assignment of variables thus maximizing the effectiveness of heuristics. They provide the freedom to move arbitrarily in the search space yet retain enough systematicity to efficiently but incompletely tour that space. We have developed a method [2] that hillclimbs in the infeasible space looking for an assignment of variables which minimizes the number of unsatisfied constraints (minConflicts). Any violated constraints at a local minima are used to derive new nogoods which preclude any subsequent global assignment from containing these values. Essentially the method backtracks over local minima instead of possible global assignments. As in constructive search, deep backtracking is performed when every assignment to some variable is nogood in the current global environment. However, deep backtracking is strongly dependent both on the variable ordering relation which chooses which variable to move next and on the nogood caching scheme used to remember disallowed previous states. The efficacy of this hybrid search algorithm has been recently demonstrated on a large sports scheduling application [1].

In conclusion, we believe that a seachange is underway in how people solve real CSPs. Static batch programming is dead. For CP, this revolution necessitates the development of more flexible and dynamic constraint solving algorithms. We have built a framework for exploring semi-systematic constraint solving, called *ReSolver*, and are currently investigating a number of interesting instances of the framework including the instance described above.

References

1. B.N. Dilkina & W.S. Havens (2004) The National Football League Scheduling Problem, *proc. 16th Innovative Applications of Artificial Intelligence Conference (IAAI-04)*, San Jose, California, July 2004, pp.814-819.
2. W.S. Havens & B.N. Dilkina (2004) Systematic Local Search for Constraint Satisfaction Problems, *proc. 17th Canadian Conf. on Artificial Intelligence*, Lecture Notes in Computer Science vol 3060 (Springer Verlag), London, Ont., May 2004, pp.248-260.
3. N. Jussein and O. Lhomme (2002) Local search with constraint propagation and conflict-based heuristics, *Artificial Intelligence* 139:21-45.
4. S. Prestwich (2001) Local Search and Backtracking vs Non-Systematic Backtracking, *proc. AAAI 2001 Fall Symposium on Using Uncertainty within Computation.*

Facing Executional Uncertainty through Partial Order Schedules

Nicola Policella[1⋆], Amedeo Cesta[1], Angelo Oddi[1], and Stephen F. Smith[2]

[1] Institute for Cognitive Science and Technology
Italian National Research Council
Rome, Italy
{policella|a.cesta|a.oddi}@istc.cnr.it
[2] The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA, USA
sfs@cs.cmu.edu

The usefulness of schedules in most practical scheduling domains is limited by their brittleness. Though a schedule offers the potential for a more optimized execution than would otherwise be obtained, it must in fact be executed as planned to achieve this potential. In practice, this is generally made difficult by a dynamic execution environment, where unexpected events quickly invalidate the schedule's predictive assumptions and bring into question the continuing validity of the schedule's prescribed actions. The lifetime of a schedule tends to be very short, and hence its optimizing advantages are generally not realized.

In our work we go beyond the classical, fixed-times formulation of the scheduling problem, which designates the start and end times of activities as decision variables and requires specific assignments to verify resource feasibility. In fact, adopting a graph formulation of the scheduling problem, wherein activities competing for the same resources are simply ordered to establish resource feasibility, it is possible to produce schedules that retain temporal flexibility where problem constraints allow. In essence, such a "flexible schedule" encapsulates a set of possible fixed-times schedules, and hence is equipped to accommodate some amount of executional uncertainty. More precisely, our approach adopts a graph formulation of the scheduling problem and focuses on generation of *Partial Order Schedules* (POSs) [1]. Within a POS, each activity retains a set of feasible start times, and these options provide a basis for responding to unexpected disruptions. An attractive property of a POS is that reactive response to many external changes can be accomplished via simple propagation in the associated temporal network (a polynomial time calculation); only when an external change exhausts all options for an activity it is necessary to recompute a new schedule from scratch. This assures a bound to the cost of reacting to unforeseen events. Given this property and given a predefined horizon H, the *size* of a POS – the number of fixed-times schedules (or possible execution futures) that it "contains" – is suggestive of its overall robustness[3]. In general, the greater the size of a POS the more robust it is. Thus, our challenge is to generate POSs of maximum possible size.

⋆ Ph.D. student at the Department of Computer and Systems Science, University of Rome "La Sapienza", Italy.

[3] The use of a finite horizon is justified by the need to compare POSs of finite size.

One important open question, though, is how to generate flexible schedules with good robustness properties. In [2] a two-stage approach to generate a flexible schedule is introduced as one possibility. Under this schema, a feasible fixed-times schedule is first generated in stage one (in this case, an early start times solution), and then, in the second stage, a procedure referred to as *chaining* is applied to transform this fixed-times schedule into a temporally flexible schedule where activities competing for the same resources are linked into precedence chains (*Chaining Form*). In a recent paper [1], this approach – find a solution then make it flexible – was shown to produce schedules – for resource constraint project scheduling with generalized precedence relation, RCPSP/max – with better robustness properties than a more direct, least-commitment generation procedure. The least commitment approach uses computed bounds on cumulative resource usage (i.e., a resource envelope [3]) to identify potential resource conflicts, and progressively winnows the total set of temporally feasible solutions into a smaller set of resource feasible solutions by resolving detected conflicts. The two step approach, instead uses conflict analysis of a specific (i.e., earliest start time) solution to generate an initial fixed-time schedule, and then expands this solution to a set of resource feasible solutions in a post-processing step.

The previous results have established the basic viability of a chaining approach. In [4] we focused specifically on the problem of generating POSs in *Chaining Form*. The paper pointed out two basic properties: (1) that a given POS can always be represented in Chaining Form; and (2) that chaining - the process of constructing a Chaining Form POS - is makespan preserving with respect to an input schedule. As a consequence of the last point, in the case of a makespan objective, the robustness of a schedule can be increased without degradation to its solution quality. On the basis of the first property, we have considered the possibility of producing POSs with better robustness and stability properties through more extended search in the space of Chaining Form POSs. In particular, an analysis of structural properties of more robust Chaining Form POSs to heuristically bias chaining decisions has been exploited. In general, consideration of Chaining Form POSs has emphasized the presence of synchronization points among chains as obstacles to flexibility, and this fact has been exploited to generate POSs with good robustness properties.

One of the current goal is the definition of broader search strategies that use a chaining operator as a core component. In fact, the result obtained by any chaining operator is biased by the initial solution that seeds the chaining procedure. From this perspective, one point to investigate is the relation between the initial solution and the partial order schedule which can be obtained through chaining.

References

1. Policella, N., Smith, S.F., Cesta, A., Oddi, A.: Generating Robust Schedules through Temporal Flexibility. In: Proceedings of ICAPS'04. (2004)
2. Cesta, A., Oddi, A., Smith, S.F.: Profile Based Algorithms to Solve Multiple Capacitated Metric Scheduling Problems. In: Proceedings of AIPS-98. (1998)
3. Muscettola, N.: Computing the Envelope for Stepwise-Constant Resource Allocations. In: Proceedings of CP 2002. (2002)
4. Policella, N., Oddi, A., Smith, S.F., Cesta, A.: Generating Robust Partial Order Schedules. In: Proceedings of CP 2004. (2004)

A Controller for Online Uncertain Constraint Handling

Alfio Vidotto, Kenneth N. Brown, J. Christopher Beck

Cork Constraint Computation Centre,
Dept. of Computer Science, UCC, Cork, Ireland
av1@student.cs.ucc.ie, k.brown@cs.ucc.ie, c.beck@4c.ucc.ie

An online problem is a problem which grows over time and such that partial solutions are to be generated before the complete problem is known. Moreover, if the problem is an optimization problem, partial solutions must be aimed at optimizing the overall final solution. There may be some uncertain knowledge on how the problems develop. How should we make intermediate decisions? Can we extend existing constraint handling techniques?

In *Online Uncertain Constraint Handling* (OUCH!), we assume that the problem starts with a conditional constraint optimization problem (CCOP). At each time step, an extension to the CCOP may arrive; that is, a set of variables, constraints and utility functions. Each variable will have a decision deadline, and a decision on that variable must be committed to by that deadline. We assume that decisions cannot be revised once they have been committed to, and also that there is no benefit in making an early commitment. The CCOP will allow us to 'reject' variables, will state what that means for each constraint, and will determine the appropriate reward. The objective will be to maximize the total reward over some (possibly infinite) time interval. Specifically, at each time step, we must decide what to do with the variables whose decision deadline has arrived, balancing the immediate reward with the potential for future rewards. If we have a probability distribution for the CCOPs that arrive at each timestep, we can express the future reward in terms of maximum expected utility. The best decision for a set of variables at time step i is:

$$\text{argmax}_{decision}\ [\textit{reward for decision} + \textit{max expected future reward}]$$

How should we now reason about those probabilities? [1] attempts to search and propagate constraints over the implied tree of possible futures; [2] samples possible futures, and then selects an action which minimises regret for that sample. In this project, we will investigate instead the use of heuristic estimates of the maximum expected utility, and, similar to [3], we will control the parameters of the heuristic by comparison with the performance relative to the optimum decisions for the observed history. We will use a flexibility measure to estimate the reward obtained from each decision, by examining the domains of the remaining variables, and combine the flexibility with the known reward by a weight parameter ($\alpha/(1-\alpha)$). The decision for a set of variables at time step i will be:

$$\text{argmax}_{decision}\ [\textit{reward for decision} + \alpha/(1-\alpha) * \textit{flexibility}]$$

The controller's main task is to tune α to get the best estimate. Our controller (*fig 1*) reacts periodically, i.e. we adjust α depending on the history over the last T time steps. For our initial studies, we will consider optimising the number of variables we accept (and assign consistent values). The extreme cases will be where $\alpha = 0$,

where we assign every variable we can without regard to the future, and $\alpha \to 1$, where we reject every variable, to maintain maximum flexibility. We expect the optimal value of α to be somewhere in between. Our current idea is to maintain an initial α until the loss of reward compared to the maximum we could have achieved exceeds a certain tolerance. Reward may be lost for two reasons: we could have assigned variables but chose to reject them, in which case the flexibility was dominant; or we are forced to reject variables because no consistent value is possible, and thus the immediate reward was dominant. In the first case, we need to decrease α, otherwise, we need to increase it.

The work of the controller can be further extended. If we don't know anything about the distribution of the future problem extensions, then looking back at the history is the only definite knowledge we have. We can use the controller to try to learn the distribution. For example, we might start with a uniform distribution, and gradually adjust the probabilities by observing the actual sequences.

We are currently applying this general idea to a specific online packing problem [4]. Future work concerns the implementation of the controller, developing general flexibility heuristics, and comparing to existing online constraint solving approaches.

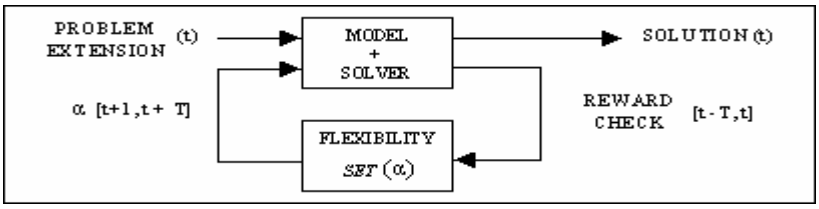


Fig. 1. Flexibility Controller

References

1. D. W. Fowler and K. N. Brown, "Branching constraint satisfaction problems and Markov Decision Problems compared", *Annals of Operations Research*, Volume 118, Issue 1-4, pp85-100, 2003.
2 R. Bent and P. Van Hentenryck. Regrets Only! Online Stochastic Optimization under Time Constraints, Proc. AAAI-04, 2004.
3 L. S. Crawford, M. P. J. Fromherz, C. Guettier, and Y Shang. A Framework for On-line Adaptive Control of Problem Solving. In: *CP'01 Workshop on On-line Combinatorial Problem Solving and Constraint Programming*, Dec. 2001.
4 A. Vidotto, "Online Constraint Solving and Rectangle Packing", to appear in *Proc CP2004 (Doctoral programme)*, 2004.

Acknowledgements: This work has received support from Enterprise Ireland (SC/2003/81), Science Foundation Ireland (00/PI.1/C075), and ILOG, SA. We thank Christine Wei Wu for many useful discussions on this work.

Dynamic Vehicle Routing With Uncertain Customer Demand

Christine Wei Wu, J. Christopher Beck, and Kenneth N Brown

Cork Constraint Computation Center,
Department of Computer Science, University College Cork, Cork, Ireland
cww1@cs.ucc.ie

Many difficult combinatorial problems have been modelled as standard CSPs and solved by classic constraint programming techniques. However, in practice, many problems are dynamic, uncertain and changing, while the decisions have to be made before the full problem is known. For example, in the Dynamic Vehicle Routing Problem (DVRP) [1], new customer orders appear over time, and new routes must be calculated as we are executing the existing solution. In the literature, there are many methods and strategies have been proposed to tackle DVRPs. [2] considered a DVRP as the extension to the standard VRP by decomposing a DVRP as a sequence of static VRPs and then solving them with ant colony system algorithm. [3] used a reactive method (agent-based constraint programming) to solve DVRPs. [4] introduced a consensus approach to the problem.

We are considering an alternative dynamic version of the VRP, in which the customers have uncertain demand. Specifically, our vehicles carry multiple types of product; each customer has an initial reported demand for certain quantities of each product; when the vehicle reaches the customer, the customer may change their request based on the current contents of the vehicle. As an example, consider a baker's delivery van carrying different types of bread. Customer A originally requested 20 white loaves. When the van arrives, A sees sun-dried tomato bread and decides instead to take 10 white loaves and 10 sub-dried tomato loaves. The van is now short of the loaves requested by Customer B. Should the van continue to B, change its route to reload at the bakery, or request another van to visit B instead? There has been some previous research on VRPs with uncertain demand. Erera and Daganzo in [5] proposed strategies allowing vehicles to cooperate by minimizing and approximating "logistic cost function". A Cross-Entropy heuristic and integer linear programming have been applied in [6]. However, all of that research restricts the problem to single commodity vehicles, and most of it examines reactive approaches.

We assume that we have probability distributions of the customer demand, and we want to use this information to make better decisions. Our intention is to use constraint programming and scheduling techniques to search for solutions at each time step, aiming to get close to the retrospectively optimal solution. In general, we aim to combine off-line robust plans with on-line rescheduling, using probabilistic models of the future. Bent & van Hentenryck have proposed a regret algorithm for online stochastic optimization under time constraints. It solves as many samples as possible and avoids distributing them among requests

[7]. Our initial plan is to adapt this approach to our problem, and then extend it to problems with more combinatorial constraints. Finally, we will attempt to extend our solutions to other problems, including dynamic bin packing and online scheduling, based on the features they have in common with VRPs [8].

Acknowledgements

This work has received support from Enterprise Ireland (SC/2003/81), Science Foundation Ireland (00/PI.1/C075), and ILOG, SA.

References

1. R. Bent & P. Van Hentenryck: *Scenario based planning for partially dynamic vehicle routing problems with stochastic customers.* Operations Research. (to appear) (2001)
2. R. Montemanni, L.M. Gambardella, A.E. Rizzoli & A.V. Donati: *A new algorithm for a dynamic vehicle routing problem based on ant colony system.* url: citeseer.ist.psu.edu/563027.html.
3. K. Zhu & K. L. Ong: *A reactive method for real time dynamic vehicle routing problem.* In 12th ICTAI 2000, Vancouver, Canada, (2000).
4. R. Bent & P. Van Hentenryck: *The value of consensus in online stochastic scheduling.* In the Fourteenth International Conference on Automated Planning and Scheduling (2004).
5. A. L. Erera & C. F. Daganzo: *Coordinated vehicle routing with undcertain demand.* Route 2000, International workshop on Vehicle rounting, Skodsborg, Denmark (2000).
6. K. Chepuri & T. Homem-de-Mello: *Solving the vehicle routing problem with stochastic demands using the cross entropy method.* Department of Industrial, Systems Engineering, The Ohio State University, Columbus, OH, USA (2004)
7. R. Bent & P. Van Hentenryck: *Regrets only. Online stochastic optimization under time constraints.* Proceedings of the 19th National Conference on Artificial Intelligence (AAAI'04) San Jose, CA (2004).
8. J.C. Beck, P. Prosser, E. Selensky: *Vehicle routing and job shop scheduling: What's the difference.* Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling (2003).