Interleaving Solving and Elicitation of Constraint Satisfaction Problems Based on Expected Cost

Nic Wilson, Diarmuid Grimes and Eugene C. Freuder

Cork Constraint Computation Centre
Department of Computer Science
University College Cork, Ireland
n.wilson@4c.ucc.ie, d.grimes@4c.ucc.ie, e.freuder@4c.ucc.ie

Abstract. We consider Constraint Satisfaction Problems in which constraints can be initially incomplete, where it is unknown whether certain tuples satisfy the constraint or not. We assume that we can determine such an unknown tuple, i.e., find out whether this tuple is in the constraint or not, but doing so incurs a known cost, which may vary between tuples. We also assume that we know the probability of an unknown tuple satisfying a constraint. We define algorithms for this problem, based on backtracking search. Specifically, we consider a simple iterative algorithm based on a cost limit on which unknowns may be determined, and a more complex algorithm that delays determining an unknown in order to estimate better whether doing so is worthwhile. We show experimentally that the more sophisticated algorithms can greatly reduce the average cost.

1 Introduction

In Constraint Satisfaction Problems it is usually assumed that the CSP is available before the solving process begins, that is, the elicitation of the problem is completed before we attempt to solve the problem. As discussed in the work on Open Constraints and Interactive CSPs [1–5], there are situations where it can be advantageous and natural to interleave the elicitation and the solving. We may not need all the complete constraints to be available in order for us to find a solution. Furthermore, it may be expensive, in terms of time or other costs, to elicit some constraints or parts of the constraints, for example, in a distributed setting. Performing a constraint check in certain situations can be computationally very expensive. We may need to pay for an option to be available, or for the possibility that it may be available. Some constraints may be related to choices of other agents, which they may be reluctant to divulge because of privacy issues or convenience, and so it could cost us something to find these out. Or they may involve an uncertain parameter, such as the capacity of a resource, and it could be expensive, computationally or otherwise, to determine more certain information about this.

Article published in Constraints Volume 15, Number 4, 540-573,
DOI: 10.1007/s10601-010-9099-7
http://www.springerlink.com/content/b145m83381800x8v/
The final publication is available at www.springerlink.com

In this paper we consider approaches for solving such partially-specified CSPs which take these costs into account. Constraints may be initially incomplete: it may be unknown whether certain tuples satisfy the constraint or not. It is assumed in our model that we can determine such an unknown tuple, i.e., find out whether this tuple is in the constraint or not, but doing so incurs a known cost, which may vary between tuples. We also assume that we know the probability of an unknown tuple satisfying a constraint. An optimal algorithm for this situation is defined to be one which incurs minimal expected cost in finding a solution.

Example

To illustrate this situation, consider a problem with two variables X and Y, where X takes values 1, 2, 3 and 4, so $D(X) = \{1, 2, 3, 4\}$, and the domain, $D(Y)$, of Y is $\{5, 6\}$. Consider, for example, that we are trying to arrange a football match, choosing one of four potential locations (the values of X), and one of two potential time-slots (the values of Y). In order for a pair (x, y) to be a feasible solution, we need to find out if (1) we have permission to use football pitch x, and (2) if x is available at time y. There are two incomplete constraints (see Table 1), the first, c_1, is a unary constraint on X, representing which football pitches we have permission to use, and the second, c_2, is a binary constraint on the two variables, representing which pitches are available at which time-slot.

It is (currently) unknown if we have permission to use the first pitch, i.e., if $X = 1$ satisfies constraint c_1. The probability p_1 that it does so is 0.9. We can determine (i.e., find out) if $X = 1$ satisfies constraint c_1, but this test incurs a cost of $K_1 = 50$ (which in this example is a measure of the effort needed to find out the information). We write $c_1(X = 1) = u_1$, where u_1 represents an unknown Boolean value. If it turns out that $u_1 = 1$ then $X = 1$ satisfies constraint c_1; otherwise, $u_1 = 0$, and $X = 1$ does not satisfy c_1. It is also unknown if values $2, 3$ and 4 satisfy c_1. We have $c_1(X = 2) = u_2$, $c_1(X = 3) = u_3$ and $c_1(X = 4) = u_4$. The cost of determining unknowns u_2, u_3 and u_4 is each 70, and the probability of success of each is 0.8.

Tuples $(2, 6)$, $(3, 5)$ and $(4, 6)$ all satisfy the binary constraint, whereas tuples $(2, 5)$, $(3, 6)$ and $(4, 5)$ do not. It is unknown whether tuples $(1, 5)$ and $(1, 6)$ satisfy the constraint, i.e., whether the first pitch is available at the two time-slots (because, although matches were provisionally booked for these times, there is a possibility that these matches have been cancelled). We have $c_2(X = 1, Y = 5) = u_5$, and $c_2(X = 1, Y = 6) = u_6$, and $c_2(X = 2, Y = 6) = c_2(X = 3, Y = 5) = c_2(X = 4, Y = 6) = 1$. The other tuples have value 0. Unknowns u_5 and u_6 each have cost 200 and probability of success 0.1. Hence there is a 10% chance that $X = 1, Y = 5$ satisfies c_2, and it costs 200 units to determine this. For $i = 1, \ldots, 6$, we write K_i as the cost of determining unknown u_i, and p_i for the probability of success, i.e., that $u_i = 1$.

Consider a standard backtracking search algorithm with variable ordering X, Y and value ordering $1, 2, 3, 4$ for X and $5, 6$ for Y. The algorithm will first incur a cost of 50 in determining u_1. This unknown will be determined successfully with 90% chance, and if so, then $X = 1$ satisfies c_1. After that, u_5 will

be determined, costing 200, but with only 0.1 chance of success. If both u_1 and u_5 are successfully determined then $(X = 1, Y = 5)$ is a solution of the CSP. However, this has only chance $p_1 \times p_5 = 0.9 \times 0.1 = 0.09$ of happening, and cost $K_1 + K_5 = 50 + 200 = 250$ is incurred (see Figure 1).

Table 1: The unary constraint c_1 and the binary constraint c_2, along with costs (K_i) and probabilities (p_i) of their associated unknowns.

	$X = 1$	$X = 2$	$X = 3$	$X = 4$
$c_1(X)$	u_1	u_2	u_3	u_4
(K_i, p_i)	$(50, 0.9)$	$(70, 0.8)$	$(70, 0.8)$	$(70, 0.8)$

$c_2(X, Y)$	$X = 1$	$X = 2$	$X = 3$	$X = 4$
$Y = 5$	u_5	0	1	0
(K_i, p_i)	$(200, 0.1)$			
$Y = 6$	u_6	1	0	1
(K_i, p_i)	$(200, 0.1)$			

It can be shown that the expected cost incurred by this algorithm is approximately 464 and can be written as $E_1 + qE_2$, where $q = 0.1 + 0.9^3 = 0.829$, and E_1 and E_2 are defined as follows: $E_1 = K_1 + p_1(K_5 + (1 - p_5)K_6) = 50 + 0.9(200 + 0.9 \times 200) = 392$; and $E_2 = K_2 + (1 - p_2)(K_3 + (1 - p_3)K_4) = 70 + 0.2(70 + 0.2 \times 70) = 86.8$. E_1 is the expected cost in the $X = 1$ branch, E_2 is the expected cost conditional on having reached the $X = 2$ constraint check, and q is the chance that the algorithm fails to find a solution with $X = 1$. This is far from optimal, mainly because determining unknowns u_5 and u_6 is very expensive, and they also have only small chance of success. An optimal algorithm for this problem (i.e., one with minimal expected cost) can be shown to have expected cost $E_2 + (0.2^3 \times 389.5) \approx 90$, which can be achieved with a backtracking search algorithm which determines unknowns in the $X = 2, 3, 4$ branches before determining unknowns in the $X = 1$ branch.

In more detail: we can first determine u_2, with cost 70 and chance of success 0.8. If $u_2 = 1$ then $(X = 2, Y = 6)$ is a solution and we stop. If $u_2 = 0$ we next determine u_3; if $u_3 = 1$ we have a solution $(X = 3, Y = 5)$. If $u_3 = 0$ we next determine u_4. The expected cost so far is equal to E_2. If we have not found a solution so far, i.e., if $u_2 = u_3 = u_4 = 0$ (which happens with chance $(1 - p_2)(1 - p_3)(1 - p_4) = 0.2^3 = 0.008$) we next determine u_5 with cost 200. If $u_5 = 1$ we determine u_1; if $u_1 = 1$ then we have a solution $(1, 5)$; if $u_1 = 0$ then there exists no solution. If $u_5 = 0$ then we determine u_6, and then u_1 if $u_6 = 1$. The expected cost can then be written as $E_2 + (0.2^3 \times E_1')$ where $E_1' = K_5 + p_5K_1 + (1 - p_5)(K_6 + p_6K_1) = 200 + 0.1 \times 50 + 0.9 \times (200 + 0.1 \times 50) = 389.5$. The expected cost is therefore equal to $86.8 + (0.008 \times 389.5) = 89.916$. ■

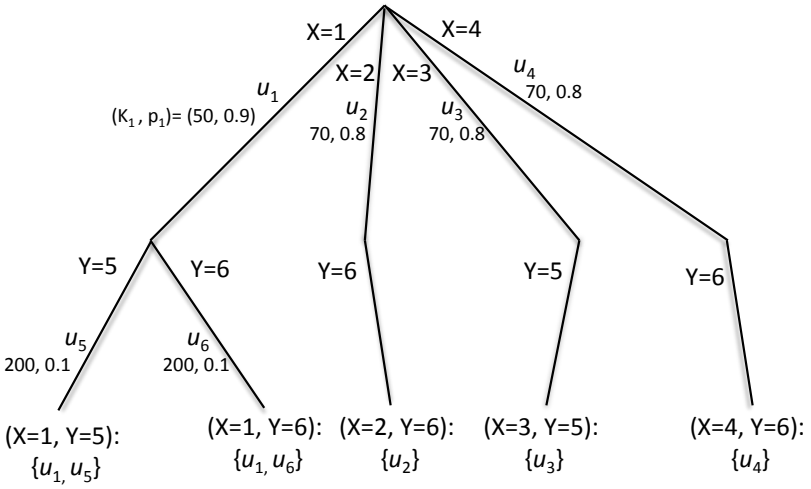


Fig. 1: Search tree for example, including the unknowns and their costs and probabilities. Leaf nodes correspond to potential solutions, which have an associated set of unknowns. For example, $(X = 1, Y = 5)$ is a solution if both unknowns u_1 and u_5 turn out to be 1, which has probability $0.9 \times 0.1 = 0.09$.

Algorithms with low expected cost will clearly need to consider the costs and the probabilities. A backtracking algorithm should ideally not always determine any unknown it meets, but allow the possibility to delay determining an unknown, to check whether it seems worthwhile doing so.

We define algorithms for this problem, based on backtracking search. Such algorithms can be crudely divided into three classes:

- Type 0: determining all unknowns to begin with;
- Type 1: determining unknowns as we meet them in the search;
- Type 2: making decisions about whether it's worth determining an unknown, making use of cost and/or probabilistic information.

The normal solving approaches for CSPs fall into Type 0, where the full CSP is elicited first and we then solve it, based on backtracking search with propagation at each node of the search tree. Algorithms for open constraints, which don't assume any cost or probability information, can be considered as being Type 1. In this paper we construct Type 2 algorithms, which make use of the cost and probabilistic information.

We first consider a simple iterative algorithm based on a limit on the costs of unknowns that may be determined; for each cost limit value, we perform a backtracking search; if this fails to find a solution we increment the cost limit, and search again. With this algorithm it can easily happen that we pay a cost of determining an unknown tuple, only to find that that particular branch fails

to lead to a solution for other reasons, as in the example, with unknown u_1. A natural idea is to delay determining an unknown, in order to find out if it is worth doing so. Our main algorithm, described in Section 4, usually will not immediately determine an unknown, but explore more deeply first. The experimental results in Section 5 strongly suggest that this can be worthwhile.

Related Work

The motivation for this work is related to part of that for Open Constraints [1–3, 6], and Interactive CSPs [4, 5], with a major difference being our assumption of there being cost and probabilistic information available ([2] considers costs in optimisation problems, but in a rather different way). Although these kinds of methods could be used for our problem, not taking costs and probabilities into account will, unsurprisingly, tend to generate solutions with poor expected cost, as illustrated by the example and our experimental results. Algorithms which interleave elicitation and solving for incomplete problems are also considered in [7–9], though in a soft constraints context. However, they do not assume the existence and use of probabilistic information regarding whether an unknown tuple satisfies a constraint; also they do not consider varying costs of eliciting a tuple. (There is also a less strongly related body of work on interactive solving e.g., [10], where a user sequentially assigns values to variables.)

Another approach is to ignore the probabilistic information, and look for complete assignments that will incur minimal cost to check if they are solutions. Weighted constraints methods e.g., [11] can be used to search for such assignments. If all the probabilities were equal to 1 then this would solve the problem. However, it may well turn out that all the lowest cost assignments also have relatively low probability. Consider the example with K_5 (the cost of determining u_5) changed to be 10 instead of 200. The assignment which then needs minimum cost to discover if it's a solution is $(X_1 = 1, X_2 = 5)$; this again leads to a suboptimal algorithm.[1] Alternatively, one could search for complete assignments which have highest probability of being a solution, as in Probabilistic CSPs [12]. Although this may perform satisfactorily if all the costs are equal, with varying costs it seems that the costs should be taken into account. Consider the example, but where p_5, the probability that $u_5 = 1$, is changed from 0.1 to 0.9. The assignment with greatest chance of being a solution is $(X_1 = 1, X_2 = 5)$; however the cost of finding this solution is 250, so checking out this potential solution first is far from optimal.

Structure of the paper

The next section describes the model and problem more formally, along with a dynamic programming algorithm which is optimal but very expensive. Section 3 derives a more practical solution approach, leading to our main algorithm,

[1] The expected cost of such an algorithm is greater than $K_5 + p_5 K_1 + (1 - p_5)E_2 + p_5(1 - p_1)E_2 = 10 + 0.1 \times 50 + 86.8(0.9 + 0.1 \times 0.1) = 93.988$.

which is described in Section 4. Section 5 describes the experimental testing and results. Section 6 shows how the framework and algorithms can be extended for non-Boolean-valued unknowns, and Section 7 discusses other extensions.

This paper extends work in [13].

2 A Formal Model for Interleaving Solving and Elicitation

Section 2.1 defines a formal framework (ECI-CSPs), for modelling situations where the elicitation and solving of a CSP can be interleaved. Section 2.2 defines what it means to solve an ECI-CSP, and Section 2.3 gives a dynamic programming algorithm which solves ECI-CSPs optimally.

2.1 Expected Cost-based Interactive CSPs (ECI-CSPs)

Standard CSPs: Let V be a set of variables, which are interpreted as decision variables, so that we have the ability to choose values for them. Each variable $X \in V$ has an associated domain $D(X)$. For any subset W of V, let $D(W)$ be the set of assignments to W, which can be written as $\prod_{X \in W} D(X)$. Associated with each (standard) constraint c over V, is a subset V_c of V, which is called its *scope*. Define a (standard) constraint c over V to be a function from $D(V_c)$ to $\{0, 1\}$. We will sometimes refer to a set of constraints C over V as a *Constraint Satisfaction Problem (CSP) over* V. Let S be an assignment to all the variables V. S is said to satisfy constraint c if $c(S') = 1$, where S' is S restricted to V_c. S is a solution of CSP C (or, S *satisfies* C) if it satisfies each constraint in C.

The Unknowns: As well as decision variables V, we consider a disjoint set of variables $\mathcal{U}$, which we call the set of *unknowns*. These are uncertain variables, and we have no control over them, so that we cannot choose their values. They are all Boolean variables (though see Section 6 for a generalisation to multi-valued variables), so for any unknown u, we have $D(u) = \{0, 1\}$. For a set $\mathcal{W}$ of unknowns we define $D(\mathcal{W})$ to be the set of assignments to $\mathcal{W}$ (which can be considered as the set of functions from $\mathcal{W}$ to $\{0, 1\}$). It is assumed that the unknowns $\mathcal{U}$ are probabilistically independent variables. Hence if α is an assignment to $\mathcal{U}$, then $\Pr(\alpha)$, the probability that α occurs, is equal to $\prod_{u\,:\,\alpha(u)=1} p_u \times \prod_{u\,:\,\alpha(u)=0}(1 - p_u)$, where, for $u \in \mathcal{U}$, p_u is the probability that $u = 1$.

We assume that, for any unknown $u \in \mathcal{U}$, we can determine (i.e., discover) the value of u, that is, whether $u = 1$ or $u = 0$. So we assume we have some procedure $Det(\cdot)$ that takes an unknown u as input and returns 1 or 0. We also assume that there is a certain cost $K_u \in [0, \infty)$ for executing this procedure on u, and that we have probabilistic information about the success of this procedure. In particular we assume that we know the probability p_u of success, i.e., the probability that $Det(u) = 1$.

Incomplete Constraints: An *incomplete constraint* c over $(V,\mathcal{U})$ has an associated subset V_c of V called its *scope*. c is a function from $D(V_c)$ to $\{0,1\} \cup \mathcal{U}$. (Note that we allow the same unknown to be associated with more than one tuple.) Hence, to any tuple $t \in D(V_c)$, c assigns 1, 0 or some unknown. c is intended as a partial representation of some standard constraint c^* over V_c. $c(t) = 1$ is interpreted as: *it is known that t satisfies the constraint c^**. Also, $c(t) = 0$ is interpreted as: *it is known that t doesn't satisfies the constraint c^**; otherwise, if $c(t) \in \mathcal{U}$, then it is unknown if t satisfies the constraint. We will sometimes refer to a set C of incomplete constraints over $(V,\mathcal{U})$ as an *incomplete CSP*. It is interpreted as the partial information we have of a CSP $C^* = \{c^* : c \in C\}$; C^* is also sometimes referred to as the *unknown CSP*.

ECI-CSPs: An Expected Cost-based Interactive CSP (ECI-CSP) is formally defined to be a tuple $\langle V, D, \mathcal{U}, K, p, C\rangle$, with the components being defined as follows: V is a set of variables, and D is a function specifying the domain $D(X)$ of each variable $X \in V$; $\mathcal{U}$ is a probabilistically independent set of unknowns, p is a function from $\mathcal{U}$ to $[0,1]$, where $p(u)$ (usually written p_u) is the probability that $u = 1$; K is a function from $\mathcal{U}$ to $[0,\infty)$, and C is a set of incomplete constraints over $(V,\mathcal{U})$.

Associated with an incomplete constraint c are two standard constraints with the same scope. The *known constraint* $\underline{c}$ is given by $\underline{c}(t) = 1$ if and only if $c(t) = 1$ (otherwise, $\underline{c}(t) = 0$). A tuple satisfies $\underline{c}$ if and only if it is known to satisfy c^*. The *potential constraint* $\overline{c}$ is given by $\overline{c}(t) = 0$ if and only if $c(t) = 0$ (otherwise, $\overline{c}(t) = 1$). A tuple satisfies $\overline{c}$ if it could potentially satisfy c^*. For a given set of incomplete constraints C, the *Known CSP* is the set of associated known constraints: $\underline{C} = \{\underline{c} : c \in C\}$, and the *Potential CSP* $\overline{C}$ is the set of associated potential constraints: $\{\overline{c} : c \in C\}$.

Suppose that $c(t) = u$, and we determine u and find out that $u = 1$. Then we now know that t does satisfy the constraint, so we can instantiate u to 1, i.e., replace $c(t) = u$ by $c(t) = 1$. Define $c[u = 1]$ to be the incomplete constraint generated from c by replacing every occurrence of u by 1. We define $c[u = 0]$ analogously. More generally, let ω be an assignment to a set $\mathcal{W} \subseteq \mathcal{U}$ of unknowns, and let c be an incomplete constraint. $c[\omega]$ is the incomplete constraint obtained by replacing each u in $\mathcal{W}$ by its value $\omega(u)$. We define $C[\omega]$ to be $\{c[\omega] : c \in C\}$. $C[\omega]$ is thus the incomplete CSP updated by the extra knowledge ω we have about unknowns.

We do not usually need to determine all unknowns to know that an incomplete CSP is solvable or not. We say that incomplete CSP C *is solved by assignment S (to variables V) in the context ω* if S is a solution of the associated known CSP $\underline{C[\omega]}$. In other words, if S is known to be a solution of C given ω. An incomplete CSP C is *insoluble in the context ω* if the associated potential CSP $\overline{C[\omega]}$ has no solution. In this case, even if all the other unknowns are found to be equal to 1, the CSP is still insoluble.

2.2 Policies for Solving ECI-CSPs

An algorithm for solving an incomplete CSP involves sequentially determining unknowns until we can find a solution. Of course, the choice of which unknown to determine next may well depend on whether previous unknowns have been determined successfully or not. In the example in Section 1, if we determine u_1 and discover that $u_1 = 0$, which implies that no solution includes the assignment $X = 1$, then there is no point in determining unknown u_5, since it only tells us whether $(X = 1, Y = 5)$ satisfies constraint c_2, which is now irrelevant, as we now know that $(X = 1, Y = 5)$ is not a solution.

What we call a *policy* for an ECI-CSP is a decision making procedure that sequentially chooses unknowns to determine. The choice of unknown to determine at any stage can depend on information received from determining unknowns previously. The sequence of decisions ends either with a solution to the known part of the CSP, or with a situation in which there is no solution, even if all the undecided unknown tuples are in their respective constraints. In more abstract terms a policy can be considered as follows:-

Given an assignment ω to some (possibly empty) set $\mathcal{W}$ of unknowns, a policy does one of the following:

(a) returns a solution of the Known CSP (given ω);
(b) returns "Insoluble" (it can only do this if the Potential CSP (given ω) is insoluble);
(c) choose another undetermined unknown.

In cases (a) and (b) the policy terminates at this point. In case (c), the chosen unknown u is determined, with value $b = 1$ or 0. ω is then extended with $u = b$, and another choice is made by the policy, given $\omega \cup [u = b]$. The sequence continues until the problem is solved or proved unsatisfiable.

We will define policies more formally as follows:

Let $\langle V, D, \mathcal{U}, K, p, C \rangle$ be an ECI-CSP. Let G be the set of all partial assignments to sets of unknowns, i.e., $G = \bigcup_{\mathcal{W} \subseteq \mathcal{U}} D(\mathcal{W})$. Let $H = D(V) \cup \mathcal{U} \cup \{\texttt{Insoluble}\}$. A policy π for this ECI-CSP is defined to be a function from G to H satisfying the following conditions:

(a) If $\pi(\omega) \in D(V)$ then $\pi(\omega)$ is a solution of the known CSP $\underline{C[\omega]}$.
(b) If $\pi(\omega) = \texttt{Insoluble}$ then the potential CSP $\overline{C[\omega]}$ is unsatisfiable.
(c) If $\pi(\omega) \in \mathcal{U}$ then $\pi(\omega) \notin \mathcal{W}$ (the next unknown chosen must not be one of the unknowns already chosen).

Policy π has an associated algorithm:

PROCEDURE $Algo_\pi$
  $\mathcal{W} := \emptyset$ and $\omega := \Diamond$ (the empty assignment);
  repeat
    if $\pi(\omega) \in D(V)$
      then return *"$\pi(\omega)$ is a solution of the CSP."* and **stop**;
      else if $\pi(\omega) = \texttt{Insoluble}$ **then** return *"CSP is insoluble."* and **stop**;

else if $\pi(\omega) \in \mathcal{U}$
then determine unknown $\pi(\omega)$ to give Boolean value b;
$\mathcal{W} := \mathcal{W} \cup \{\pi(\omega)\}$
$\omega := \omega \cup \{[\pi(\omega) := b]\}$
until $\mathcal{W} = \mathcal{U}$

We say that this algorithm *implements policy* π.

Define a *scenario* to be a complete assignment to all the unknowns, i.e., an element of $D(\mathcal{U})$. Let $\Pr(\alpha)$ be the probability of scenario α occurring. Since the variables $\mathcal{U}$ are probabilistically independent, we have $\Pr(\alpha) = \prod_{u\,:\,\alpha(u)=1} p_u \times \prod_{u\,:\,\alpha(u)=0}(1-p_u)$. A policy iteratively chooses unknowns to determine until it terminates. [2]

The Expected Cost of a policy, and optimal policies: The behaviour of the algorithm implementing policy π depends (only) on the true values of the unknowns, i.e., on the scenario. In other words, the cost incurred by a policy is a function only of the scenario. Let us write $\mathrm{Cost}_\pi(\alpha)$ for the cost incurred by policy π in scenario α. Since the scenario is a random variable, we can consider the expected cost $\mathrm{EC}(\pi)$ of policy π, which equals $\sum_{\alpha \in D(\mathcal{U})} \Pr(\alpha)\mathrm{Cost}_\pi(\alpha)$, where the summation is over all scenarios α.

Let $\mathcal{W}_\alpha$ be the set of unknowns determined by the algorithm implementing π, given scenario α (i.e., given that α represents the actual values of the unknowns). Thus $\mathcal{W}_\alpha$ is the value of $\mathcal{W}$ at the end of the algorithm. The cost $\mathrm{Cost}_\pi(\alpha)$ that the algorithm incurs in scenario α equals $\sum_{u \in \mathcal{W}_\alpha} K_u$, the sum of costs of the determined unknowns.

Evaluating Policies. We evaluate policies in terms of their expected cost. So, we aim to define algorithms that implement policies which have relatively low expected cost. A policy π is optimal if it has minimum expected cost, i.e., if $\mathrm{EC}(\pi) \leq \mathrm{EC}(\pi')$ for all policies π'.

2.3 Using Dynamic Programming to Generate an Optimal Policy

Although the problem involves minimising expected cost over all policies, the structure of the decisions—dynamically choosing a sequence from a (large) set of objects—does not fit very naturally into such formalisms as Influence Diagrams [15], Markov Decision Processes [16] and Stochastic Constraint Programming [17]. We describe below a simple dynamic programming [18] algorithm for generating an optimal policy.

Consider ECI-CSP $\mathcal{E} = \langle V, D, \mathcal{U}, K, p, C\rangle$. Let ω be an assignment to some set of unknowns $\mathcal{W} \subseteq \mathcal{U}$. Define ECI-CSP $\mathcal{E}_\omega$ to be the $\mathcal{E}$ updated with ω, i.e., $\langle V, D, \mathcal{U} - \mathcal{W}, K, p, C[\omega]\rangle$, where functions K and p are restricted to $\mathcal{U} - \mathcal{W}$.

[2] The effect of a policy in a particular scenario can be viewed in terms of Dynamic CSPs [14], since the sequence of Known CSPs generated involves constraints being incrementally relaxed (tuples being added) until there is a solution.

Define $A_{\mathcal{E}}(\omega)$ to be the minimal expected cost over all policies for solving $\mathcal{E}_\omega$. Then $A_{\mathcal{E}}(\omega) = 0$ if either the associated Known CSP $\underline{C[\omega]}$ is soluble or the associated Potential CSP $\overline{C[\omega]}$ is insoluble. Otherwise, any policy (in particular, an optimal policy) chooses some unknown $u \in \mathcal{U} - \mathcal{W}$ to determine, incurring cost K_u and with chance p_u of finding that $u = 1$. If $u = 1$ then we have the incomplete CSP $C(\omega \cup \{u = 1\})$ to solve, which has minimal expected cost $A_{\mathcal{E}}(\omega \cup \{u = 1\})$. This leads to the following equation for $A_{\mathcal{E}}$:

Proposition 1. *Let $\mathcal{E} = \langle V, D, \mathcal{U}, K, p, C\rangle$ be an ECI-CSP, let $\mathcal{W}$ be a subset of $\mathcal{U}$, and let $\omega \in D(\mathcal{W})$ be an assignment to $\mathcal{W}$. If $\underline{C[\omega]}$ is soluble or $\overline{C[\omega]}$ is insoluble then $A_{\mathcal{E}}(\omega) = 0$. Otherwise,*

$$A_{\mathcal{E}}(\omega) = \min_{u \in \mathcal{U}-\mathcal{W}} \big(K_u + p_u A_{\mathcal{E}}(\omega \cup \{u = 1\}) + (1 - p_u) A_{\mathcal{E}}(\omega \cup \{u = 0\})\big).$$

Proof: Throughout this proof we abbreviate $A_{\mathcal{E}}$ to just A.

If $\underline{C[\omega]}$ is soluble or $\overline{C[\omega]}$ is insoluble then there exists a policy which either produces a solution of $\underline{C[\omega]}$ or returns "Insoluble" if $\overline{C[\omega]}$ is insoluble, without determining any unknowns in $\mathcal{U} - \mathcal{W}$, and hence incurring no cost. Thus in these cases, $A(\omega) = 0$.

Now suppose otherwise that $\underline{C[\omega]}$ is not soluble and $\overline{C[\omega]}$ is soluble. Let $B = \min_{u \in \mathcal{U}-\mathcal{W}}(K_u + p_u A(\omega \cup \{u = 1\}) + (1 - p_u)A(\omega \cup \{u = 0\}))$. We will show that $A(\omega) \leq B$ and $A(\omega) \geq B$, proving the result.

Proving $A(\omega) \leq B$: Let u' be an element of $\mathcal{U} - \mathcal{W}$ that minimises $K_u + p_u A(\omega \cup \{u = 1\}) + (1 - p_u)A(\omega \cup \{u = 0\})$, so that $B = K_{u'} + p_{u'} A(\omega_1) + (1 - p_{u'})A(\omega_0)$, where ω_1 equals $\omega \cup \{u' = 1\}$, i.e., ω extended by the assignment $\{u' = 1\}$, and ω_0 equals $\omega \cup \{u' = 0\}$. Let π_1 be an optimal policy for $\mathcal{E}_{\omega_1}$, so that $\mathrm{EC}(\pi_1) = A(\omega_1)$, and let π_0 be an optimal policy for $\mathcal{E}_{\omega_0}$, so that $\mathrm{EC}(\pi_0) = A(\omega_0)$.

Consider the policy π for $\mathcal{E}_\omega$ given by determining unknown u' first (i.e., $\pi(\Diamond) = u'$) and then following π_1 if $u' = 1$, and following π_0 if $u' = 0$. Thus we define $\pi(\Diamond) = u'$, and $\pi(\omega') = \pi_1(\omega')$ for $\omega' \in D(\mathcal{U} - \mathcal{W})$ which include the assignment $u' = 1$, and $\pi(\omega') = \pi_0(\omega')$ for $\omega' \in D(\mathcal{U} - \mathcal{W})$ which include the assignment $u' = 0$, and define $\pi(\omega')$ arbitrarily for other $\omega' \in D(\mathcal{U} - \mathcal{W})$ (they're irrelevant).

$\mathrm{EC}(\pi) = K_{u'} + p_{u'}\mathrm{EC}(\pi_1) + (1 - p_{u'})\mathrm{EC}(\pi_0) = K_{u'} + p_{u'} A(\omega_1) + (1 - p_{u'})A(\omega_0) = B$. The minimality of $A(\omega)$ then implies that $A(\omega) \leq B$.

Proving $A(\omega) \geq B$: Consider an optimal policy π'' for $\mathcal{E}_\omega$, so that $\mathrm{EC}(\pi'') = A(\omega)$. Let u'' be the first unknown that π'' determines, i.e., $\pi''(\Diamond) = u''$. Let ω^1 be ω extended with the assignment $u'' = 1$, i.e., $\omega^1 = \omega \cup \{u = 1\}$, and let $\omega^0 = \omega \cup \{u = 0\}$.

Given $u'' = 1$, π'' defines a policy for $\mathcal{E}_{\omega^1}$. Thus the expected cost of π'' given $u'' = 1$ is at least $K_{u''} + A(\omega^1)$.

Similarly, the expected cost of π'' given $u'' = 0$ is at least $K_{u''} + A(\omega^0)$. Thus $\mathrm{EC}(\pi'') \geq p_{u''}(K_{u''} + A(\omega^1)) + (1 - p_{u''})(K_{u''} + A(\omega^0))$, which equals $K_{u''} + p_{u''} A(\omega^1) + (1 - p_{u''})A(\omega^0)$.

Therefore, $A(\omega) = \mathrm{EC}(\pi'') \geq K_{u''} + p_{u''}A(\omega \cup \{u'' = 1\}) + (1 - p_{u''})A(\omega \cup \{u'' := 0\})$, which proves that $A(\omega) \geq B$.

■

Algorithm for finding optimal solution of an ECI-CSP. The minimal expected cost over all policies for solving the original ECI-CSP is equal to $A[\Diamond]$, where $\Diamond$ is the assignment to the empty set of variables. We can thus find the minimal expected cost by using a simple dynamic programming algorithm, iteratively applying the above equation, starting with all scenarios (or from minimal assignments ω such that $\overline{C[\omega]}$ is insoluble); we can also find an optimal policy in this way, by recording, for each ω, a choice $u \in \mathcal{U} - \mathcal{W}$ which minimises the expression for $A(\omega)$. The correctness of the following algorithm for computing the minimum expected cost $A[\Diamond]$ follows immediately from Proposition 1.

PROCEDURE *Optimal-Expected-Cost*

for $i = 0, \ldots, |\mathcal{V}|$
  for all $\mathcal{W} \subseteq \mathcal{V}$ with $|\mathcal{W}| = |\mathcal{V}| - i$
    for all $\omega \in D(\mathcal{W})$
      if $\underline{C[\omega]}$ is soluble **or** $\overline{C[\omega]}$ is insoluble **then** $A(\omega) := 0$;
        else let u_ω be an element of $\mathcal{U} - \mathcal{W}$ with minimum value of
$$K_u + p_u A(\omega \cup \{u = 1\}) + (1 - p_u)A(\omega \cup \{u = 0\});$$
$$A(\omega) := K_{u_\omega} + p_{u_\omega} A(\omega \cup \{u_\omega\} = 1) + (1 - p_{u_\omega})A(\omega \cup \{u_\omega\} = 0);$$
      end for ;
    end for ;
  end for ;
Return $A(\Diamond)$.

A corresponding optimal policy π^* can be defined as follows:

— If the CSP $\underline{C[\omega]}$ is soluble then let $\pi^*(\omega)$ be any solution of it;
— if $\overline{C[\omega]}$ is insoluble then $\pi^*(\omega) =$ `Insoluble`;
— otherwise, define $\pi^*(\omega) = u_\omega$ (as generated in the above algorithm).

However, there are $3^{|\mathcal{U}|}$ different possible assignments ω (because ω is an assignment to some subset $\mathcal{W}$ of $\mathcal{U}$, so to fix ω we have three choices for each $u \in \mathcal{U}$, either $u \notin \mathcal{W}$ or $u = 1$ or $u = 0$). Therefore this algorithm uses space proportional to $3^{|\mathcal{U}|}$ and hence will only be feasible for problems with very small $|\mathcal{U}|$, i.e., very few unknowns (whereas problem instances in some of our experiments in Section 5 involve more than 2,000 unknowns). More generally, it seems that we will need to use heuristic algorithms.

3 Minimising Scaled Expected Cost

The hardness of the problem of generating an optimal policy means that it is natural to look for approximation algorithms. We will consider policies of the following form.

– *We choose a potential solution, i.e., a solution of the potential CSP, and check whether this is a solution. This will involve determining unknowns (and instantiating them); if all the unknowns are determined successfully, we have a solution, and we stop. Otherwise, one of the unknowns fails, i.e., is found to be* 0*; we choose and check another potential solution, continuing until either we have found a solution, or there are no remaining potential solutions (in which case the unknown CSP is unsatisfiable).*

There are two procedures that need to be defined for an algorithm implementing such a policy.

(1) Choosing the next potential solution to evaluate.
(2) Choosing the order of unknowns to be determined in checking if a potential solution is a solution (of the unknown, actual, CSP). We then determine unknowns in that order until either one is determined unsuccessfully, or they are all determined successfully, and we have a solution of the Known CSP, and hence of the (unknown) CSP, and can therefore stop.

Regarding (2), our approach is to minimise the expected cost incurred in checking if a potential solution is a solution. We describe this in more detail in Section 3.1.

For (1), we define the scaled expected cost of a potential solution to be the ratio R/P where R is the expected cost in checking if it's a solution, and P is the probability that the potential solution actually is a solution. We then choose potential solutions which have minimal scaled expected cost. This is described in Section 3.2. In fact, as we discuss in Section 3.3, it is convenient for computational purposes to simplify the algorithm slightly so that the solutions are chosen only approximately in increasing order of minimal scaled expected cost; this is the basis of the algorithm described in Section 4. Unsurprisingly, the algorithm is not optimal; we give examples to illustrate this in Section 3.4.

3.1 Evaluating a Complete Assignment

In this section we consider the problem of testing if a given complete assignment is a solution of an ECI-CSP $\langle V, D, \mathcal{U}, K, p, C\rangle$; the key issue is the order in which we determine the associated unknowns. This analysis is relevant for our main algorithm described in Section 4.

Associated with each potential solution S (i.e., solution of the associated potential CSP $\overline{C}$) is a set of unknowns, which can be written as: $\{c(S) : c \in C\} \cap \mathcal{U}$ (where $c(S)$ is an abbreviation for $c(S')$, and S' is the projection of S to the scope V_c of c). An unknown u is in this set if and only if there exists some constraint c such that $c(S) = u$. Label these unknowns as $U = \{u_1, \ldots, u_m\}$; we abbreviate p_{u_i} to p_i, and K_{u_i} to K_i. We also define $r_i = K_i/(1 - p_i)$, where we set $r_i = \infty$ if $p_i = 1$ (so that if $p_i = p_j = 1$ then $r_i = r_j$). Assignment S is a solution of the unknown CSP if and only if each of the unknown values in U is actually a 1. Since the unknowns are independent variables, the probability that S is a solution of the CSP is $p_1 p_2 \cdots p_m$, which we write as $P(U)$.

For example, suppose that a potential solution S has associated set of unknowns $\{u_1, u_2\}$. If we determine unknown u_1 first, then the expected cost of determining if the potential solution is a solution is $K_1+p_1K_2$, since we incur cost K_1, and only determine K_2 if it turns out that $Det(u_1) = 1$, which has probability p_1 of occurring. If we determine u_2 first the expected cost incurred is $K_2+p_2K_1$, since if $Det(u_1) = 0$, then we know that S is not a solution. The difference between these two is $K_1-p_2K_1-K_2+p_1K_2$ which equals $(1-p_2)K_1-(1-p_1)K_2$. So it is less expensive to determine u_2 first if $(1-p_2)K_1 > (1-p_1)K_2$, that is, if $K_1/(1-p_1) > K_2/(1-p_2)$, i.e., $r_1 > r_2$. The difference can be very significant; for example, if $K_1 = 100$ and $K_2 = 50$, and $p_1 = 0.5$ and $p_2 = 0.1$, then determining u_1 first leads to an expected cost of 125, whereas determining u_2 first leads to expected cost 60.

More generally, to evaluate the set of unknowns $\{u_1, \ldots, u_m\}$, we determine them in some order until either we find one which fails, i.e., until $Det(u_i) = 0$, or until we have determined them all. Associated with an unknown u_i is the cost K_i and success probability p_i. Suppose we evaluate the unknowns in the sequence $u_1, \ldots, u_m$. We start by determining u_1, incurring cost K_1. If u_1 is successfully determined (this event has chance p_1), we go on to determine u_2, incurring additional cost K_2, and so on. The expected cost in evaluating these unknowns in this order is therefore $K_1+p_1K_2+p_1p_2K_3+\cdots+p_1p_2\cdots p_{m-1}K_m$. Let R_π be the expected cost incurred in evaluating unknowns $\{u_1, \ldots, u_m\}$ in the order $\pi(1), \pi(2), \ldots, \pi(m)$, i.e., with $u_{\pi(1)}$ first, and then $u_{\pi(2)}$, etc. Expected cost R_π is therefore equal to $K_{\pi(1)} + p_{\pi(1)}K_{\pi(2)} + p_{\pi(1)}p_{\pi(2)}K_{\pi(3)} + \cdots + p_{\pi(1)}p_{\pi(2)}\cdots p_{\pi(m-1)}K_{\pi(m)}$.

The following result shows which is the order of determining unknowns that minimises the expected cost of evaluating the potential solution.

Proposition 2. *For a given set of unknowns $\{u_1, \ldots, u_m\}$, R_π is minimised by choosing π to order unknowns with smallest r_i $(= K_i/(1-p_i))$ first, i.e., setting ordering $\pi(1), \pi(2), \ldots, \pi(m)$ in any way such that $r_{\pi(1)} \leq r_{\pi(2)} \leq \cdots \leq r_{\pi(m)}$.*

Proof: [3] Consider any permutation σ representing ordering $\sigma(1), \sigma(2), \ldots, \sigma(m)$. To simplify notation we will write $p_{\sigma(i)}$ as q_i, and $K_{\sigma(i)}$ as L_i so that R_σ equals $L_1+q_1L_2+q_1q_2L_3+\cdots+q_1q_2\cdots q_{m-1}L_m$. We also write, $r_{\sigma(i)}$ as r'_i which equals $L_i/(1-q_i)$.

Let k be some number in $\{1, \ldots, m-1\}$. Let σ' be the permutation σ but with the k^{th} and $(k+1)^{th}$ elements in the sequence swapped. The expressions for R_σ and $R_{\sigma'}$ only differ on two terms. $R_\sigma - R_{\sigma'}$ can be seen to be equal to

$$(q_1\cdots q_{k-1})(L_k + q_kL_{k+1}) - (q_1\cdots q_{k-1})(L_{k+1} + q_{k+1}L_k),$$

which equals $(q_1\cdots q_{k-1})((1-q_{k+1})L_k - (1-q_k)L_{k+1})$.

We will show that $r'_k \geq r'_{k+1}$ implies $R_\sigma \geq R_{\sigma'}$, i.e., $(q_1\cdots q_{k-1})((1-q_{k+1})L_k - (1-q_k)L_{k+1}) \geq 0$. If $q_k = 1$ then the latter inequality clearly holds

[3] This optimisation problem can be seen to be a version of the Spanish Treasure problem (see e.g. [19]); however we include the proof for the sake of completeness.

so we have $R_\sigma \geq R_{\sigma'}$. So, we can assume that $q_k \neq 1$. Suppose that $r'_k \geq r'_{k+1}$, i.e., $L_k/(1-q_k) \geq L_{k+1}/(1-q_{k+1})$. Since $q_k \neq 1$ we also have $q_{k+1} \neq 1$ (or else $r'_{k+1} = \infty > r'_k$). Hence, $(1-q_{k+1})L_k - (1-q_k)L_{k+1}) \geq 0$ and so $R_\sigma \geq R_{\sigma'}$, as required.

The value of expected cost, R_π, is therefore not increased if we iteratively swap a u_i with smallest value of r_i with its preceding element in the sequence, until it's the first element in the sequence. Similarly, the expected cost is not increased if we iteratively swap the u_j with next smallest value of r_j to be the second element in the sequence, and so on, until we generate a permutation π which orders elements in increasing order of r_i, showing that $R_\pi \leq R_\sigma$, which proves the result. ■

For a set of unknowns $U = \{u_1, \ldots, u_m\}$ we define $R(U)$ to be R_π, where π is chosen so as the minimise the cost, by ordering the unknowns to have smallest r_i first, as shown by Proposition 2. (Proposition 2 also implies that it doesn't matter which way we order consecutive elements with equal value of r_i.)

3.2 Scaled expected cost for evaluating potential solutions

We need to define a procedure which chooses the potential solution that will be evaluated next. A natural way is to define a real-valued measure on potential solutions which scores them; we then choose a potential solution which optimises this measure.

One simple measure is the probability that the potential solution is a solution, which corresponds, because of independence of the unknowns, to the product of the probabilities of the unknowns. Specifically, if potential solution S has associated set of unknowns U_S, we choose S with maximal $P(U_S)$. An obvious weakness of this is that it doesn't take into account the costs, so can lead to very large costs being incurred unnecessarily.

Another simple measure is the sum of the costs of the unknowns, so that we choose a potential solution which has minimum associated sum of costs, i.e., minimal $\sum_{u \in U} K_u$. However, a clear weakness is that the probability of the potential solution being a solution is not taken into account.

The measure we focus on is what we call the scaled expected cost, which equals the expected cost divided by the probability that the potential solution is actually a solution:

Definition 1 (Scaled expected cost). *Let S be a solution of the potential CSP of an ECI-CSP $\langle V, D, \mathcal{U}, K, p, C \rangle$, and let $U = \{c(S) : c \in C\} \cap \mathcal{U}$ be the associated set of unknowns, which we again label as $\{u_1, \ldots, u_m\}$. Recall that $P(U) = p_1 p_2 \cdots p_m$, and $R(U)$ is the minimum expected cost of determining if S is a solution, i.e., of determining if all the unknowns in U have true value* 1. *The scaled expected cost of S is defined to be $R(U)/P(U)$.*

Expected utility interpretation of scaled expected cost: Imagine a situation where we are given an ECI-CSP, and a complete assignment S which is a possible

solution. We will consider an expected-utility-based analysis of whether it is worth determining these unknowns, to test if S is a solution, where cost is negative utility. Suppose the utility of finding that S is a solution is Q. The chance of finding that S is a solution is $P(U)$, so the expected reward is $P(U) \times Q$. (Note that the ordering of evaluating unknowns does not affect the expected reward.) If we determine all the unknowns U (based on the minimal cost order) then the expected cost is $R(U)$. Therefore the overall expected gain is $(P(U) \times Q) - R(U)$, so there is a positive expected gain if and only if $P(U) \times Q > R(U)$, i.e., if and only if $Q > R(U)/P(U)$. This is if and only if the utility of finding a solution is greater than the scaled expected cost. On this basis it seems natural to choose solutions that minimise scaled expected cost.

3.3 Approximating the scaled expected cost algorithm

Finding potential solutions in order of the solution measure is awkward computationally. To simplify the structure of the algorithms we make a slight approximation. We perform repeated complete searches over the space of solutions; with each search, we maintain an upper bound Q on the scaled expected cost, and we only consider potential solutions which have scaled expected at most equal to Q. For the next search we increment Q slightly, and so potential solutions are evaluated approximately in increasing order of scaled expected cost. This is the basis of our main algorithm, described in Section 4.

The monotonicity property, shown by the following proposition, is important since it allows the possibility of subtrees being pruned: if a partial assignment S has associated set of unknowns U, and we find that $R(U)/P(U)$ is more than our cost bound Q, then we can backtrack, since the set of unknowns U' associated with any complete assignment extending S will also have $R(U')/P(U') > Q$.

Proposition 3. *Let U and U' be any sets of unknowns with $U \subseteq U' \subseteq \mathcal{U}$. Then $R(U)/P(U) \leq R(U')/P(U')$.*

Proof: Since $\mathcal{U}$ is finite, it is sufficient to show the result for when U' contains a single extra unknown, say u, so that $U' = U \cup \{u\}$, since we can then repeatedly add extra unknowns one-by-one to prove the proposition. Let $r_u = K_u/(1-p_u)$. Write U as $\{u_1, \ldots, u_k\}$, where $r_i \leq r_j$ if $i \leq j$. By Proposition 2 we have $R(U) = K_1 + p_1 K_2 + p_1 p_2 K_3 + \cdots + p_1 p_2 \cdots p_{m-1} K_m$. Therefore, $R(U)/P(U) = \sum_{i=1}^{k} z_i$, where $z_i = K_i/(p_i \cdots p_k)$ for $i = 1, \ldots, k$. Let $j \in \{0, 1, \ldots, k\}$ be maximal such that $r_j \leq r_u$, where r_0 is defined to be 0. $R(U')/P(U')$ can be written as

$$\sum_{i=1}^{j} \frac{z_i}{p_u} + \frac{K_u}{p_u(p_{j+1} \cdots p_k)} + \sum_{i=j+1}^{k} z_i.$$

Hence $R(U')/P(U') - R(U)/P(U)$ equals the sum of non-negative terms

$$\frac{K_u}{p_u(p_{j+1} \cdots p_k)} + \sum_{i=1}^{j} z_i \left(\frac{1}{p_u} - 1\right),$$

showing that $R(U')/P(U') - R(U)/P(U) \geq 0$, and hence $R(U)/P(U) \leq R(U')/P(U')$. ■

3.4 Examples that illustrate non-optimality

Based on the analysis and discussion in Section 2.3, the problem of finding the optimal policy appears to be extremely hard, probably exponential in the number of unknowns. It is therefore not at all surprising that our relatively simple approach does not always find an optimal policy. We give two examples to illustrate this; these relate to the two required procedures described at the beginning of Section 3. The first shows that it is not always best to determine unknowns associated with a potential solution in the order described in Section 3.1. The second shows that it is not always best to choose potential solutions with minimal scaled expected cost.

Consider a problem with two variables, X and Y, both with domain $\{1, 2\}$ (see Table 2). The first constraint c_1 is a unary constraint on X, with $c_1(X = 1) = u_1$, and $c_1(X = 2) = u_2$, the second constraint c_2 is a unary constraint on Y with $c_2(Y = 1) = u_3$, and $c_2(Y = 2) = 0$. We have probabilities $p_1 = p_2 = p_3 = 0.5$, and costs $K_1 = 100$ and $K_2 = K_3 = 101$. There are two potential solutions, ($X = 1$, $Y = 1$), with associated set of unknowns $\{u_1, u_3\}$, and ($X = 2$, $Y = 1$), with associated set of unknowns $\{u_2, u_3\}$.

Table 2: Unary constraints c_1 and c_2, along with the costs (K_i) and probabilities (p_i) of their associated unknowns.

	$X = 1$	$X = 2$			$Y = 1$	$Y = 2$
$c_1(X)$	u_1	u_2		$c_2(X)$	u_3	0
(K_i, p_i)	$(100, 0.5)$	$(101, 0.5)$		(K_3, p_3)	$(101, 0.5)$	

The optimal policy involves determining u_3 first, and then, if this is successful, determining u_1, and then u_2, if $Det(u_1) = 0$. The expected cost of this policy is $K_3 + 0.5 \times (K_1 + 0.5K_2)$, which equals 176.25.

Compare this with the minimal scaled expected cost described above in Section 3.2. This also involves determining first whether ($X = 1$, $Y = 1$) is a solution, but u_1 would be determined first, since this choice minimises the expected cost of determining whether ($X = 1$, $Y = 1$) is a solution. However, this doesn't take into account that determining u_3 is valuable also for the other potential solution. This algorithm has expected cost $K_1 + 0.5K_3 + 0.5 \times (K_3 + 0.5K_2)$, which equals 226.25.

Now consider an example where there is an additional potential solution, ($X = 1$, $Y = 2$), with associated set of unknowns $\{u_4, u_5\}$, with $p_4 = p_5 = 0.5$

and $K_4 = K_5 = 100$. (For example, replace c_1 by a binary constraint with $c_1(X = 1, Y = 1) = u_1$, $c_1(X = 1, Y = 2) = u_4$, $c_1(X = 2, Y = 1) = u_2$ and $c_1(X = 2, Y = 2) = 0$, and change c_2 to the unary constraint with $c_2(Y = 1) = u_3$ and $c_2(Y = 2) = u_5$.)

The optimal policy will again determine u_3 first, and then, if this is successful, determine u_1; it thus checks first if $(X = 1, Y = 1)$ is a solution. However, our algorithm in Section 3.2 is sub-optimal since it will check first if $(X = 1, Y = 2)$ is a solution, because this solution has minimal scaled expected cost.

4 Iterative Expected Cost-bound Algorithm

In this section we define our main algorithm for solving a given Expected Cost-based CSP $\langle V, D, \mathcal{U}, K, p, C\rangle$. The key idea behind this algorithm is to allow the possibility of delaying determining an unknown associated with a constraint check, until it has explored further down the search tree; this is in order to see if it is worth paying the cost of determining that unknown. The algorithm performs a series of depth-first searches; each search is generated by the procedure *TreeSearch*. The structure of each search is very similar to that of a standard backtracking CSP algorithm.

The behaviour in each search (i.e., in each call of *TreeSearch*) depends on the value of a global variable Q, which is involved in a backtracking condition, and is increased with each tree search. For example, in the experiments described in Section 5, we set $Q_{initial} = 20$ and define $Next(Q)$ to be $Q \times 1.5$, so that the first search has Q set to 20, the second search has $Q = 30$, and then $Q = 45$, and so on. The value of Q can be roughly interpreted as the cost that the algorithm is currently prepared to incur to solve the problem (see the analysis in Section 3.2).[4]

The only differences between one tree search and the next are (i) the value of Q has changed to allow the search to go deeper at particular points; (ii) the previous tree search will (usually) have determined some unknowns (which, of course, won't need to be determined again); this may also allow the search to go deeper (i.e. backtrack less easily).

The procedure *TopLevel* first initialises the cost incurred (`GlobalCost`) to zero. It then performs repeated tree searches until a solution is found (see procedure *ProcessNode*$(\cdot)$ below) or until all relevant unknowns have been determined; more precisely: until the condition $R(\texttt{Unknowns}_N)/P(\texttt{Unknowns}_N) > Q$, which appears in procedures *ProcessNode*(N) and *ProcessLeafNode*(N), is not satisfied throughout the whole tree search. The algorithm will then determine all unknowns associated with leaf nodes, i.e., those that could be associated with a solution. An alternative would be to exit the loop when Q reaches a certain size

[4] Our experimental results for the main algorithm (without the size limit modification) tally very well with this interpretation, with the average Q for the last iteration being close to the average overall cost incurred (within 25% of the average cost for each of the four distributions used).

(or after a given number of values of Q have been tried), and then have a final tree search with Q set to ∞.

PROCEDURE *TopLevel*

```
GlobalCost := 0;    Q := Q_initial
repeat
    TreeSearch
    Q := Next(Q)
    until all relevant unknowns have been determined
TreeSearch
```

PROCEDURE *TreeSearch*

```
Unknowns_root := ∅;
Construct child N of root node
ProcessNode(N)
```

The core part of the algorithm is the procedure *ProcessNode*(N). Let N be the current node, and let $Pa(N)$ be its parent, i.e., the node above it in the search tree. Associated with node N is the set $\texttt{Unknowns}_N$ of *current unknowns*, which are unknowns which need to be successfully determined for the current partial assignment to form part of a solution (see below). At a node, if any of the current unknowns evaluates to 0 then there is no solution below this node. Conversely, if all of the current unknowns at a node evaluate to 1 then the current partial assignment is consistent with all constraints that have been checked so far. If all the current unknowns at a leaf node evaluate to 1 then the current assignment is a solution.

At a search node, we perform, as usual, a constraint check for each constraint c whose scope V_c has just been fully instantiated (i.e., such that (i) the last variable instantiated is in the scope, and (ii) the set of variables instantiated contains the scope). A constraint check returns either 1, 0 or some unknown. The algorithm first determines if any constraint check fails, i.e., if it returns 0. If so, we backtrack to the parent node, in the usual way, assigning an untried value of the associated variable, when possible, and otherwise backtracking to *its* parent node. Propagation (based on the constraints in the Potential CSP) can be used in the usual way to eliminate elements of a domain which cannot be part of any solution extending the current assignment.

The set $\texttt{DirectUnknowns}_N$, of unknowns directly associated with the node N, is defined to be the set of unknowns which are generated by the constraint checks at the node. The set of *current unknowns* at the node, $\texttt{Unknowns}_N$, is then initialised to be the union of $\texttt{DirectUnknowns}_N$ and the current unknowns of the parent node.

The algorithm then tests to see if it is worth continuing, or if it is expected to be too expensive to be worth determining the current set of unknowns. The backtracking condition is based on the analysis in Section 3.2. We view Q as representing (our current estimate of) the value of finding a solution. Then the expected gain, if we determine all the unknowns in $\texttt{Unknowns}_N$, is $P(\texttt{Unknowns}_N) \times Q$ where $P(\texttt{Unknowns}_N)$ is the chance that all the current unknowns evaluate to 1.

The expected cost of determining these unknowns sequentially is $R(\texttt{Unknowns}_N)$, as defined in Section 3.1, since we evaluate unknowns with smallest r_i first. So, determining unknowns $\texttt{Unknowns}_N$ is not worthwhile if the expected gain is less than the expected cost: $P(\texttt{Unknowns}_N) \times Q < R(\texttt{Unknowns}_N)$. Therefore we backtrack if $R(\texttt{Unknowns}_N)/P(\texttt{Unknowns}_N) > Q$.

We then construct a child node in the usual way, by choosing the next variable Y to instantiate, choosing a value y of the variable, and extending the current assignment with $Y = y$. If Y is the last variable to be instantiated then we use the *ProcessLeafNode*$(\cdot)$ procedure on the new node; otherwise we use the *ProcessNode*$(\cdot)$ procedure on the new node.

The *ProcessLeafNode*$(\cdot)$ procedure is similar to *ProcessNode*$(\cdot)$, except that we can no longer delay determining unknowns, so we determine each current unknown until we fail, or until all have been determined successfully. We determine an unknown with smallest $r_i = K_i/(1 - p_i)$ first (based on Proposition 2). If an unknown u_i is determined unsuccessfully, then there is no solution beneath this node. In fact, if N' is the furthest ancestor node of N which u_i is directly associated with (i.e. such that $\texttt{DirectUnknowns}_{N'} \ni u_i$), then there is no solution beneath N'. Therefore, we jump back (in the tree search) to N' and backtrack to its parent node $Pa(N')$. If all the unknowns $\texttt{Unknowns}_N$ have been successfully determined then the current assignment, which assigns a value to all the variables V, has been shown to be a solution of the CSP, so the algorithm has succeeded, and we terminate the algorithm.

PROCEDURE *ProcessNode*(N)

if any constraint check returns 0 **then** backtrack
$\texttt{Unknowns}_N := \texttt{Unknowns}_{Pa(N)} \cup \texttt{DirectUnknowns}_N$
if $R(\texttt{Unknowns}_N)/P(\texttt{Unknowns}_N) > Q$
    then backtrack to parent node
Construct (new) child node N' of N
if N' is a leaf node (all variables are instantiated)
    then *ProcessLeafNode*(N') **else** *ProcessNode*(N')

PROCEDURE *ProcessLeafNode*(N)

if any constraint check returns 0 **then** backtrack
$\texttt{Unknowns}_N := \texttt{Unknowns}_{Pa(N)} \cup \texttt{DirectUnknowns}_N$
if $R(\texttt{Unknowns}_N)/P(\texttt{Unknowns}_N) > Q$ **then** backtrack to parent node
while Unknowns non-empty **do**:
    Let u_i be unknown in $\texttt{Unknowns}_N$ with minimal r_i
    Determine u_i;
    $\texttt{GlobalCost} := \texttt{GlobalCost} + K_i$
    if u_i determined unsuccessfully
        then exit procedure, jumping back to furthest
            ancestor node associated with u_i
            and backtrack to its parent node
    $\texttt{Unknowns}_N := \texttt{Unknowns}_N - \{u_i\}$
    end (**while**)

Return current (complete) assignment as a solution and **stop**

If we apply this algorithm to the example in Section 1 (again using variable ordering X, Y, and numerical value ordering), no unknown will be determined until we reach an iteration where Q is set to at least 87.5. Then the first leaf node N that the algorithm will reach is that associated with the assignment ($X = 1,\ Y = 5$). $\texttt{Unknowns}_N$ is equal to $\{u_1, u_5\}$. $r_1 = K_1/(1 - p_1) = 50/0.1 = 500 > r_5 = 200/0.9$, so $R(\{u_1, u_5\}) = K_5 + p_5 K_1 = 205$, and $R(\{u_1, u_5\})/P(\{u_1, u_5\}) = 205/(0.9 \times 0.1) \approx 2278$, so the algorithm will backtrack; similarly for the leaf node associated with assignment ($X = 1,\ Y = 6$). The leaf node corresponding to ($X = 2,\ Y = 6$) has current set of unknowns $\{u_2\}$. Since $R(\{u_2\})/P(\{u_2\}) = 70/0.8 = 87.5 \leq Q$, unknown u_2 will be determined. If it evaluates to 1 then ($X = 2,\ Y = 6$) is a solution, and the algorithm terminates. Otherwise, u_3 and u_4 will be next to be determined. In fact, for this example, the algorithm generates an optimal policy, with expected cost of around 90.

One approach to improving the search efficiency of the algorithm is to set a limit `SizeLimit` on the size of the current unknown set $\texttt{Unknowns}_N$ associated with a node. If $|\texttt{Unknowns}_N|$ becomes larger than `SizeLimit` then we repeatedly determine unknowns, in increasing order of r_i, and remove the unknown from the current set until $|\texttt{Unknowns}_N| = \texttt{SizeLimit}$. It is natural then to change the backtracking condition to take this into account. In particular, we can change the test to be $(\texttt{CostDetSucc}_N + R(\texttt{Unknowns}_N))/P(\texttt{Unknowns}_N) > Q$, where $\texttt{CostDetSucc}_N$ is the cost incurred in (successfully) determining unknowns in ancestors of the current node, which can be considered as the cost that has already been spent in consistency checking of the current assignment.[5]

In the algorithm whenever we determine an unknown in $\texttt{Unknowns}_N$ we choose an unknown u_i with minimum r_i. This is in order to minimise the expected cost of determining the set of current unknowns, because of Proposition 2. Alternatively, one could bias the ordering towards determining more informative unknowns (cf. Section 3.4). For example, suppose $\texttt{Unknowns}_N$ includes two unknowns u_i and u_j, where u_i is associated with a unary constraint, and u_j is associated with a constraint of larger arity. Even if r_i is slightly more than r_j, it may sometimes be better to determine u_i before u_j since u_i may well be directly associated with many other nodes in the search tree.

5 Experimental Testing

In this section we provide an empirical evaluation of a number of algorithms for handling ECI-CSPs, including cost-based algorithms, probability-based algorithms, and variations of our algorithm which considers both the cost and probability of unknowns. We test these algorithms on two types of ECI-CSP based on random binary CSPs and k-colouring CSPs respectively. We show that

[5] An alternative choice would be to not include this amount, since this cost has already been incurred, so can't be undone. However, this will tend to cause the algorithm to determine more unknowns when `SizeLimit` is small.

ignoring either of the attributes of the unknowns proves detrimental to the objective of minimising the cost occurred in finding a solution to an ECI-CSP.

5.1 Problem Types

We generated two types of ECI-CSP by altering the following basic problems:

1. Random binary CSPs
2. Incomplete k-colouring CSPs

For the first type, we generate a random binary CSP in accordance with model B generation [20] with parameters $\langle n, d, m, t \rangle$, where:

- n is the number of variables
- d is the uniform domain size
- m is the graph density, i.e. the number of constraints per problem as a fraction of the total possible number of binary constraints over the n variables
- t is the constraint tightness, i.e. the number of tuples not allowed by each constraint as a fraction of the total possible number of tuples

For each constraint in the CSP we randomly select s allowed tuples and s disallowed tuples to be assigned unknown, where the value s is set to the floor of (i.e., the greatest integer less than) two thirds of the minimum of (#allowed tuples, #disallowed tuples). Each of these $2s$ tuples is assigned a different unknown u_i, with an associated probability p_{u_i} chosen independently from a uniform distribution taking values between 0 and 1, and an associated cost K_{u_i} generated from a distribution as described below. Each u_i is allocated its true value (which the algorithms only have access to when they determine u_i): this is assigned 1 with probability p_i, otherwise it is assigned 0 (where $u_i = 1$ means that the associated tuple satisfies the constraint).

We tested our algorithms on problems with varying cost distribution, and on problems with varying density (described later). We use four different distributions for cost, where costs are integers in our experiments. Each distribution has minimum value 1 and has median around 50. For $k = 1, 2, 3, 4$ using the k^{th} distribution, each cost K_{u_i} is an independent sample of the random variable: $50 \times (2 \times \mathbf{rand})^k$ rounded up to be an integer, where **rand** is a random number taking values between 0 and 1 with a uniform distribution. Therefore $k = 1$ has a linear distribution, $k = 2$ is a scaled (and truncated) square root distribution, and so on.

The second type of ECI-CSP is generated similarly. A k-colouring CSP is generated with parameters $\langle n, d, m \rangle$, where the number of colours is equal to d, the domain size. The number, s, of allowed and disallowed tuples to be assigned unknown, is set to the floor of two thirds of the domain size (which equals the number of disallowed tuples in the inequality constraint). For these problems, however, we restricted the set of allowed tuples for possible conversion to unknowns, to be of the form (i, j) where $|i - j| = 1$, so that s of these (initially allowed) $2d - 2$ tuples are chosen as unknowns, as are s of the d (initially disallowed) tuples of the form (i, i). (This means that the constraints generated are,

in a sense, still close to being inequality constraints.) Each of these $2s$ tuples is assigned an unknown u_i in an identical manner to that described for the random binary CSPs.

The parameters for problem sets with varying cost distributions were mainly chosen so that each instance was likely to have many solutions. Each problem set contained 100 problems. All problems were generated so as to be insoluble without determining at least one unknown. Furthermore, for both types of problem we began with an initial spanning tree so that there were no unconnected components.

5.2 Algorithms

The following algorithms were tested (where, according to the terminology in Section 1, the first is Type 1, and the others are Type 2):

Basic Algorithm: The basic algorithm works like a normal CSP depth-first search algorithm (maintaining arc consistency on the potential CSP) except that it determines each unknown as soon as it is encountered. As usual, a constraint check is performed as soon as all the variables in the constraint's scope are instantiated. When a constraint check returns an unknown u_i, we immediately determine u_i, incurring cost K_i. If u_i is determined successfully, i.e., u_i is found to be 1, then the constraint check is successful.

Basic Iterative (Cost-Bound Algorithm): This algorithm performs iterative searches, parameterised by increasing cost bound q; each search is similar to the basic algorithm, except that all unknowns with cost greater than q are removed from search for the current iteration (that is, they are set to 0, which allows for improved pruning through propagation). If a solution is found, search terminates; otherwise, the search is complete, after which search restarts with q incremented by a constant, q_{inc}. The process continues until either a solution has been found or all unknowns have been determined and the algorithm has proven insolubility. For the experiments reported below, q starts off with value 0, and q_{inc} is 5.

Iterative Expected Cost-Bound Algorithm (ECB): This is the main algorithm described in Section 4. For this and the next two algorithms, $Q_{initial}$ is set to 20, with a multiplicative increment of 1.5.

ECB with Size Limit Algorithm (ECB-SL): This is the adapted version of the last algorithm discussed at the end of Section 4, which incorporates a limit `SizeLimit` on the cardinality of the set `Unknowns` of current unknowns, so that if $|\texttt{Unknowns}| > \texttt{SizeLimit}$ then elements of `Unknowns` are determined until either one is found to be 0 or $|\texttt{Unknowns}| = \texttt{SizeLimit}$. In the experiments reported below, we set `SizeLimit` to 5.

ECB with Individual Cost Limit Algorithm (ECB-CL): This algorithm modifies the main Expected Cost-Bound algorithm by having, for each iteration, a cost limit q on the unknowns to be considered, in order to improve the search efficiency (because of additional propagation), whilst maintaining cost effectiveness. It therefore is a kind of hybrid of the ECB algorithm and the basic iterative cost-bound algorithm described above. Let `maxK` be the maximum cost of any unknown in the current problem instance. On the first search all unknowns

with cost greater than 30%(maxK) are removed from search, so that q starts at 30%(maxK); on each iteration this bound q is incremented by $q_{inc} = 5\%$(maxK).

Cost Only Algorithm: This algorithm is based on the first simple measure given in Section 3.2. It works similarly to the ECB algorithm, but only uses information regarding the cost of the unknowns. Unknowns are determined at leaf nodes in order of increasing cost. The condition which causes backtracking for the set Unknowns of current unknowns is

$$\sum_{u \in \texttt{Unknowns}} K_u \geq L,$$

where L is a limit with initial value 0 which is incremented by 5 with each iteration.

Probability Only Algorithm: This complement of the previous algorithm is based on the second simple measure given in Section 3.2. It only uses information regarding the probability of the unknowns. Unknowns are determined at leaf nodes in order of increasing probability, (this minimises the expected number of unknowns determined before encountering an unknown with value 0). The condition which causes backtracking for the set Unknowns of current unknowns is

$$\prod_{u \in \texttt{Unknowns}} p_u \leq L,$$

where L is a limit with initial value 1 which is multiplied by 0.95 with each iteration.

The algorithms which delay determining an unknown, i.e. the non-"basic" algorithms, all incorporate backjumping. When an unknown is added to the list of current unknowns, the current level of search is stored with the unknown. If an unknown is determined and found to be 0, search backjumps to the level of search at which the unknown was added.

All algorithms used the same variable ordering heuristic: for the random binary problems the heuristic used was min domain; for the colouring problems we used the Brelaz heuristic [21], which chooses the variable with smallest domain and breaks ties by choosing the variable with the most uninstantiated neighbors.

We also implemented a cost-based value ordering heuristic. The heuristic chooses the value that has minimum total cost over the constraints between the variable and its instantiated neighbours, i.e. it minimizes the maximum cost that would immediately be spent determining unknowns if that value were chosen. The improvements for the iterative algorithms were minimal (indeed it can only provide an improvement for the final iteration) so we only present the results for the basic algorithm with the value ordering (*Basic Val*).

5.3 Results

We compare the performance of the different approaches in terms of cost incurred in solving a problem. In order to provide further insight into the behaviour of the algorithms, we also present results on the number of unknowns determined

in finding a solution by the algorithms, and, in some cases, number of search nodes explored. All results are averaged over 100 problems.

Random binary ECI-CSPs:

Varying Cost Distribution: The parameters for theproblems were $\langle 20, 10, 0.163, 0.4\rangle$ for which we generated four problem sets using four different cost distributions ($k = 1 \ldots 4$). Each problem had over 2,400 unknowns. The results are given in Table 3.

ECB performs best in terms of average cost: *Basic Iter* incurs between 7 and 11 times more cost for these instances, and *Basic* has average cost one or two orders of magnitude worse than *ECB*. (Naturally, all the algorithms do much better than determining all the unknowns prior to search, at a cost of more than 100,000.)

Table 3: Results For Different Cost Distributions - Random Binary

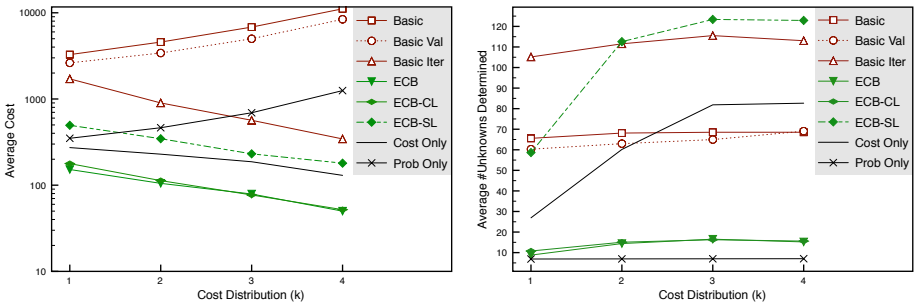
	Basic	Basic Val	Basic Iter	ECB	ECB-CL	ECB-SL	Cost Only	Prob Only
Linear ($k = 1$)	3272	2625	1711	**152**	178	495	273	350
Square ($k = 2$)	4574	3414	900	**105**	113	346	229	464
Cube ($k = 3$)	6823	4997	566	79	**77**	231	187	692
Fourth ($k = 4$)	11123	8401	344	**50**	52	180	130	1251
($k = 1$) Search Nodes	57	**55**	652	2.1×10^6	1.2×10^5	1.2×10^6	4.5×10^6	6.0×10^4
($k = 4$) Search Nodes	**59**	61	537	1.8×10^6	4.8×10^5	5.5×10^5	3.4×10^4	4.8×10^4

Notes: Problem parameters $\langle 20, 10, 0.163, 0.4\rangle$.
First 4 rows are average costs over 100 problems
Bottom two rows gives mean search nodes for $k = 1$ and $k = 4$

Unsurprisingly, *ECB* is vastly slower than the basic algorithms, generating on average around two million total search tree nodes for each problem instance, whereas *Basic Iter* generates only a few hundred. The two variants of the *ECB* algorithm both aim to improve the efficiency somewhat. *ECB-SL* cuts search tree nodes by more than 50% compared to *ECB*, but incurs roughly three or more times as much average cost. *ECB-CL* trades off cost and search efficiency much more effectively for these instances, with only slightly worse average costs, but generating only a fraction of the search tree nodes, less than 6% for the linear distribution, and less than 30% for the other distributions.

The poor performance of the basic algorithms and *ECB-SL* illustrates the benefits of only determining unknowns at leaf nodes. A similar result was found by Gelain et al. for incomplete soft CSPs [7–9], although in their work unknowns do not have associated costs and probabilities. Furthermore, their method of determining an unknown involved querying a user to specify the value of the worst/best unknown over the set of current unknowns, which does not fit with our framework.

The average number of unknowns determined by the different algorithms is given in Figure 2, along with a graphical representation of the results of Table 1. Trends in the average cost can be more clearly seen here, the average cost decreases as the cost distribution becomes more skewed for the algorithms that iteratively increase a cost limit on search. The opposite occurs for the three algorithms without a cost limit (i.e. *Basic*, *Basic Val* and *Probability Only*), where the average cost has a positive correlation with the cost distribution.



(a) Average Cost (b) Unknowns Determined

Fig. 2: Random binary problems $\langle 20, 10, 0.163, 0.4 \rangle$, different cost distributions.

The average number of unknowns determined by the different algorithms shows much less variance with regard to the cost distribution. The algorithms *ECB-SL* and *Cost Only* have the most obvious correlation between average unknowns determined and cost distribution. As expected, *Probability Only* determines fewest unknowns, however both *ECB* and *ECB-CL* determine only slightly more unknowns on average. Interestingly both *ECB-SL* and *Basic Iter* determine nearly twice as many unknowns as *Basic* for the non-linear cost distributions, despite incurring significantly less cost (order of magnitude less for $k = 3$ and 4).

The results in Table 3 showed that *ECB* incurred on average 50% less cost than *Cost Only*, we see in Figure 2b that ECB determined at most half as many unknowns on average per problem set. Clearly, the lack of probability information resulted in *Cost Only* determining many unknowns which had *true* value 0. On the other hand, *Probability Only* determined slightly fewer unknowns than ECB

but the lack of cost information resulted in it incurring a cost 2 to 25 times worse than *ECB*, depending on the cost distribution.

Varying Density: The previous results concerned problems where the complete problem had many solutions. We now look at the the performance of the different algorithms as problems approach the phase transition. This was achieved by fixing n, d and t for random binary problems and varying the density. Problems were, again, generated so as to be insoluble without determining at least one unknown, and all problems were soluble when all unknowns were determined.

For simplicity, the method of generation was slightly altered. The CSPs, prior to conversion to ECI-CSPs, were generated so as to have solutions as before. However as one approaches the phase transition, the likelihood of there remaining a solution after altering the set of allowed and disallowed tuples is greatly reduced. Thus we generated unknowns in the same manner as previously, except we randomly selected an allowed tuple when an unknown was assigned a true value of 1, and a disallowed tuple when an unknown was assigned a true value of 0. It can be shown that this does not bias the probabilities.

We generated several sets of random binary ECI-CSPs with 15 variables, domain size 5, tightness 0.4, while the density ranged from 0.15 to 0.5 in increments of 0.05. Each set contained 100 problems, the linear cost distribution was used for all problem sets. The number of unknowns per problem ranged from 336 to 720. We present the results in Table 4.

ECB once again performs best, with *ECB-CL* performing as well for the denser problems. Somewhat surprisingly, *Probability Only* performed better than all algorithms except *ECB* for the sparser problem sets; indeed the two algorithms, *ECB-CL* and *Cost Only*, had been consistently better than *Probability Only* in our previous experiments.

A possible explaination is that both algorithms remove unknowns with cost greater than a given limit from the search, since these smaller problems have fewer unknowns, there may be an unknown with a relatively large cost that is necessary to find a solution, which these two algorithms ignore until they have determined a number of unknowns with small cost. However, when we skewed the cost distribution for the set with density 0.15, the average cost incurred by *Probability Only* was significantly worse than *ECB-CL*, *ECB-SL* and *Cost Only*.

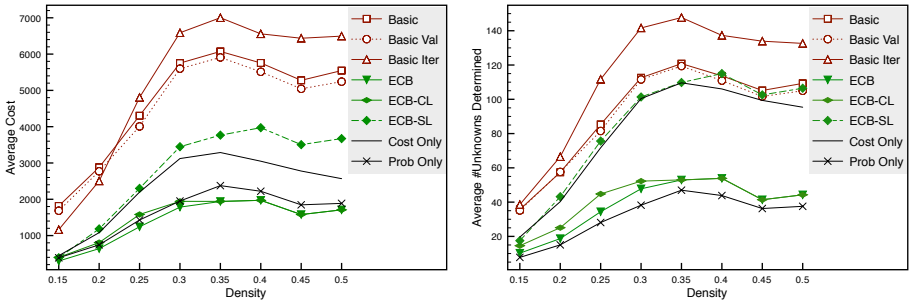
Figure 3 presents graphs for average cost and average number of unknowns determined. As the density increases, the average number of unknowns determined by the algorithms that don't take probabilities into account, or determine unknowns at internal nodes, shows a sharp rise. On the other hand, the increase in the average number of unknowns determined by *ECB*, *ECB-CL* and *Probability Only* is much more gradual.

***k*-colouring ECI-CSPs with varying cost distributions:** The parameters for the incomplete colouring problems were $\langle 30, 5, 0.227 \rangle$, for which we generated four problem sets using the same cost distributions as previously. Each problem had over 700 unknowns. The results are given in Table 5.

Table 4: Results For Different Density - Random Binary

	Basic	Basic Val	Basic Iter	ECB	ECB-CL	ECB-SL	Cost Only	Prob Only
0.15	1815	1685	1163	**299**	400	389	445	387
0.2	2880	2771	2504	**640**	807	1184	1081	745
0.25	4303	4007	4806	**1242**	1581	2300	2198	1429
0.3	5752	5599	6588	**1784**	1943	3448	3122	1952
0.35	6073	5908	7002	**1939**	1942	3767	3288	2376
0.4	5756	5513	6558	**1971**	**1971**	3972	3049	2221
0.45	5277	5044	6437	**1579**	**1579**	3505	2778	1847
0.5	5546	5241	6495	**1709**	**1709**	3672	2568	1887

Notes: Problem parameters $\langle 15, 5, m, 0.4 \rangle$.
Linear Cost Distribution
Costs are Averaged Over 100 Problems



(a) Average Cost (b) Unknowns Determined

Fig. 3: $\langle 15, 5, m, 0.4 \rangle$ problems, varying density.

Table 5: Results For Different Cost Distributions - Coloring

	Basic	Basic Val	Basic Iter	ECB	ECB-CL	ECB-SL	Cost Only	Prob Only
Linear ($k = 1$)	3520	3179	827	129	**127**	223	214	231
Square ($k = 2$)	4455	3876	315	**62**	68	122	104	315
Cube ($k = 3$)	6959	6017	167	**38**	**38**	80	67	464
Fourth ($k = 4$)	10619	8955	167	**32**	33	79	70	723
($k = 1$) Search Nodes	88	**87**	726	5.3×10^6	6.7×10^4	3.9×10^6	2.9×10^6	7.4×10^4
($k = 4$) Search Nodes	85	**84**	460	2.7×10^6	3.6×10^5	1.0×10^6	7.6×10^3	3.4×10^4

Notes: Colouring problem parameters $\langle 30, 5, 0.227\rangle$.
First 4 rows are average costs over 100 problems
Bottom two rows gives mean search nodes for $k = 1$ and $k = 4$

The relative performance of the algorithms is quite similar to those of Table 3 with *ECB* incurring least cost overall. *Basic Iter* incurs roughly a factor of 5 more cost on average, while *Basic* again has average cost one or two orders of magnitude worse than *ECB*. Interestingly, there is very little difference in the average cost between *ECB* and the *ECB-CL* algorithm. Furthermore, the magnitude of improvement over the *ECB-SL* and *Cost Only* algorithms is less for these problems, although it is still significant (at least 40% improvement).

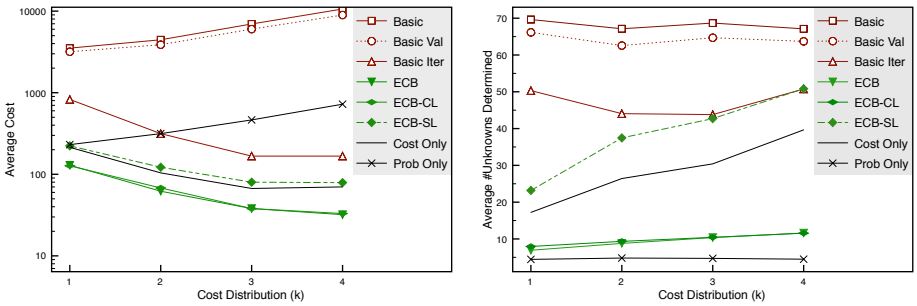
As expected, the search efficiency of *ECB* is worse for these problems as propagation for colouring problems is generally weak. Here, the *ECB-CL* algorithm is roughly one to two orders of magnitude better than *ECB* in terms of search nodes explored depending on the cost distribution.

Figure 4 illustrates the average costs (4a) and average number of unknowns determined (4b) for the colouring problems. Again we can see that Although Figure 4a is nearly identical to Figure 2a, there are some interesting differences between the number of unknowns determined by the algorithms on the different problem types. The most noticeable difference is in the number of unknowns determined by *ECB-SL* and *Basic Iter* in comparison with *Basic*, for the 20 variable random problems these two algorithms had determined nearly twice as many unknowns as *Basic*, here the opposite is the case.

Summary We have shown, across a range of problems of different sizes, that the ECB algorithm consistently performs best, i.e., finds solutions with least cost, when compared to a number of algorithms. Furthermore, we have shown that using cost information about unknowns, but ignoring their probabilities, resulted

in determining many more unknowns which had true value 0, thereby incurring a larger cost than ECB. Similarly, using probability information about unknowns but ignoring their cost information can result in a huge cost, even though it required few unknowns to be determined to find a solution. This clearly shows the importance of using both cost and probability information for minimising cost when solving ECI-CSPs.

Although ECB can be quite expensive to run in terms of search effort, we have shown, with the results for the ECB-CL algorithm, that this aspect of the algorithm can be greatly improved with little fall off in terms of cost incurred. Finally, the relatively poor performance of the *ECB-SL* algorithm illustrates the benefits of delaying determining unknowns until leaf nodes.



(a) Average Cost (b) Unknowns Determined

Fig. 4: Colouring problems $\langle 30, 5, 0.227 \rangle$, different cost distributions.

6 Extending the Framework and Algorithms for Multi-valued Unknowns

The assumption that unknowns be Boolean can be restrictive. It can be useful to allow more general forms of unknowns, for instance, to model a constraint $X - Y \geq \lambda$ where λ is an unknown constant which can take values in a numerical domain. In Section 6.1, we give an example of a Constraint Satisfaction Problem involving such multi-valued unknowns. We define the extended framework in Section 6.2, and show, in Section 6.3, how the minimum expected scaled cost algorithm can be extended.

6.1 Example involving multi-valued unknowns

Suppose that we have two tasks, A and B, to schedule, each starting in time period 1, 2, 3, or 4. Both tasks have the same unknown duration, λ_1, where $\lambda_1 \in \{1, 2, 3\}$, and the two tasks cannot overlap. Both tasks have pre-conditions

that affect their possible start times: Task A can be started at time λ_2 or later, and Task B can be started at time λ_3 or later, where λ_2 and λ_3 are unknown values in $\{1, 2, 3, 4\}$.

Letting the start times of tasks A and B be X and Y, respectively, we have the following constraints: $X \geq \lambda_2$, $Y \geq \lambda_3$ and $|X - Y| \geq \lambda_1$. We need to generate a solution, i.e., an assignment to variables X and Y, which we are sure satisfies the constraints.

Parameters λ_1, λ_2 and λ_3, although determinate, are unknown; we can determine (i.e., find out) the values of each, but there is a cost of doing so. Specifically, the costs of determining unknown parameters λ_1, λ_2 and λ_3 are 50, 100 and 70, respectively. For λ_2 we estimate that the probability distribution of each element in its domain is equal, i.e., 0.25 for each of $1, 2, 3, 4$; similarly for λ_3. Our probability distribution over the possible values of λ_1 is $0.25, 0.5, 0.25$ for values 1, 2 and 3, respectively (see Table 6).

To generate a solution we will need to determine at least one unknown parameter. The problem is to decide how to determine unknown parameters in such a way as to minimise the expected cost of generating a solution.

Table 6: The unknown parameters.

Unknown parameter	λ_1	λ_2	λ_3
Cost K_i	50	100	70
Domain	$\{1, 2, 3\}$	$\{1, 2, 3, 4\}$	$\{1, 2, 3, 4\}$
Probability distribution	$(0.25, 0.5, 0.25)$	$(0.25, 0.25, 0.25, 0.25)$	$(0.25, 0.25, 0.25, 0.25)$

6.2 The framework for multi-valued unknowns

These kind of problem is very similar to that discussed in earlier sections, and one could attempt to model it using an ECI-CSP. For example, to represent the constraint $X \geq \lambda_2$, we might try to use an incomplete constraint c with $c(X = 1) = u_1$, $c(X = 2) = u_2$, $c(X = 3) = u_3$, and $c(X = 4) = 1$, where $p_1 = 0.25$, $p_2 = 0.5$ and $p_3 = 0.75$. However, one would also need to represent dependencies between the unknowns, with, $u_1 = 1$ implying $u_2 = 1$, which in turns implies $u_3 = 1$. In addition, the formalism would have to be extended to allow there to be a single cost for determining a group of unknowns.

Instead of this, we take a different approach; the Boolean unknowns of ECI-CSPs are generalised to unknown parameters which can take more than two values. Furthermore, an incomplete constraint now associates with a tuple both an unknown parameter and a subset of the domain of the unknown parameter. For example, we represent $X \geq \lambda_2$ with the incomplete constraint c_2 defined by $c_2(X = 1) = (\lambda_2, \{1\})$, $c_2(X = 2) = (\lambda_2, \{1, 2\})$, $c_2(X = 3) = (\lambda_2, \{1, 2, 3\})$ and

$c_2(X = 4) = 1$. Thus, for example, the assignment $X = 2$ satisfies this constraint if and only if $\lambda_2 \in \{1, 2\}$. The constraints for the example are summarised in Table 7.

Table 7: The constraints.

Incomplete constraint c_1 representing $|X - Y| \geq \lambda_1$.

$c_1(X, Y)$	$X = 1$	$X = 2$	$X = 3$	$X = 4$
$Y = 1$	0	$(\lambda_1, \{1\})$	$(\lambda_1, \{1, 2\})$	1
$Y = 2$	$(\lambda_1, \{1\})$	0	$(\lambda_1, \{1\})$	$(\lambda_1, \{1, 2\})$
$Y = 3$	$(\lambda_1, \{1, 2\})$	$(\lambda_1, \{1\})$	0	$(\lambda_1, \{1\})$
$Y = 4$	1	$(\lambda_1, \{1, 2\})$	$(\lambda_1, \{1\})$	0

Incomplete constraint c_2 representing $X \geq \lambda_2$.

	$X = 1$	$X = 2$	$X = 3$	$X = 4$
$c_2(X)$	$(\lambda_2, \{1\})$	$(\lambda_2, \{1, 2\})$	$(\lambda_2, \{1, 2, 3\})$	1

Incomplete constraint c_3 representing $Y \geq \lambda_3$.

	$Y = 1$	$Y = 2$	$Y = 3$	$Y = 4$
$c_3(Y)$	$(\lambda_3, \{1\})$	$(\lambda_3, \{1, 2\})$	$(\lambda_3, \{1, 2, 3\})$	1

The Unknowns: As well as decision variables V, we consider a disjoint set of variables Λ, which we call the set of *unknown parameters*. The domain of $\lambda \in \Lambda$, written as δ_λ, is finite and has at least two values. We assume that, for any unknown $\lambda \in \Lambda$, we can determine the value of λ, that is, the element of δ_λ that λ is equal to. So we assume we have some procedure $Det(\cdot)$ that takes an unknown λ as input and returns an element of δ_λ.

As before, we also assume that there is a certain cost $K_\lambda \in [0, \infty)$ for executing this procedure on λ, and that we have a probability distribution P_λ over the domain δ_λ, indicating the probabilities over the true value of λ. We assume that the unknown parameters are probabilistically independent. A scenario is again defined to be a complete assignment to all the unknowns, i.e., a function α which assigns a value $\alpha(\lambda)$ in δ_λ to each unknown parameter $\lambda \in \Lambda$. Let $\Pr(\alpha)$ be the probability of scenario α occurring. Since the variables Λ are probabilistically independent, we have $\Pr(\alpha) = \prod_{\lambda \in \Lambda} P_\lambda(\alpha(\lambda))$.

Incomplete Constraints for multi-valued unknowns: An *incomplete constraint* c over (V, Λ) has an associated subset V_c of V called its *scope*. c is a function on $D(V_c)$, where, for tuples $t \in D(V_c)$, $c(t)$ is either, 1, 0 or a pair (λ, c_t), where c_t is a non-empty proper subset of δ_λ.

c is intended as a partial representation of some standard constraint c^* over V_c. $c(t) = 1$ is interpreted as: *it is known that t satisfies the constraint c^**. Also,

$c(t) = 0$ is interpreted as: *it is known that t doesn't satisfies the constraint* c^*; otherwise, if $c(t)$ equals a pair (λ, c_t), then it is unknown if t satisfies the constraint. If it turns out that $\lambda = a$, i.e., if $Det(\lambda) = a$, and $a \in c_t$, then tuple t satisfies the incomplete constraint, i.e., t satisfies c^*. If there exists tuple t with $c(t)$ equalling a pair of the form (λ, c_t), then we say that unknown parameter λ is *associated with* c.

A Multi-valued Expected Cost-based Interactive CSP (MECI-CSP) is defined to be a tuple $\langle V, D, \Lambda, \delta, K, P, C\rangle$, for set of variables V with the domains specified by function D, so that variable X has domain $D(X)$, probabilistically independent set of unknowns Λ, and function $K : \Lambda \to [0, \infty)$; C is a set of incomplete constraints over (V, Λ), δ is the function that associates the domain (state space) δ_λ with each unknown parameter λ, and P is the function that associates with unknown parameter λ a probability distribution P_λ over δ_λ. It is also assumed that each unknown parameter $\lambda \in \Lambda$ is associated with at most one incomplete constraint in C.

As before we associate, with an incomplete constraint c, two standard constraints with the same scope: the known constraint and the potential constraint. The *known constraint* $\underline{c}$ is given by $\underline{c}(t) = 1$ if and only if $c(t) = 1$ (otherwise, $\underline{c}(t) = 0$). A tuple satisfies $\underline{c}$ if and only if it is known to satisfy c^*. The *potential constraint* $\overline{c}$ is given by $\overline{c}(t) = 0$ if and only if $c(t) = 0$ (otherwise, $\overline{c}(t) = 1$). A tuple satisfies $\overline{c}$ if it could potentially satisfy c^*. For a given set of incomplete constraints C, the *Known CSP* is the set of associated known constraints: $\underline{C} = \{\underline{c} : c \in C\}$, and the *Potential CSP* $\overline{C}$ is the set of associated potential constraints: $\{\overline{c} : c \in C\}$.

Let λ be an unknown parameter. Suppose we determine λ and find out that $\lambda = a$, for some value $a \in \delta_\lambda$. Consider any constraint c and tuple t involving λ, so that $c(t) = (\lambda, c_t)$. If we have $a \in c_t$, then we now know that t does satisfy the constraint, so we can replace $c(t) = (\lambda, c_t)$ by $c(t) = 1$. Similarly, if $a \notin c_t$ then we can replace $c(t) = (\lambda, c_t)$ by $c(t) = 0$. Define $c[\lambda = a]$ to be the incomplete constraint generated from c in this way, i.e., c with λ instantiated to a.

More generally, let ω be an assignment to a set $\mathcal{W} \subseteq \Lambda$ of unknown parameters, and let c be an incomplete constraint. $c[\omega]$ is the incomplete constraint obtained by replacing each λ in $\mathcal{W}$ with the value $\omega(\lambda) \in \delta_\lambda$, i.e., c is the incomplete constraint produced by iteratively instantiating each λ to $\omega(\lambda)$. We define $C[\omega]$ to be $\{c[\omega] : c \in C\}$. $C[\omega]$ is thus the incomplete CSP updated by the extra knowledge ω we have about the unknown parameters.

We say that incomplete CSP C *is solved by assignment S (to variables V) in the context* ω if S is a solution of the associated known CSP $\underline{C[\omega]}$. In other words, if S is known to be a solution of C given ω. An incomplete CSP C is *insoluble in the context* ω if the associated potential CSP $\overline{C[\omega]}$ has no solution.

We can define policies in just the same way as in Section 2.2. Given an assignment ω to some (possibly empty) set $\mathcal{W}$ of unknowns, a policy does one of the following:

(a) returns a solution of the Known CSP (given ω);

(b) returns "Insoluble" (it can only do this if the Potential CSP (given ω) is insoluble);
(c) choose another undetermined unknown.

A policy iteratively chooses unknowns to determine until it terminates. The expected cost, $\mathrm{EC}(\pi)$, of a policy π can be defined in just the same way as in Section 2.2, i.e., the sum over all scenarios α of $\Pr(\alpha)K_\alpha$, where K_α is the sum of costs of all the unknown parameters determined by the policy in scenario α. A policy is optimal if it has minimal expected cost over all policies.

Consider MECI-CSP $\langle V, D, \Lambda, \delta, K, P, C\rangle$. Let ω be an assignment to some set of unknowns $\mathcal{W} \subseteq \Lambda$. Define MECI-CSP $\mathcal{E}_\omega$ to be the $\mathcal{E}$ updated with ω, i.e., $\langle V, D, \Lambda - \mathcal{W}, \delta, K, P, C[\omega]\rangle$ where functions δ, K and P are restricted to $\Lambda - \mathcal{W}$. Define $A_\mathcal{E}(\omega)$ to be the minimal expected cost over all policies for solving $\mathcal{E}_\omega$.

The following proposition can be proved in almost exactly the same way as Proposition 1.

Proposition 4. *Let $\mathcal{E} = \langle V, D, \Lambda, \delta, K, P, C\rangle$, be an MECI-CSP, let $\mathcal{W}$ be a subset of Λ, and let $\omega \in D(\mathcal{W})$ be an assignment to $\mathcal{W}$. If $\underline{C[\omega]}$ is soluble or $\overline{C[\omega]}$ is insoluble then $A_\mathcal{E}(\omega) = 0$. Otherwise,*

$$A_\mathcal{E}(\omega) = \min_{\lambda \in \Lambda - \mathcal{W}} \big(K_\lambda + \sum_{a \in \delta_\lambda} P_\lambda(a) A_\mathcal{E}(\omega \cup \{\lambda = a\})\big).$$

This leads, in the same way as in Section 2.3, to a dynamic programming algorithm for finding the value of an optimal policy.

6.3 Extending the minimum expected scaled cost algorithm

It turns out that the minimum expected scaled cost algorithm can be easily extended to MECI-CSPs; as before we choose potential solutions based on their scaled expected cost, and then determine if they are actual solutions in the order that minimises expected cost.

Consider a potential solution S (i.e., solution of the associated Potential CSP $\overline{C}$), so that for all $c \in C$, $c(S) \neq 0$. Let $C'(S)$ be the set of incomplete constraints in C with $c(S) \neq 1$. Then for all incomplete constraints $c \in C'(S)$ we have $c(S) = (\lambda, c_S)$ for some unknown parameter λ, where c_S is defined to be c_t, with t being the projection of S to the scope of c. We write $u_c(S)$ for the event that $Det(\lambda) \in c_S$, and let $U = \{u_c(S) : c \in C'(S)\}$. Complete assignment S is a solution if and only if $u_c(S)$ holds for all $c \in C'(S)$. Let $p_c(S)$ be the probability that $u_c(S)$ holds, i.e., $P_\lambda(c_S)$.

To emphasise the connection with the situation and approach described in Section 3, let us relabel U as $\{u_1, \ldots, u_m\}$, let p_i be the probability that u_i holds, and let K_i be the cost associated with the unknown parameter corresponding to u_i. By independence, the probability that S is a solution of the (unknown) CSP is $p_1 p_2 \cdots p_m$, which we again write as $P(U)$. We can define $R(U)$ in just the same way as in Section 3.1, as the minimal expected cost of determining if all u_i hold, i.e., of checking if S is a solution.

We again use algorithmic approaches based on focusing on minimising the scaled expected cost, i.e., $R(U)/P(U)$. In particular we can choose a potential solution with minimal value of scaled expected cost, and then check if this is an actual solution. One can slightly improve the expected cost of this algorithm with the following amendment: after determining an unknown parameter and instantiating it with its true value, we check to see if the Known CSP is now satisfiable (in which case we needn't determine any further unknowns). Similarly the iterative expected cost-bound algorithm can be adapted for MECI-CSPs.

Example continued: Consider the complete assignment $S = (X = 4, Y = 2)$. Recall that c_1 is the incomplete constraint representing $|X-Y| \geq \lambda_1$, incomplete constraint c_2 represents $X \geq \lambda_2$, and c_3 represents $Y \geq \lambda_3$. Then $c_1(S) = (\lambda_1, \{1, 2\})$, since S satisfies c_1 if and only if the true value of parameter λ_1 is either 1 or 2. $c_2(S) = 1$, and $c_3(S) = (\lambda_3, \{1, 2\})$. Thus $C'(S) = \{c_1, c_3\}$. u_{c_1} is the event $\lambda_1 \in \{1, 2\}$, and u_{c_3} is the event $\lambda_3 \in \{1, 2\}$. S is a solution of the unknown CSP if and only if $\lambda_1 \in \{1, 2\}$ and $\lambda_3 \in \{1, 2\}$, i.e., if and only if both u_{c_1} and u_{c_3} hold. The probability that S is a solution is therefore $P_{\lambda_1}(\{1, 2\}) \times P_{\lambda_3}(\{1, 2\}) = 0.75 \times 0.5 = 0.375$. If we determine λ_1 first and then λ_3 then the expected cost of determining if S is a solution is $K_1 + P_{\lambda_1}(\{1, 2\})K_3 = 50 + 0.75 \times 70 = 102.5$; if we determine λ_3 first then the expected cost is $K_3 + P_{\lambda_3}(\{1, 2\})K_1 = 70 + 0.5 \times 50 = 95$, so the minimum expected cost of determining if S is a solution is 95. Hence the scaled expected cost is $95/0.375 = 253\frac{1}{3}$. In fact, $(X = 4, Y = 2)$ has minimal scaled expected cost among all potential solutions.

The scaled expected cost algorithm will therefore determine first if $(X = 4, Y = 2)$ is a solution, and will determine unknown parameter λ_3 first.

If $Det(\lambda_3) = 1$ then the incomplete constraint c_3 becomes $Y \geq 1$, which is trivially satisfied. The Known CSP has a solution, specifically, $(X = 4, Y = 1)$, and so the algorithm can stop here. If $Det(\lambda_3) = 2$, Boolean unknown u_{c_3} holds, so we go on to determine λ_1.

If $Det(\lambda_3) = 3$ or 4 then $(X = 4, Y = 2)$ is not a solution, so we choose another potential solution to evaluate. If $Det(\lambda_3) = 3$, the best solution (according to minimising scaled expected cost) to check next is $(4, 3)$. If $\lambda_3 = 4$, and so $Y = 4$, the best solution to check next is $(2, 4)$. This algorithm turns out to be optimal for this particular problem instance, or close to it (depending on how ties are broken when checking solution $(2, 4)$ for the $\lambda_3 = 4$ case).

7 Further Extensions and Summary

Extending the algorithms: Our main algorithm can be considered as searching for complete assignments with small values of $R(U)/P(U)$, where U is the set of unknowns associated with the assignment, $P(U)$ is the probability that all of U are successfully determined (and hence that the assignment is a solution), and $R(U)$ is the expected cost of checking this. There are other ways of searching for assignments with small values of $R(U)/P(U)$, in particular, one could use

local search algorithms or branch-and-bound algorithms. Such algorithms can be used to generate promising assignments, which we can sequentially test to see if they are solutions or not. If not, then we move on to the next potential solution (possibly updating the problem to take into account determined unknowns).

The efficiency of our main algorithm and a branch-and-bound algorithm would probably be greatly increased if one could design an efficient propagation mechanism of an upper bound constraint on $R(U)/P(U)$. Failing that, one might use a propagation method for weighted constraints [11] to prune subtrees of assignments with total associated cost above a threshold, or with probabilities below a threshold (the latter using a separate propagation, making use of the log/exponential transformation between weighted constraints and probabilistic constraints).

Further extensions of the model: Our model of interleaving solving and elicitation is a fairly simple one. There are a number of natural ways of extending it to cover a wider range of situations. In particular, the framework and algorithms can be easily adapted to situations where there is a cost incurred for determining a *set* of unknowns (rather than a single unknown); for example, there may be a single cost incurred for determining all the unknowns in a particular constraint. The paper has focused on the case of the probabilities being independent; however, the model and algorithms can be applied in non-independent cases as well. Our model and algorithms also apply to the case where determining an unknown may leave it still unknown; unsuccessfully determining an unknown then needs to be reinterpreted as meaning that we are unable to find out if the associated tuple satisfies the constraint or not. Our current model allows a single unknown to be assigned to several tuples (which may be in the same constraint); although this can allow some representation of intensional constraints, we may also wish to allow non-Boolean unknowns, for example, for a constraint $X - Y \geq \lambda$ where λ is an unknown constant.

The framework and algorithms can also be extended to the case of optimisation for soft constraints problems [22], with much of the framework similar to those in Sections 2 and 6. This relates also to the framework in [9], but with a probabilistic basis. An *incomplete soft constraint* then assigns to tuples either an element of I, where I is the set of preference degrees, or an unknown, where an unknown is associated with a subset of I, and a probability distribution over this subset. The problem is to generate an optimal solution for a collection of incomplete soft constraints, whilst determining unknowns with as little cost as possible. The definition of a policy is a little different from that in Sections 2.2 and 6.2: given an assignment to a set of unknowns, a policy either (a) returns a necessarily optimal complete assignment, i.e. a complete assignment which is optimal in all scenarios; or, (b) chooses another undetermined unknown to determine. One might also consider other elicitation models, such as those described in [9], where, for example, the worst missing preference degree associated with a potential solution is returned.

In many situations, it could be hard to reliably estimate the success probabilities and costs, in particular, if a cost represents the time needed to find the

associated information. However, since the experimental results indicate that taking costs and probabilities into account can make a very big difference to the expected cost, it could be very worthwhile making use of even very crude estimates of costs and probabilities.

Summary

The paper defines a particular model for when solving and elicitation are interleaved, which takes costs and success probabilities into account. A formal representation of such a problem is defined. A dynamic programming algorithm can be used to solve the problem optimally, i.e., with minimum expected cost; however this is only computationally feasible for situations in which there are few unknowns, i.e., very little unknown information. We define and experimentally test a number of algorithms based on backtracking search, with the most successful (though computationally expensive) ones being based on delaying determining an unknown until more information has been received.

Acknowledgements

This material is based upon works supported by the Science Foundation Ireland under Grant No. 05/IN/I886 and Grant No. 08/PI/I1912.

References

1. Faltings, B., Macho-Gonzalez, S.: Open constraint satisfaction. In: Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming (CP-2002). (2002) 356–370
2. Faltings, B., Macho-Gonzalez, S.: Open constraint optimization. In: Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming (CP-2003). (2003) 303–317
3. Faltings, B., Macho-Gonzalez, S.: Open constraint programming. Artificial Intelligence **161**(1–2) (2005) 181–208
4. Cucchiara, R., Lamma, E., Mello, P., Milano, M.: An interactive constraint-based system for selective attention in visual search. In: International Syposium on Methodologies for Intelligent Systems. (1997) 431–440
5. Lamma, E., Mello, P., Milano, M., Cucchiara, R., Gavanelli, M., Piccardi, M.: Constraint propagation and value acquisition: why we should do it interactively. In: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99). (1999) 468–477
6. Lallouet, A., Legtchenko, A.: Consistencies for partially defined constraints. In: Proc. International Conference on Tools with Artificial Intelligence (ICTAI'05). (2005)
7. Gelain, M., Pini, M.S., Rossi, F., Venable, K.B.: Dealing with incomplete preferences in soft constraint problems. In: Proc. CP'07. Volume 4741 of LNCS., Springer (2007) 286–300
8. Gelain, M., Pini, M.S., Rossi, F., Venable, K.B., Walsh, T.: Elicitation strategies for fuzzy constraint problems with missing preferences: Algorithms and experimental studies. In: Proc. CP'08. Volume 5202 of LNCS., Springer (2008) 402–417

9. Gelain, M., Pini, M.S., Rossi, F., Venable, K.B., Walsh, T.: Elicitation strategies for soft constraint problems with missing preferences: Properties, algorithms and experimental studies. Artificial Intelligence **174**(3-4) (2010) 270 – 294
10. Amilhastre, J., Fargier, H., Marquis, P.: Consistency restoration and explanations in dynamic CSPs—Application to configuration. Artificial Intelligence **135** (2002) 199–234
11. Larrosa, J., Schiex, T.: Solving weighted CSP by maintaining arc consistency. Artificial Intelligence **159**(1–2) (2004) 1–26
12. Fargier, H., Lang, J.: Uncertainty in Constraint Satisfaction Problems: a probabilistic approach. In: Proc. ECSQARU-93. (1993) 97–104
13. Wilson, N., Grimes, D., Freuder, E.: A cost-based model and algorithms for interleaving solving and elicitation of csps. In: Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming (CP-2007). (2007) 666–680
14. Dechter, R., Dechter, A.: Belief maintenance in dynamic constraint networks. In: Proc. AAAI-88. (1988) 37–42
15. Howard, R., Matheson, J.: Influence diagrams. In: Readings on the Principles and Applications of Decision Analysis. (1984) 721–762
16. Puterman, M.: Markov Decision Processes, Discrete Stochastic Dynamic Programming. John Wiley & Sons (1994)
17. Tarim, S.A., Manadhar, A., Walsh, T.: Stochastic constraint programming: A scenario-based approach. Constraints **11**(1) (2006) 53–80
18. Bellman, R.: Dynamic Programming. Princeton University Press (1957)
19. Dechter, A., Dechter, R.: On the greedy solution of ordering problems. ORSA Journal on Computing **1**(3) (1989) 181–189
20. Gent, I.P., MacIntyre, E., Prosser, P., Smith, B.M., Walsh, T.: Random constraint satisfaction: Flaws and structure. Constraints **6**(4) (2001) 345–372
21. Brélaz, D.: New methods to color the vertices of a graph. Communications of the ACM **22**(4) (1979) 251–256
22. Bistarelli, S., Montanari, U., Rossi, F., Schiex, T., Verfaillie, G., Fargier, H.: Semiring-based CSPs and Valued CSPs: Frameworks, properties and comparison. Constraints **4**(3) (1999) 199–240