On Supporting Dynamic Constraint Satisfaction with Order of Magnitude Preferences

Jeroen Keppens and Qiang Shen
Centre for Intelligent Systems and their Applications
The University of Edinburgh
{jeroen,qiangs}@dai.ed.ac.uk

Abstract

Many application problems can be formulated as dynamic preference constraint satisfaction problems. Such a problem employs activity constraints that govern what attributes and constraints are part of the current problem description, and has a preference associated with each of the domain values. The preferences can be combined using any commutative, associative and monotonic operator to compute the preference of the overall solution. The important problem of expressing user preference under incomplete knowledge and combining them has not been addressed however. This paper introduces an order of magnitude preference (OMP) calculus to handle reasoning with preferences. The benefits of this calculus are twofold. Firstly, it allows for a partial ordering of preferences, rather than the usually imposed total ordering, thereby simplifying the knowledge required for problem formulation. Secondly, computational efficiency can be improved in solving complex problems as the OMP calculus ignores those preferences that are an order of magnitude lower than the ones that make a real difference to the overall quality of an emerging solution.

Introduction

A relatively recent trend in constraints research has been the enrichment of the existing representation framework, and the corresponding solution techniques, in order to be able to solve more sophisticated problems. A particular approach addresses the valued constraint satisfaction problems where valuations of emerging solutions are computed. Thus, a best quality solution can be searched instead of merely one that satisfies hard constraints (Schiex, T., Fargier, H., & Verfaillie, G. 1995). Another approach concerns about dynamic constraint satisfaction problems (DCSPs) in which activity constraints are employed to determine which attributes to become part of the problem (Mittal, S. & Falkenhainer, B. 1990).

This paper proposes an integrated approach to jointly handle dynamic *and* valued constraint satisfaction problems, which are hereafter referred to as dynamic preference constraint satisfaction problems (DPCSPs). Such a problem is a DCSP similar to that defined in (Mittal, S. & Falkenhainer, B. 1990), but the individual attribute-value assignments each have a corresponding preference. Each candidate solution of the CSP is associated with an overall preference, by combining the preferences of the individual assignments. The potential application of this approach covers a broad range of synthesis problems, including configuration problems (Mittal, S. & Falkenhainer, B. 1990), compositional modelling problems (Falkenhainer, B. & Forbus, K.D. 1991; Keppens, J. & Shen, Q. 2001) and planning problems (Blum, A. & Furst, M. 1997).

In recent work on compositional modelling of ecological systems (Keppens, J. & Shen, Q. 2000), for example, the suitability of DCSPs in efficiently guiding the selection of model fragments was explored. However, due to the incomplete knowledge of the laws that govern the ways in which ecological systems evolve over time, different ecologists usually have a distinct set of preferences over the choice of different approaches. The development of representational schemes and their associated solution mechanisms of DPCSPs will allow the existing DCSP-based compositional modelling techniques to be of a more general utility (although compositional modelling is itself not addressed here in detail).

Expressing human preferences and combining them in a consistent way can be very difficult. Indeed, it is well-known that when experts or users are requested to provide a total ordering of their options, either numerically or symbolically, they often find the rational consequences of their choices inconsistent or paradoxical (Green, D. & Shapiro, I. 1995; Tversky, A. & Thaler, R.H. 1990). For example, a preference ordering of population growth models could be totally unrelated to a given preference ordering of climate models.

To address the issue of incomplete knowledge on preference values, basic ideas of order of magnitude reasoning (OMR) (Raiman 1991) are employed. However, existing order of magnitude calculi are deeply rooted in the real-number line, and, as argued above, humans have great difficulty in reasoning about their preferences in a single totally ordered domain. Therefore, a new OMR calculus is suggested that can deal with a partially ordered underlying domain.

Problem Specification

Dynamic preference constraint satisfaction problems

A classical hard CSP is specified by

- a set of attributes $\mathbf{X} = \{x_1, \ldots, x_n\}$,
- a set of domains $\mathbf{D} = \{D_1, \ldots, D_n\}$ with $D_i = \{d_{i1}, \ldots, d_{in_i}\}$ for each attribute x_i, and

- a set of compatibility constraints $\mathbf{C^c}$, where a compatibility constraint c over attributes $x_i, \ldots, x_j$ is a relation $c : D_i \times \ldots \times D_j \rightarrow \{\top, \bot\}$.

A set of assignments $\{x_1 : d_{1k_1}, \ldots, x_i : d_{ik_i}, \ldots, x_j : d_{jk_j}, \ldots, x_n : d_{nk_n}\}$ is said to satisfy this compatibility constraint c if $c(d_{ik_i}, \ldots, d_{jk_j}) = \top$.

A DCSP, as defined in (Mittal, S. & Falkenhainer, B. 1990), is an extension of a hard CSP in which attributes can be active and inactive. An attribute x_i is said to be active (denoted by active(x_i)) if and only if it is assigned a value from its domain. The activity of attributes is governed by a set of activity constraints $\mathbf{C^a}$, which are defined via implications that establish conditions under which certain attributes become active. A set of assignments $\{x_1 : d_{1k_1}, \ldots, x_m : d_{mk_k}, \neg\text{active}(x_{m+1}), \ldots, \neg\text{active}(x_n)\}$ is said to satisfy an activity constraint a if the conjunction of assignments is not inconsistent with a, that is $(x_1 : d_{1k_1} \wedge \ldots \wedge x_m : d_{mk_k}, \neg\text{active}(x_{m+1}) \wedge \ldots \wedge \neg\text{active}(x_n)), a \nvdash \bot$.

The present work enriches the notion of DCSP, allowing the representation and thereforth the solution of dynamic preference constraint satisfaction problems (DPCSPs), by introducing to it elements from valued constraint satisfaction problems (Schiex, T., Fargier, H., & Verfaillie, G. 1995). More formally, a DPCSP extends a DCSP with a preference valuation $p(x_i : d_{ij}) \in \mathbb{P}$ for each assignment $x_i : d_{ij}$, where $\mathbb{P}$ denotes the domain of preference valuations. The preference of a (partial) solution $\{x_i : d_{ik_i}, \ldots, x_j : d_{jk_j}\}$ is computed as

$$p(x_i : d_{ik_i}, \ldots, x_j : d_{jk_j}) = p(x_i : d_{ik_i}) \oplus \ldots \oplus p(x_j : d_{jk_j})$$

where $\oplus$ is a commutative, associative, closed binary operation on $\mathbb{P}$. The preference values in $\mathbb{P}$ are partially ordered by $\prec$, where $p_i \prec p_j, p_i, p_j \in \mathbb{P}$ is interpreted so that the assignment associated with p_j has a higher preference over the assignment associated with p_i.

The solution of such a CSP consists of all sets of assignments $\{x_i : d_{ik_i}, \ldots, x_j : d_{jk_j}\}$ that satisfy all given compatibility and activity constraints, such that no other sets of assignments $\{x_p : d_{pk_p}, \ldots, x_q : d_{qk_q}\}$ satisfy the compatibility and activity constraints with $p(x_i : d_{ik_i}, \ldots, x_j : d_{jk_j}) \prec p(x_p : d_{pk_p}, \ldots, x_q : d_{qk_q})$.

DPCSPs differ from existing types of CSP in two respects. Firstly, they integrate the features of *dynamic* CSPs with those of *valued* CSPs, thus providing a richer representational framework. Secondly, unlike the vast majority of valued CSPs, which are types of so-called semiring-based CSPs (Bistarelli, S., Montanari, U., & Rossi, F. 1997), the preference combination operator $\oplus$ employed in this work is not assumed to be idempotent ($a \oplus a = a$). Idempotent combination operators are commonly employed in valued CSPs because they enable the use of existing local consistency algorithms (Schiex, T., Fargier, H., & Verfaillie, G. 1995), which are known to be effective and efficient. However, the semantics of idempotent combination operators are not always suitable for synthesis problems. In DPCSPs, the preferences express utility contributions of individual attribute-value assignments, and each of these utility contributions is presumed to add to the overall utility (and therefore $a \oplus a$ should be preferred over a, rather than $a \oplus a = a$).

Because of these feature, a DPCSP can be applied to address various synthesis problems, including for example, configuration, compositional modelling and planning problems. The reasons that the present approach is applicable to such problems are summarised in table 1 to save space.

Order of magnitude reasoning

As formalised in (Raiman 1991), order of magnitude reasoning (OMR) systems perform inferences based on a calculus of coarse values. Coarse values are abstract representations of precise values taken from a totally ordered set, usually the set of real numbers $\mathbb{R}$. A typical OMR calculus is then designed in such way that it generalises computations over precise values to computations over coarse values. This is of course the same approach taken by any qualitative reasoning system. What makes OMR distinct is that the coarse values are generally of different order of magnitude.

Depending on the way the coarse values are defined, different OMR calculi can be generated. This can be illustrated by means of a number of important examples. In FOG (Raiman 1986) and extensions to FOG such as O[M] (Mavrovouniotis, M.L. & Stephanopoulos, G. 1987; 1988), ROM(K) (Dague 1993b) and ROM($\mathbb{R}$) (Dague 1993a), coarse values are defined by means of ordering relations that express the distance between coarse values on a totally ordered domain in relation to the range they cover on that domain. NAPIER groups precise values in the same coarse value if they have the same logarithm with respect to a given base (Nayak, P.P. 1992; 1993).

These approaches employ one or more domain specific values to determine the grain size by which coarse values are defined and differentiated from one another. Other approaches, such as the work presented in (Murthy 1988) and (Travé-Massuyès, L. & Piera, N. 1989) generalise this notion of granularity. In particular, the latter defines coarse values as subsets or supersets of other coarse values, thus making an explicit link between algebra over $\mathbb{R}$ (or an abstraction thereof) and sign algebra.

OMR work has been applied in many areas. However, as before, when human preferences are projected onto a totally ordered domain, a judgement is made by comparing preferences that are unrelated and with which the human experts may not agree. To preserve the incomplete knowledge that is inherent to reasoning with preference a partially ordered domain is required. This work introduces such an OMR calculus.

Order of magnitude preference calculus

In DPCSPs valuations are attached to individual attribute assignments and combined to obtain a preference valuation for an emerging CSP solution. The actual "values" of these preferences do not matter, however, all that is useful is that they can be compared and combined with one another to derive an overall most preferred solution. Nevertheless, expressing preferences and combining them in a totally consistent manner throughout a complex CSP can be very difficult. Thus, only a partial order is employed in the development of the preference calculus herein.

	Configuration	Compositional Modelling	Planning
X	Components	Assumptions	Activities at time instance
D	Component options	Assumption classes	Enabled activities at given state
$\mathbf{C^c}$	Inconsistent combinations and requirements	Inconsistent parts of models and requirements	Inconsistent states and requirements
$\mathbf{C^a}$	Component prerequisites	Model fragment prerequisites	Prerequisite states
$\mathbb{P}$	Utility/costs	Utility contributions	Rewards, resource/time costs

Table 1: Applications of dynamic preference constraint satisfaction

Theoretical foundation

In this work, it is presumed that the user specifies a space $\mathbb{B}$ of basic preference quantities (BPQs). BPQs are the smallest units of preference valuation and are partially ordered.

Employing some of the underlying ideas of OMR, BPQs are related to one another by the "order of magnitude smaller than" relation $\ll$, the "equivalent order of magnitude as" relation $\sim$ and by the "smaller than within the same order of magnitude" relation $<$. Note that the latter relation also implies that the BPQs have an equivalent order of magnitude. Therefore, $\forall p_1, p_2 \in \mathbb{B}, p_1 < p_2 \rightarrow p_1 \sim p_2$. Naturally, order of magnitude smaller than relations are shared by all BPQs within the same order of magnitude. That is,

$$\forall p_1, p_2, p_3 \in \mathbb{B}, p_1 \sim p_2 \wedge p_2 \ll p_3 \rightarrow p_1 \ll p_3$$
$$\forall p_1, p_2, p_3 \in \mathbb{B}, p_1 \sim p_2 \wedge p_3 \ll p_2 \rightarrow p_3 \ll p_1$$

BPQs are combined with one another to form so-called order of magnitude preferences (OMPs). In general, the implicit value of an OMP P equals the combination $p_1 \oplus \ldots \oplus p_n$ of its constituent BPQs $p_1, \ldots, p_n$. In what follows, an approach will be presented to compute a partial ordering relation $\prec$ over the OMPs, based on the constituent BPQs of the OMPs. Generally speaking, the calculus is based on the following assumptions:

- The combination operator $\oplus$ is assumed to be commutative, associative and strictly monotonic ($P \prec P \oplus P$). The latter assumption is made to better reflect the ideas underpinning conventional utility calculi.
- A combination of BPQs is never an order of magnitude greater than its constituent BPQs. That is, given the following ordering of BPQs $p_1 \sim p_2 \sim \ldots \sim p_n \ll p$, then

$$p_1 \oplus p_2 \oplus \ldots \oplus p_n \prec p$$

- Distinctions at higher orders of magnitude are considered to be more significant than those at lower orders of magnitude. That is, given an ordering of BPQs $p_1 \sim \ldots \sim p_{m-1} \sim p_m \sim \ldots \sim p_n \ll p_a < p_b$, then

$$p_1 \oplus \ldots \oplus p_{m-1} \oplus p_a \prec p_m \oplus \ldots \oplus p_n \oplus p_b$$

  This assumption is commonly made in OMR. In terms of OMPs it means that the DPCSP algorithm will prioritise the optimisation associated with preferences of higher order of magnitude.
- Even though distinctions at higher orders of magnitude are more significant, distinctions at lower orders of magnitude are not negligible. That is, given an ordering of BPQs $p_1 < p_2$ and an OMP P, then $p_1 \oplus P \prec p_2 \oplus P$, irrespective of the orders of magnitude of the BPQs that constitute P. This is a departure from conventional OMR. If the OMPs associated with two (partial) DPCSP solutions contain equal BPQs at a higher order of magnitude, it is usually desirable to compare both solutions further in terms of the (less important) constituent BPQs at lower orders of magnitude.
- Conventional OMR is motivated by the need for abstract descriptions of real-world behaviour, whereas the OMP calculus is motivated by incomplete information. As opposed to conventional OMR, OMPs do not map onto the real number line. This implies that, when the user states, for example, that $p_1 < p_2 < p$ and that $p_3 < p_4 < p$, the explicit absence of ordering information between the BPQs in $\{p_1, p_2\}$ and those in $\{p_3, p_4\}$ means that the user can not compare them (e.g. because they are entirely different things). Consequently, $p_1 \oplus p_2$ would be deemed incomparable to $p_3 \oplus p_4$ (i.e. $p_1 \oplus p_2 ? p_3 \oplus p_4$), rather than roughly equivalent.

These assumptions can be formalised in a more general definition of the ordering relation $\prec$. Let an OMP $P = p_1 \oplus \ldots \oplus p_n$ be defined as a function $f_P : \mathbb{B} \rightarrow \mathbb{N} : p \mapsto f_P(p)$ where $\mathbb{B}$ is the set of BPQs, $\mathbb{N}$ is the set of natural numbers and $f_P(p)$ calculates the number of occurrences of p in $p_1, \ldots, p_n$. For example, given that $P = p_a \oplus p_b \oplus p_b$, then $f_P(p_a) = 1$ and $f_P(p_b) = 2$. Let $\mathbb{B}(p)$, $p \in \mathbb{B}$, be the subset of $\mathbb{B}$ that contains the BPQs of the same order of magnitude as p, i.e. $\mathbb{B}(p) = \{p_i \mid p_i \in \mathbb{B}, p_i \sim p\}$. Then, the constituent BPQs of an OMP P_1 that are within the same order of magnitude as a given BPQ p, are less than or equal to those of an OMP P_2 if $\forall p_i \in \mathbb{B}(p)$:

$$(f_{P_1}(p_i) + \sum_{p_j \in \mathbb{B}, p_i < p_j} f_{P_1}(p_j)) \leq (f_{P_2}(p_i) + \sum_{p_j \in \mathbb{B}, p_i < p_j} f_{P_2}(p_j))$$

This is denoted by $P_1 \preccurlyeq_p P_2$. The constituent BPQs of an OMP P_1 that are within the same order of magnitude as a given BPQ p, are less than but not equal to those of an OMP P_2 if $P_1 \preccurlyeq_p P_2 \wedge \neg(P_2 \preccurlyeq_p P_1)$. This is denoted by $P_1 \prec_p P_2$.

More generally, an OMP P_1 is less than an OMP P_2 if, for each distinct order of magnitude, either P_1 is less than P_2 for the BPQs within this order of magnitude, or there are BPQs at a higher order of magnitude for which P_1 is less than P_2:

$$P_1 \prec P_2 \leftarrow \forall p_a \in \mathbb{B}, (P_1 \prec_{p_a} P_2) \vee (\exists p_b \in \mathbb{B}, p_a \ll p_b \wedge P_1 \prec_{p_b} P_2)$$

It can be shown that this definition of $\prec$ results in a partial ordering of OMPs that meets the aforementioned assumptions. It allows for the case where two OMPs P_1 and P_2 are incomparable, denoted by $P_1 ? P_2$, meaning $\neg(P_1 \prec P_2) \wedge \neg(P_2 \prec$

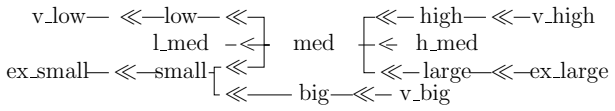


Figure 1: Sample OM preference scale

$P_1) \wedge (P_1 \neq P_2)$ holds. Note that $P_1 = P_2$ if P_1 and P_2 are a combination of the same collection of BPQs.

Figure 1 shows a sample set of BPQs and ordering relations between them. Based on it, the following comparisons can be made between OMPs:

$$\text{h\_med} \oplus \text{med} \prec \text{high} \quad (1)$$

$$\text{h\_med} \oplus \text{med} \succ \text{med} \oplus \text{l-med} \quad (2)$$

$$\text{high} \oplus \text{large} ? \text{big} \oplus \text{small} \quad (3)$$

Relation (1) is true because high is an order of magnitude greater than h_med and med. Relation (2) is justified by (h_med > med) and (med > l_med). The OMPs in (3) are incomparable because there is no path between big and high or between big and large.

Comparison of order of magnitude preferences

To show the decidability of the above theory, this subsection describes an algorithm to compare two OMPs. First, some further definitions must be introduced.

- The ordering relations $<$ and $\ll$, which are responsible for the ordering of BPQs are redefined for presentational simplicity as sets of pairs:

$$O_< = \{(p_1, p_2) \mid p_1 < p_2\}, \quad O_\ll = \{(p_1, p_2) \mid p_1 \ll p_2\}$$

- A *cross-over quantity* p with respect to an ordering relation $O \in \{O_<, O_\ll\}$, which is expressed by $\text{co}(p, O)$, is a BPQ for which O defines at least two BPQs to be greater or smaller than it. That is:

$$\forall p, \exists p_1, p_2, ((p_1, p) \in O \wedge (p_2, p) \in O) \vee ((p, p_1) \in O \wedge (p, p_2) \in O) \rightarrow \text{co}(p, O)$$

  In the ongoing example, small and med are the two cross-over quantities.

- A set of BPQs $\{p_1, p_2, \ldots, p_n\}$ is regarded as a *path* from p_b to p_e with respect to an ordering relation O, denoted by $\text{path}(O, p_b, p_e)$, if $(p_b, p_1) \in O$, $(p_1, p_2) \in O$, ..., $(p_n, p_e) \in O$. For example, {med, high} is a $\text{path}(O_\ll, \text{small}, \text{v\_high})$.

- The *distance* of a path equals the number of BPQs in it. That is, the distance $d(O, p_b, p_e)$ between two BPQs p_b and p_e with respect to an ordering relation O equals the largest distance of any paths between p_b and p_e. When there is a path from p_e to p_b, then $d(O, p_e, p_b) = -d(O, p_b, p_e)$. For example, $d(O_\ll, \text{small}, \text{v\_high}) = 2$.

- The space of BPQs $\mathbb{B}$ is partitioned into totally ordered subsets of BPQs. That is, given an ordering relation O, a strand $\text{str}(O, p_1, p_2)$ is a $\text{path}(O, p_1, p_2)$ such that

$$(\text{co}(p_1, O) \vee \nexists p' \in \mathbb{B}, (p', p_1) \in O) \wedge (\text{co}(p_2, O) \vee \nexists p' \in \mathbb{B}, (p_2, p') \in O)$$

  and that $\text{path}(O, p_1, p_2)$ does not contain any cross-over quantities. Additionally, each cross-over quantity p_c is said to be the only member of the strand $\text{str}(O, p_c, p_c)$. It can be shown that the strands defined by an ordering relation entail a unique partition of $\mathbb{B}$.

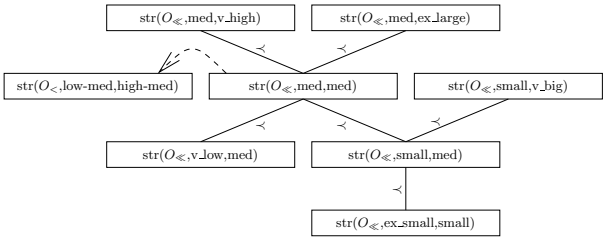


Figure 2: Ordering of strands

The strands can be compared with one another based on the BPQs within them. A strand s_1 is smaller than a strand s_2, denoted $s_1 < s_2$, if:

$$\forall p_1, p_2 \in \mathbb{B}, p_1 \in s_1 \wedge p_2 \in s_2 \rightarrow \text{path}(O, p_1, p_2)$$

Figure 2 shows the strands in the ongoing example and their ordering.

Each BPQ $p \in \mathbb{B}$ can now be uniquely identified by the strand $\text{str}(O, p_1, p_2)$ of which it is a member and by the distance $d(O, p_1, p)$. The tuple $\langle \text{str}(O, p_1, p_2), d(O, p_1, p) \rangle$ is said to be the *label* $L(p, O)$ of p within O.

Continuing with the example, the following BPQ labels can be computed:

$$L(\text{small}, O_\ll) = \langle \text{str}(O_\ll, \text{small}, \text{small}), 0 \rangle$$
$$L(\text{big}, O_\ll) = \langle \text{str}(O_\ll, \text{small}, \text{v\_big}), 1 \rangle$$

From this, BPQs can be compared in terms of their labels with respect to an individual ordering O. Given two BPQ labels $L(p_1, O) = \langle s_1, d_1 \rangle$ and $L(p_2, O) = \langle s_2, d_2 \rangle$, $\langle s_1, d_1 \rangle \geq \langle s_2, d_2 \rangle$ if:

$$(s_1 < s_2) \vee (s_1 = s_2 \wedge d_1 \geq d_2)$$

It can be shown that if $L(p_1, O) \geq L(p_2, O)$ a $\text{path}(O, p_1, p_2)$ exists. For instance, in the ongoing example, $L(\text{small}, O_\ll) \leq L(\text{big}, O_\ll)$.

For a given OMP $P = p_1 \oplus \ldots \oplus p_n$, where $p_1, \ldots, p_n \in \mathbb{B}$, a label can be computed by counting the number of each strand-distance pair in the labels of the constituent BPQs:

$$\mathcal{L}(P, O) = \{\langle s, d, n \rangle \mid (n = \sum_{p \in P, L(p,O) = \langle s,d \rangle} 1) \wedge (n \geq 1)\}$$

For example,

$$\mathcal{L}(\text{small} \oplus \text{big}, O_\ll) = \{ \langle \text{str}(O_\ll, \text{small}, \text{v\_big}), 1, 1 \rangle, \langle \text{str}(O_\ll, \text{small}, \text{small}), 0, 1 \rangle \}$$

In this way, two labels $\mathcal{L}(P, O_\ll)$ and $\mathcal{L}(P, O_<)$ can be computed for each OMP P. Both labels are related to one another through the underlying set of BPQs they represent. In particular, each strand-distance pair $\langle s_\ll, d_\ll \rangle$ under $O_\ll$ corresponds to a set of strand-distance pairs under $O_<$ that provide finer grain distinctions between quantities. Therefore, a label $\mathcal{L}(P, O_<, L)$ can be obtained, which contains the subset of

$\mathcal{L}(P, O_<)$ that corresponds to a subset $L \subset \mathcal{L}(P, O_\ll)$, such that:

$$\mathcal{L}(P, O_<, L) = \{\langle s_<, d_<, n_<\rangle \mid \langle s_<, d_<, n_<\rangle \in \mathcal{L}(P, O_<) \wedge (\exists \langle s_\ll, d_\ll, n_\ll\rangle \in L, \exists q, \langle s_<, d_<\rangle \in \mathcal{L}(q, O_<) \wedge \langle s_\ll, d_\ll\rangle \in \mathcal{L}(q, O_\ll))\}$$

Algorithm 1: COMPARE(P_1, P_2)

```
c_o ←'='; O_1 ← L(P_1, O_≪); O_2 ← L(P_2, O_≪);
while ¬(O_1 = ∅ ∨ O_2 = ∅)
do {
    B_1 ← {⟨s,d,n⟩ ∈ Ø_1 | ∄⟨s',d',n'⟩ ∈ O_1, ⟨s,d⟩ < ⟨s',d'⟩};
    B_2 ← {⟨s,d,n⟩ ∈ Ø_2 | ∄⟨s',d',n'⟩ ∈ O_2, ⟨s,d⟩ < ⟨s',d'⟩};
    H_1 ← {⟨s,d,n⟩ ∈ B_1 | ∄⟨s',d',n'⟩ ∈ B_2, ⟨s,d⟩ ≤ ⟨s',d'⟩};
    H_2 ← {⟨s,d,n⟩ ∈ B_2 | ∄⟨s',d',n'⟩ ∈ B_1, ⟨s,d⟩ ≤ ⟨s',d'⟩};
    if (H_1 ≠ ∅) ∧ (H_2 ≠ ∅)
      then return ('?');
      else {
        if (H_1 ≠ ∅)
          then { c_n ←'>'; E ← B_1 − H_1;
                 REMOVE(O_1, H_1); REMOVE(O_2, H_1); }
        else {
          if (H_2 ≠ ∅)
            then { c_n ←'<'; E ← B_2 − H_2;
                   REMOVE(O_1, H_2); REMOVE(O_2, H_2); }
            else c_n ←'='; E ← H_1;
        }
    if (c_o = c_n) ∨ (c_o ='=')
      then {
        (c_o, H) ← COMPARE-WM(E, P_1, P_2, O_<, c_n);
        if c_o ='?'
          then return ('?')';
        REMOVE(O_1, H); REMOVE(O_2, H);
      }
      else return ('?');
    O_1 ← O_1 − B_1; O_2 ← O_2 − B_2;
}
return (c_o);
procedure COMPARE-WM(E, P_1, P_2, O_<, c_o)
  H ← {};
  for each e ∈ E
  do {
    L_1 ← L(P_1, O_<, {e}); L_2 ← L(P_2, O_<, {e});
    if ≥_value (L_1, L_2)
      then {
        if ¬ ≥_value (L_2, L_1)
          then {
            if c_o ∈ {'=','>'}
              then { c_o ←'>'; H ← H ∪ {e}; }
              else return ('?', H);
          }
          else {
            if ≥_value (L_2, L_1)
              then {
                if c_o ∈ {'=','<'}
                  then { c_o ←'<'; H ← H ∪ {e}; }
                  else return ('?', H);
              }
              else return ('?', H);
          }
      }
  return (c_o, H);
```

By means of their respective labels, two OMPs can then be compared using the algorithm COMPARE(P_1, P_2). This algorithm iterates through the $O_\ll$ labels of P_1 and P_2. At each iteration, the sets of the highest remaining strand-distance pairs in the labels of P_1 and P_2, respectively denoted B_1 and B_2, are considered. The algorithm systematically compares the tuples $\langle s, d, n\rangle$ of B_1 and B_2 with one another:

- Each $\langle s, d, n\rangle \in B_1$ ($\langle s, d, n\rangle \in B_2$), such that $\langle s, d, n\rangle$ is either greater than or incomparable to the tuples in B_2 (B_1), is stored in a set H_1 (H_2), with $H_1 \neq \emptyset$ ($H_2 \neq \emptyset$) indicating that $P_1 \succ P_2$ ($P_2 \succ P_1$) for the tuples involved, i.e. $\forall p \in H_1, P_2 \prec_p P_1$ ($\forall p \in H_2, P_1 \prec_p P_2$).
- The tuples $\langle s, d, n\rangle \in B_1$ ($\langle s, d, n\rangle \in B_2$), such that a tuple $\langle s, d, n'\rangle \in B_2$ ($\langle s, d, n'\rangle \in B_1$) exists are stored in a set E. Because the BPQs referred by the tuples in E can not be compared with one another by means of the $O_\ll$ labels, the corresponding $O_<$ labels L_1 and L_2 are computed and compared in COMPARE-WM(). In order to determine whether L_1 is greater than L_2, this procedure employs the $\geq_{\text{value}}$ function:

$$\geq_{\text{value}} (L_1, L_2) \leftarrow \forall \langle s_2, d_2, n_2\rangle \in L_2, \sum_{\langle s,d,n\rangle \in L_1, \langle s_2,d_2\rangle \leq \langle s,d\rangle} n \geq \sum_{\langle s,d,n\rangle \in L_2, \langle s_2,d_2\rangle \leq \langle s,d\rangle} n$$

The results of these comparisons may be contradictory when at least one tuple from B_1 is greater than anything in B_2 and at least one tuple from B_2 is greater than anything in B_1. In that case, the algorithm terminates with returning incomparable ('?'). If the comparisons are not contradictory, then the tuples in B_1 and B_2 which yielded a strict ordering $'\prec'$ or $'\succ'$ (i.e. H_1 or H_2), need to be differentiated from the ones for which no such ordering could be established. All tuples in the labels of P_1 and P_2 that are smaller than H_1 or H_2 are removed. For the remaining tuples, the algorithm considers the next highest ones, that are still part of the labels, in the subsequent iteration or it terminates if there are no remaining tuples. The variables c_o and c_n are used in the algorithm to store the outcome of partial comparisons with respect to BPQs in other parts of the partial ordering.

For example, when applying this algorithm to compare P_1 = big $\oplus$ small with P_2 = high $\oplus$ large, the highest strand-distance pairs from the $O_\ll$ labels are compared first. This leads to $H_1 = \{\langle \text{str}(O_\ll, \text{med}, \text{v\_big}), 1, 1\rangle\}$ and $H_2 = \{\langle \text{str}(O_\ll, \text{med}, \text{v\_high}), 1, 1\rangle, \langle \text{str}(O_\ll, \text{med}, \text{v\_large}), 1, 1\rangle\}$ and hence, the algorithm returns them as incomparable. When comparing P_1 = h_med$\oplus$med with P_2 = med$\oplus$l_med, $H_1 = H_2 = \emptyset$ and $E = \{\langle \text{str}(O_\ll, \text{med}, \text{med}), 0, 2\rangle\}$. Therefore, COMPARE-WM$(E, P_1, P_2, O_<, '=')$ is called where

$$L_1 = \{\langle \text{str}(O_<, \text{l\_med}, \text{h\_med}), 2, 1\rangle, \langle \text{str}(O_<, \text{l\_med}, \text{h\_med}), 1, 1\rangle\}$$
$$L_2 = \{\langle \text{str}(O_<, \text{l\_med}, \text{h\_med}), 1, 1\rangle, \langle \text{str}(O_<, \text{l\_med}, \text{h\_med}), 0, 1\rangle\}$$

Because, $\geq_{\text{value}} (L_1, L_2)$ holds, the procedure returns $'\succ'$. As O_1 and O_2 are now empty, $'\succ'$ becomes the result of the algorithm, meaning $P_1 \succ P_2$.

Solution Techniques

This section presents two algorithms for solving DPCSPs. Although OMPs are used in this work, both algorithms can take any DPCSP provided that it employs a preference calculus with a commutative, associative and monotonic combination operator. However, the use of OMPs provides a convenient way of specifying incomplete preference information and yields efficiency improvements as a total ordering of preferences is not artificially imposed when none exists.

Basic algorithm

Algorithm 2: SOLVE(X, D, C, A, P)

$n \leftarrow$ createNode(nil, X_a); $X_a(n) \leftarrow \{x_i \mid \{\}, A \vdash \text{active}(x_i)\}$;
$O \leftarrow$ createOrderedQueue(); $CP(n) \leftarrow 0$;
$PP(n) \leftarrow \oplus_{x \in X} \max_{d \in D(x)} P(x : d)$;
PROCESS(first(X_a), X_a, n, C, A, P, O);
while $O \neq \emptyset$
do
- $n \leftarrow$ dequeue(O);
- **if** $X_u(n) \neq \emptyset$
  - **then**
    - $x \leftarrow$ first($X_u(n)$);
    - PROCESS(x, n, C, A, P, O);
  - **else**
    - $X_u(n) \leftarrow \{x_i \mid \text{solution}(n), A \vdash \text{active}(x_i)\} - X_a(n)$;
    - **if** $X_u(n) = \emptyset$
      - **then**
        - $n_{\text{next}} \leftarrow$ first(O);
        - **if** $CP(n) \not\prec PP(n_{\text{first}})$
          - **then return** ($S(n)$);
          - **else**
            - $PP(n) \leftarrow CP(n)$;
            - enqueue($O, n, CP(n), PP(n)$);
      - **else**
        - $x \leftarrow$ first($X_a(n)$);
        - PROCESS(x, n, C, A, P, O);

procedure PROCESS$(x, n_{\text{parent}}, C, A, P, O)$
for $d \in D(x)$
do
- **if** solution(n_{parent}) $\cup \{x : d\}, C \not\prec \perp$
  - **then**
    - $n_{\text{child}} \leftarrow$ new node;
    - solution(n_{child}) $\leftarrow$ solution(n_{parent}) $\cup \{x : d\}$;
    - $X_d \leftarrow$ deactivated(solution(n_{child}), $X(n_{\text{parent}})$);
    - $X_{nd}(n_{\text{child}}) \leftarrow X_{nd}(n_{\text{parent}}) - \{x\} - X_d$;
    - $X_a(n_{\text{child}}) \leftarrow X_a(n_{\text{parent}}) \cup \{x\}$; $X_u(n_{\text{child}}) \leftarrow X_u(n_{\text{parent}}) - \{x\}$;
    - $CP(n_{\text{child}}) \leftarrow CP(n_{\text{parent}}) \oplus P(x : d)$;
    - COMPUTE($x, n_{\text{child}}, n_{\text{parent}}, P, O$);

procedure COMPUTE$(x, n_{\text{child}}, n_{\text{parent}}, P, O)$
$PP(n_{\text{child}}) \leftarrow CP(n_{\text{child}}) \oplus (\oplus_{x \in X_{nd}(n)} \max_{d \in D(x)} P(x : d))$;
enqueue($O, n_{\text{child}}, PP(n_{\text{child}}), CP(n_{\text{child}})$);

This algorithm is somewhat similar to that presented in (Mittal, S. & Falkenhainer, B. 1990), but it implements a best first search (BFS) by means of a priority queue O of nodes n. For each node n, a set $X_u(n)$ of remaining active but unassigned attributes is maintained. At each iteration, a node n is taken from O, and the assignments of the first attribute $x \in X_u(n)$ are processed. For every assignment $x : d$ that is consistent with the solution of the current node n (i.e. solution$(n) \cup \{x : d\}, C \not\prec \perp$), a new child node is created. If $X_u(n)$ is empty, the activity constraints are fired in order to find a new set of active but unassigned attributes. That is,

$$X_u(n) = \{x_i \mid \text{solution}(n), A \vdash \text{active}(x_i)\} - X_a(n)$$

where $X_a(n)$ represents the active, but already assigned attributes in node n.

In the priority queue O, nodes are maintained by means of two heuristics: committed preference ($CP(n)$) and potential preference ($PP(n)$). These are defined as follows. Given a node n,

$$CP(n) = \oplus_{x:d \in \text{solution}(n)} P(x : d)$$
$$PP(n) = CP(n) \oplus (\oplus_{x \in X_{nd}(n)} \max_{d \in D(x)} P(x : d))$$

where $X_{nd}(n)$ is the set of unassigned attributes that can still be activated given the partial assignment solution(n) (the actual implementation employs an assumption-based truth maintenance system (de Kleer, J. 1986) to efficiently determine which attribute's activity can no longer be supported). In other words, $CP(n)$ is the preference associated with the partial attribute-value assignment in node n and $PP(n)$ is $CP(n)$ combined with the highest possible preference assignments taken from all the values of the domains of the attributes in X_{nd}. Thus, $PP(n)$ computes an upper boundary on the preference of a DPCSP solution that includes the partial attribute value assignments corresponding to n.

Theorem 1 SOLVE(X, D, C, A, P) *is admissible*
Proof: SOLVE(X, D, C, A, P) *is a BFS guided by a heuristic function* $PP(n) = CP(n) \oplus h(n)$*, where* $CP(n)$ *is the actual preference of node n and* $h(n) = \oplus_{x \in X_{nd}(n)} \max_{d \in D(x)} P(x : d)$*. It follows from the previous discussion that* $h(n)$ *is greater than or equal to the combined preference of any value-assignment of unassigned attributes that is consistent with the partial solution of n. In this BFS, the nodes n are maintained in a priority queue in descending order of* $PP(n)$*. Let* δ *be a distance function that reverses the preference ordering such that* $\delta(P_1) \prec \delta(P_2) \leftarrow P_1 \succ P_2$*.* SOLVE$(X, D, C, A, P)$ *can then be described as a BFS guided by* $\delta(PP(n)) = \delta(CP(n)) \oplus \delta(h(n))$*, where the nodes n are maintained in a priority queue in ascending order of* $\delta(PP(n))$ *and where* $\delta(h(n))$ *is a lower bound on the distance between n and the optimal solution. Therefore,* SOLVE(X, D, C, A, P) *is an A* algorithm, guaranteed to find a solution S with a minimal* $\delta(P(S))$ *or a maximal* $P(S)$*.*

Improved algorithm

Procedure 3: COMPUTE$(x, n_{\text{child}}, n_{\text{parent}}, P, O)$

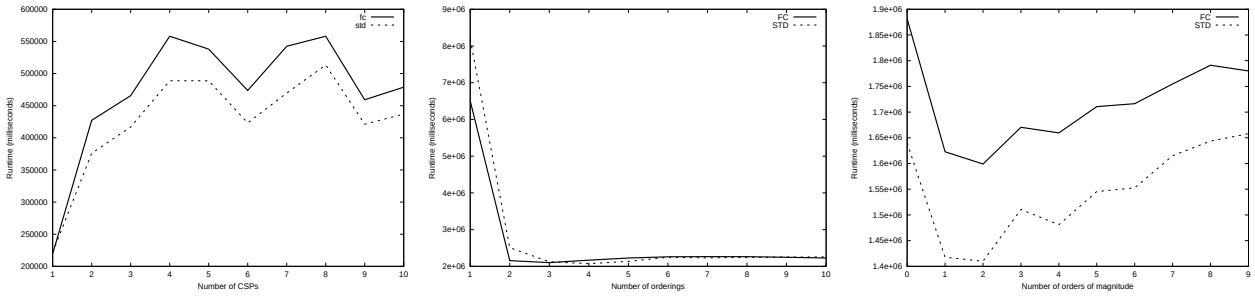
$PP(n_{\text{child}}) \leftarrow CP(n_{\text{parent}})$;
for $y \in$ successors(x, X_a)
do
- **if** $D_\top(y, n, C) \neq \emptyset$
  - **then**
    - $v_{\max} \leftarrow \max_{v \in D_\top(y,n,C)} P(y : v)$;
    - $PP(n_{\text{child}}) \leftarrow PP(n_{\text{child}}) \oplus v_{\max}$;
  - **else** $PP(n_{\text{child}}) \leftarrow$ nil

if $PP(n_{\text{child}}) \neq$ nil
then
- **for** $y \in X(n_{\text{child}})$
- **do**
  - $v_{\max} \leftarrow \max_{v \in D_\top(y,n,C)} P(y : v)$;
  - $PP(n_{\text{child}}) \leftarrow PP(n_{\text{child}}) \oplus v_{\max}$;
- enqueue($O, n_c, PP(n_{\text{child}}), CP(n_{\text{child}})$);

Forward checking, a technique commonly used for early detection of failing search paths, can be applied to extend the basic algorithm. In this work, it enables more accurate estimates of PP and hence focuses the search and potentially improves efficiency (Tsang, E. 1993). This variation of the basic algorithm works as follows: for each of the unassigned but active or potentially active attribute y, the set $D_\top(y, n, C)$ of all domain values that are consistent with the partial solution of the current node n is computed:

$$D_\top(y, n, C) = \{v \in y.D \mid \text{solution}(n) \cup \{x : d\} \cup \{y : v\}, C \not\prec \perp\}$$

When computing PP for the current node, only the assignments in $D_\top(y, n, C)$ are considered for each unassigned attribute y. When $D_\top(y, n, C)$ is empty, there is no consistent assignment for attribute y. In particular, if y is not active, no inconsistency will arise for the partial solution found so far. If, however, y is already active, no child nodes will be created for n because the problem then contains an attribute (y) that has no consistent assignment. As such, the modified algorithm combines an improved computation of PP with the standard early failure feature of conventional forward checking.



(a) Multiple CSPs (b) Partial ordering (c) Order of magnitude

Figure 3: Experimental results: runtimes

Results

This section briefly discusses preliminary results of applying the above algorithms to randomly generated DPCSP problems. The random DCSP generator used produces DPCSPs with a certain number of each of the following: sub-CSPs, attributes per sub-CSP, values per domain, attributes per antecedent of each activity constraint and attributes per compatibility constraint, and with the probabilities of a combination of attributes/values occurring in a compatibility/activity constraint. Then, a given set of preferences is assigned to the values of each domain according to a preset distribution.

Overall, the results obtained have shown that a DPCSP involving a numerical, or totally ordered, preference calculus is very complex. This conforms to the findings of (Schiex, T., Fargier, H., & Verfaillie, G. 1995): A valued CSP with a strictly monotonic combination operator ($p \oplus p \succ p$) is a far more complex problem than valued CSPs using idempotent operators ($p \oplus p = p$). Nevertheless, this can be an important class of problems since strict monotonicity may express well the combination of utility, costs or preferences appropriately in some domains.

The use of activity constraints and the partial orderings at different degrees of coarseness (provided by the OMP calculus), however, is not only beneficial to the descriptiveness of the problem specification, but it is also able to improve the required runtime performance. Figure 3(a) shows average runtimes for sets of simple CSPs (with 3 attributes each) linked by activity constraints (approximately 25% of attribute value combinations activate one or more attributes in the next sub-CSP). The x-axis of the figure shows the number of CSPs. Initially, adding sub-CSPs to the DPCSP severely increases runtime costs until a depth of about 3 levels of sub-CSPs. From then onwards, the increase in runtime is not very significant because further sub-CSPs are rarely instantiated. If this DPCSP were implemented as a valued CSP without explicit use of activity constraints, the low likelihood of activation of attributes would translate to high constrainedness and this would significantly affect the efficiency of the solution algorithms. This shows the importance of research into dynamic valued CSPs.

Figure 3(b) illustrates the average runtime performance (to the first solution) of applying the basic and improved algorithms to sets of 50 random DCSPs with different sets of preferences. The domain preference values in all these problems were randomly assigned one of 20 different OMPs. The OMPs consisted of a random number (up to 10) of single BPQs. The number of scales (shown in the x-axis) represents the number of BPQs for which OMPs were constructed. For example, in the case of $X = 4$, four unordered BPQs were created and five different OMPs were generated for each of them.

A total ordering of valuations corresponds to $X = 1$, and no two OMPs are incomparable in this case. As mentioned earlier, a partial ordering is a more appropriate structure in which to represent expert/user preferences. In addition, the results of test runs such as figure 3(b) indicate that the use of partial orderings is beneficial to the algorithms runtime as well. The reason for this is obviously that the space of best solutions increases as many solutions become incomparable.

Dividing the set of 20 quantities into different orders of magnitude improves performance only slightly, as is shown in figure 3(c) and only for certain partitionings. Again, this is due to the fact that the preference calculus is non-monotonic. When two OMPs have equivalent BPQs of higher orders of magnitude, a comparison proceeds with checking BPQs at lower orders of magnitude and this affects performance. A further approximate approach could just ignore the lower orders of magnitude in order to improve efficiency.

Conclusions

This paper has introduced a novel type of constraint satisfaction problem (CSP), the dynamic preference CSP (DPCSP) that incorporates features from dynamic *and* valued CSPs. From dynamic CSPs, a DPCSP takes activity constraints that govern which attributes and the corresponding constraints are part of the problem to be solved. From valued CSPs, a DPCSP borrows the concept of assigning valuations to domain values which can be combined to compute the overall valuation of an emerging solution.

To allow efficient representation and solution of DPCSPs, a preference calculus based on order of magnitude reasoning has been introduced. This calculus produces a partial ordering of valuations and distinguishes between different degrees of coarseness. Thus, it better suits the expression of user pref-

erences than approaches employing preferences that are totally ordered. Also, it helps improve search efficiency as the potential solution space would be larger if less distinctions can be made between solution valuations.

Following the idea of order of magnitude based DPCSPs, two solution algorithms have been presented. The first performs best-first search, adapted to deal with activity constraints, by working with a preference estimator that prevents it from getting stuck in local optima. The second is an extension of the first that employs forward checking techniques to fail inconsistent paths early.

Future work includes the development of alternative solution techniques for DPCSPs. One source of inefficiency of the algorithms presented herein is their insistence on finding an optimal solution. However, for many practical applications, finding a close to optimal solution often suffices (Tsang, E. & Warwick, T. 1990). Therefore, a future focus of this research is to investigate the use of genetic algorithms for solving a DPCSP.

The solution techniques for DPCSPs can be employed in various synthesis problems such as compositional modelling, configuration, planning and scheduling, though the actual applications are beyond the scope of this paper. Currently, this work is being employed by a compositional modeller for ecological systems. A knowledge base of model fragments is being built based on a large model, known as MODMED (Legg, C.J., Muetzelfeldt, R.I., & Heathfield, D.N. 1995), which describes how Mediterranean vegetation is being affected by various climate-related factors, managed and accidental fires and cattle farming. Such model-building tasks are dynamic and of various preferences over the choice of model fragments.

Acknowledgements

The first author has been supported by a scholarship of the Faculty of Science and Engineering of the University of Edinburgh.

References

Bistarelli, S.; Montanari, U.; and Rossi, F. 1997. Semiring-based constraint satisfaction and optimization. *Journal of the ACM* 44(2):201–236.

Blum, A., and Furst, M. 1997. Fast planning through planning graph analysis. *Artificial Intelligence* 90(1–2):281–300.

Dague, P. 1993a. Numeric reasoning with relative orders of magnitude. In *Proceedings of the National Conference on Artificial Intelligence*, 541–547.

Dague, P. 1993b. Symbolic reasoning with relative orders of magnitude. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, 1509–1514.

de Kleer, J. 1986. An assumption-based TMS. *Artificial Intelligence* 28:127–162.

Falkenhainer, B., and Forbus, K.D. 1991. Compositional modeling: finding the right model for the job. *Artificial Intelligence* 51:95–143.

Green, D., and Shapiro, I. 1995. *Pathologies of Rational Choice Theory*. New Haven: Yale University Press.

Keppens, J., and Shen, Q. 2000. Towards compositional modelling of ecological systems via dynamic flexible constraint satisfaction. In *Proceedings of the 14th International Workshop on Qualitative Reasoning about Physical Systems*, 74–82.

Keppens, J., and Shen, Q. 2001. On compositional modelling. *Knowledge Engineering Review* 16(2):157–200.

Legg, C.J.; Muetzelfeldt, R.I.; and Heathfield, D.N. 1995. Modelling vegetation dynamics in mediterranean ecosystems: Issues of scale. In *Proceedings of the 39th Symposium of the International Association for Vegetation Science*.

Mavrovouniotis, M.L., and Stephanopoulos, G. 1987. Reasoning with orders of magnitude and approximate relations. In *Proceedings of the National Conference on Artificial Intelligence*, 626–630.

Mavrovouniotis, M.L., and Stephanopoulos, G. 1988. Formal order-of-magnitude reasoning in process engineering. *Computer and Chemical Engineering* 12(9/10):867–881.

Mittal, S., and Falkenhainer, B. 1990. Dynamic constraint satisfaction problems. In *Proceedings of the 8th National Conference on Artificial Intelligence*, 25–32.

Murthy, S. 1988. Qualitative reasoning at multiple resolutions. In *Proceedings of the National Conference on Artificial Intelligence*, 296–300.

Nayak, P.P. 1992. Order of magnitude reasoning using logarithms. In *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning*, 201–210.

Nayak, P.P. 1993. Order of magnitude reasoning using logarithms. In *Proceedings of the International Workshop on Qualitative Reasoning about Physical Systems*.

Raiman, O. 1986. Order of magnitude reasoning. In *Proceedings of the National Conference on Artificial Intelligence*, 100–104.

Raiman, O. 1991. Order of magnitude reasoning. *Artificial Intelligence* 51:11–38.

Schiex, T.; Fargier, H.; and Verfaillie, G. 1995. Valued constraint satisfaction problems: Hard and easy problems. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, 631–637.

Travé-Massuyès, L., and Piera, N. 1989. The orders of magnitude models as qualitative algebras. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, 1261–1266.

Tsang, E., and Warwick, T. 1990. Applying genetic algorithms to constraint satisfaction optimization problems. In *Proceedings of the 9th European Conference on Artificial Intelligence*, 649–654.

Tsang, E. 1993. *Foundations of Constraint Satisfaction*. London and San Diego: Academic Press.

Tversky, A., and Thaler, R.H. 1990. Anomalies preference reversal. *Journal of Economic Perspectives* 4:201–211.