CORA Cork Open Research Archive
Cartlann Taighde Oscailte Chorcaí

Title	Conditional lexicographic orders in constraint satisfaction problems
Authors	Wallace, Richard J.;Wilson, Nic
Publication date	2009-09-01
Original Citation	Wallace, RJ, Wilson, N. (2009) 'Conditional lexicographic orders in constraint satisfaction problems'. Annals of Operations Research, 171 (1): 3-25. doi: 10.1007/s10479-008-0385-3
Type of publication	Article (peer-reviewed)
Link to publisher's version	http://link.springer.com/article/10.1007%2Fs10479-008-0385-3 - 10.1007/s10479-008-0385-3
Rights	© Springer Science+Business Media, LLC 2008. The final publication is available at http://link.springer.com/article/10.1007%2Fs10479-008-0385-3
Download date	2026-11-02 21:11:38
Item downloaded from	https://hdl.handle.net/10468/1115



UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Ann Oper Res
DOI 10.1007/s10479-008-0385-3

Conditional lexicographic orders in constraint satisfaction problems

Richard J. Wallace · Nic Wilson

© Springer Science+Business Media, LLC 2008

Abstract The lexicographically-ordered CSP ("lexicographic CSP" or "LO-CSP" for short) combines a simple representation of preferences with the feasibility constraints of ordinary CSPs. Preferences are defined by a total ordering across all assignments, such that a change in assignment to a given variable is more important than any change in assignment to any less important variable. In this paper, we show how this representation can be extended to handle conditional preferences in two ways. In the first, for each conditional preference relation, the parents have higher priority than the children in the original lexicographic ordering. In the second, the relation between parents and children need not correspond to the importance ordering of variables. In this case, by obviating the "overwhelming advantage" effect with respect to the original variables and values, the representational capacity is significantly enhanced. For problems of the first type, any of the algorithms originally devised for ordinary LO-CSPs can also be used when some of the domain orderings are dependent on assignments to "parent" variables. For problems of the second type, algorithms based on lexical orders can be used if the representation is augmented by variables and constraints that link preference orders to assignments. In addition, the branch-and-bound algorithm originally devised for ordinary LO-CSPs can be extended to handle CSPs with conditional domain orderings.

Keywords Preference · Lexicographic · Constraint programming · Constraint satisfaction problem

1 Introduction

An important contribution of artificial intelligence to the study of preferences has been the development of methods for representing and handling conditional preferences. This work

R.J. Wallace (✉) · N. Wilson
Cork Constraint Computation Center and Department of Computer Science, University College Cork, Cork, Ireland
e-mail: r.wallace@4c.ucc.ie

N. Wilson
e-mail: n.wilson@4c.ucc.ie

assumes that preference orderings are often context-dependent. Once one considers preferences in this way, many examples spring to mind. To take one such: what I prefer to eat may depend on the country I am in, especially if I am inclined to 'go native'. So in Spain I may prefer paella and tortillas, while in Germany I may prefer bratwurst and sauerkraut.

The most widely discussed representation of conditional preferences is the "CP-net", which is characterized by *ceteris paribus* conditions on preferences between the different values of an attribute Boutilier et al. (1999, 2004b). In the present work we describe an alternative representation for conditional preferences based on lexicographic orders.

Lexicographic orders have appealing computational properties; in particular, comparing alternatives is very efficient in contrast to CP-nets. Moreover, a single best solution can always be specified. Until now, however, the rigidity of the ordering, especially the inability to handle tradeoffs due to the "overwhelming advantage" of more important attributes over all less important ones (Fishburn 1974), has limited the usefulness of this form of representation.

In this paper, we introduce two forms of lexicographic ordering that are elaborated to handle conditional preferences. In one case, we show that it is possible to support conditionalities that oppose the basic importance ordering. (This is in contrast to CP-nets or to TCP-nets, an extension of CP-nets which allows importance to be defined between specific pairs of variables.) This has the surprising effect of avoiding the overwhelming advantage feature with respect to particular attribute values without giving up a lexicographic ordering of alternatives. It, therefore, has the potential of greatly extending the application of lexicographic ordering for preference representation.

In this work, conditional preferences are studied in the context of constraint satisfaction problems (CSPs). This means that outcomes are "framed" in relation to domains of values associated with distinct variables (cf. Wellman and Doyle 1991). As with CP-nets, this allows us to specify conditions of preferential independence between values in different domains, which state which variables are irrelevant to the degrees of preference of values of a given variable (Keeney and Raiffa 1993).

In earlier work, we investigated lexicographic orderings incorporated into a standard CSP representation, which we termed the *lexicographically ordered CSP* (Freuder et al. 2003). This is a special kind of soft constraint system in which a lexicographic ordering is imposed on complete assignments, based on orderings of variables and domain values. Solutions are compared first on the most important variable, and then, if they agree on that variable, they are compared on the second most important variable, and so on. This means that a good assignment for a more important variable is more significant than a good assignment for any less important variable in deciding the overall ranking of solutions. The preference ordering is assumed to be independent of any constraints that hold among these variables. The latter, therefore, restrict the alternatives given by an ideal preference ordering to those that can be realized.

Lexicographic CSPs represent problems in which preferences involve multiple objectives and attributes and where feasibility constraints impose restrictions on assignments that are actually possible. From the point of view of representation as well as computation they offer significant benefits. This is partly because, like CP-nets (cf. Boutilier et al. 2004a), they allow a radical decoupling of the preference structure from the feasibility conditions Freuder et al. (2003). This allows users to concentrate on their preferences without regard to feasibility constraints, which they may not know or understand.

When this form of lexicographic ordering is extended to *conditional lexicographic orders*, the same type of ordering holds as in ordinary LO-CSPs, but domain orderings are conditional on assignments chosen from other domains. We consider two classes of conditional LO-CSPs. In the first, conditionalities respect the priority ordering of the variables;

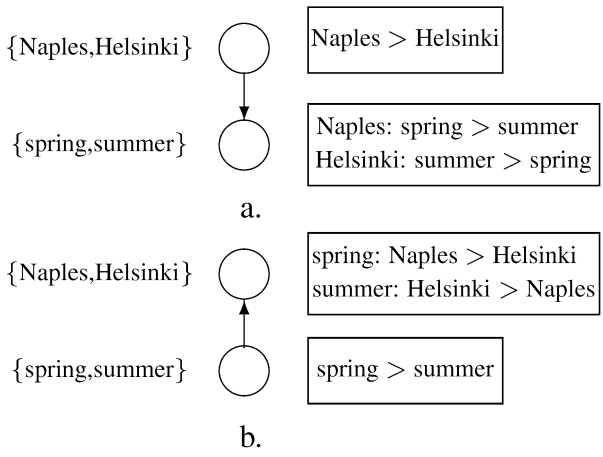


Fig. 1 Examples of conditional lexicographic preference orderings. As with CP-nets, a directed graph represents conditional preference relations among variables. Conditional preference tables are on the *right*. In both cases, the more important variable is above the less important one. In (**a**) the conditionality is consistent with relative importance; in (**b**) the two are opposed

in the second, they do not. For the first form of conditional preferences, we demonstrate an important dominance relation between conditional lexicographic orders and CP-nets, which allows us to obtain optimal solutions to combinatorial optimisation problems based on the latter using the lexicographic form of representation.

In the latter case, despite complications engendered in the representation, the basic algorithmic strategies devised for ordinary (unconditional) LO-CSPs can be extended to handle these problems. Since these complications are formal and computational, they need not affect usability or user comprehension. In fact, by removing the limitation on relative importance between parents and children, we impose fewer restrictions on the user, restrictions that might distort the preference relations he or she would otherwise express.

In this paper, we illustrate these ideas with a simple example. Consider a situation in which a customer is deciding among vacations. There are two seasons when he can travel: spring and summer. For simplicity, we consider only two locations: Naples and Helsinki. In the first scenario ("simple" conditional lexicographic ordering), location is more important than time of travel and the preferred season depends on the location chosen. This is shown in Fig. 1a, where following Boutilier et al. (1999), the conditional preferences are represented as conditional preference tables (CPTs). The preference statement Naples: spring $\succ$ summer, for example, means that if Naples is chosen, then spring is preferred to summer. The associated preference ordering over complete assignments is:

$$\langle \text{Naples, spring} \rangle \succ \langle \text{Naples, summer} \rangle \succ \langle \text{Helsinki, summer} \rangle \succ \langle \text{Helsinki, spring} \rangle$$

In the second scenario ("extended" conditional lexicographic ordering), location is again the primary feature, but the preference for location depends on the season chosen. Thus, our customer prefers Naples in the spring and Helsinki in the summer; in addition, a vacation in spring is preferred over one in the summer. In this case, it is not immediately obvious how to order alternatives such as $\langle$ Naples, spring $\rangle$ and $\langle$ Helsinki, summer $\rangle$. If we are willing to interpret this case as the alternatives being equally good as far as location is concerned, while the former scores better on the season criterion, then the preference ordering is:

$$\langle \text{Naples, spring} \rangle \succ \langle \text{Helsinki, summer} \rangle \succ \langle \text{Helsinki, spring} \rangle \succ \langle \text{Naples, summer} \rangle$$

On the other hand, an explicit ordering of tuples by the user may yield, for example,

$$\langle \text{Helsinki, summer} \rangle \succ \langle \text{Naples, spring} \rangle \succ \langle \text{Helsinki, spring} \rangle \succ \langle \text{Naples, summer} \rangle$$

which may also be acceptable. Note that in either case, choosing a specific value for the more important attribute does not force the resulting alternative to be preferred over all other alternatives having other values. Thus, in the first case, one alternative that includes the city Naples is preferred over any other alternative while the other alternative with this value is actually the least-preferred.

To summarise, lexicographically ordered CSPs have some appealing features: the simplicity of the inputs, the decisiveness of the ordering, and the fact that they support efficient solving methods. The decoupling of preferences and feasibility constraints is also quite intuitive. The present paper shows that this formulation can be extended to conditional preferences, which are natural in many situations. This enhances the expressiveness of LO-CSPs while overcoming a major limitation (the "overwhelming advantage" effect). In addition, we show that computationally these systems are still manageable, since problems are still amenable to suitable extensions of the approaches developed earlier for ordinary LO-CSPs.

At the same time, our example suggests that there are interpretative issues with the extended form of conditional lexicographic ordering. As we will argue, there is a reasonable interpretation using the positions or rankings of values, under which the first of the two orderings just presented is the only acceptable one. On the other hand, an interpretation based on direct ordering of tuples of child and parent values may allow either ordering and possibly others as well.

In the remainder of the paper, Sect. 2 defines LO-CSPs and CSPs with conditional lexicographic orders, and discusses relations with soft constraint representations. Section 3 discusses interpretations of extended conditional lexicographic orders. Section 4 discusses relations with CP-nets and TCP-nets. Section 5 considers algorithms for simple conditional LO-CSPs, Sect. 6 for extended conditional LO-CSPs. Section 7 describes the experimental set up and discusses the experimental results. Section 8 gives conclusions.

The present paper is an extension of work reported earlier in Wallace and Wilson (2006), Wallace (2006).

2 Background and definitions

In this section we recall a definition for lexicographic CSPs, and define two kinds of conditional lexicographic orders, the first where the preference degree of a variable's value can depend only on the values of more important variables, the second where this assumption is removed, i.e., the preference degree of a variable's value can depend on any other variables.

Some basic definitions for ordering relations A relation $\succeq$ is said to be *reflexive* if $\alpha \succeq \alpha$ holds for all α. It is *transitive* if $\alpha \succeq \beta$ and $\beta \succeq \gamma$ implies $\alpha \succeq \gamma$. Relation $\succeq$ is *antisymmetric* if $\alpha \succeq \beta$ and $\beta \succeq \alpha$ implies $\alpha = \beta$. It is *complete* if for all α and β, either $\alpha \succeq \beta$ or $\beta \succeq \alpha$ holds (or both). A pre-order is a reflexive and transitive relation. A partial order is an antisymmetric pre-order. A total order is a complete partial order. A total pre-order is a complete pre-order.

2.1 Lexicographic and conditional lexicographic CSPs

Definition 1 (Lexicographic CSP) A finite CSP is defined in the usual way as a triple $\langle V, D, C\rangle$, where V is a set of variables, D is a set of domains each of which is associated with a member of V, and C is a set of constraints, or relations holding between subsets of variables.

To specify a CSP as lexicographic, we introduce the following definitions. A labelling of set V is a bijection between $\{1, \ldots, |V|\}$ and V. A *lexicographic structure* L over V is a pair $\langle \lambda, \{>_X : X \in V\}\rangle$, where the second component is a family of total orders, with $>_X$ being a total order on the domain of X, and λ is a labelling of V. We write $\lambda(i)$ as X_i, so that the ordering of the variables is $X_1, \ldots, X_n$. The associated *lexicographic order* $>_L$ on (complete) assignments is defined as follows: $\alpha >_L \beta$ if and only if $\alpha \neq \beta$ and $\alpha(X_i) >_{X_i} \beta(X_i)$, where X_i is the first variable (i.e., with minimum i) such that α and β differ.

A *lexicographic CSP* is a tuple $\langle V, D, C, \lambda, \{>_X : X \in V\}\rangle$, where $\langle V, D, C\rangle$ is a finite CSP and $\langle \lambda, \{>_X : X \in V\}\rangle$ is a lexicographic structure over V.

A solution to a lexicographic CSP is the unique assignment α such that

(i) α is a satisfying assignment, that is, it is consistent with, or satisfies, all constraints in C, and
(ii) $\alpha >_L \beta$ for any other satisfying assignment β.

As is well-known, we can embed a lexicographic ordering within a weighted CSP framework:

Lexicographic CSP as a weighted CSP For each $i = 1, \ldots, n$ we define a unary weighted constraint W_i over variable X_i, given by $W_i(x) = kb^{n-i}$, where x is the kth best value in the domain of X_i and b is the largest domain size. Then for assignments α and β, the sum of weights associated to α is less than the sum associated to β if and only if $\alpha >_L \beta$.

Similar embeddings can be used for the conditional lexicographic and extended conditional lexicographic cases.

The lexicographically-ordered CSP is a special case of the "lexicographic CSP" or "lex-VCSP" as defined in Schiex et al. (1995). These authors also show that lex-VCSPs are equivalent to a kind of weighted CSP. However, because of the character of the ordering in our case, we do not need to represent preferences numerically, and we can build up partial solutions correctly without reference to numerical operations such as addition. So, while we follow (Schiex et al. 1995) in using the term "lexicographic CSP", we are designating a very special case of the class they describe, with implications both for its usefulness as a representation in the context of preferences and its ability to support efficient algorithms. For this reason, we also retain the term "lex-VCSP", using it to refer to the more general category of CSPs whose evaluations can be ordered lexicographically.

Definition 2 ((Simple) conditional lexicographic CSP) A *conditional lexicographic network* over V is defined as a tuple $K = \langle \lambda, G, \mathrm{CPT}\rangle$, where λ is a labelling of V, with $\lambda(i)$ being written X_i, and G is a directed acyclic graph on V which is compatible with λ, i.e., $(X_i, X_j) \in G$ implies $i < j$. If $(X_i, X_j) \in G$ then X_i is said to be a parent of X_j, and X_j is a child of X_i. We define U_i to be the set of parents of X_i. CPT is a function which associates a conditional preference table $\mathrm{CPT}(X_i)$ to each $X_i \in V$. As in CP-nets (Boutilier et al. 2004b), each conditional preference table $\mathrm{CPT}(X_i)$ associates a total order $>_u^{X_i}$ on the domain of X_i with each instantiation u of the parents U_i of X_i (with respect to G). (If X_i has

no parents, i.e., $U_i = \emptyset$, then we can write $>_u^{X_i}$ as just $>^{X_i}$, since this total order is then unconditional.) The associated *conditional lexicographic order* $\succ_K$ on assignments is defined as follows: $\alpha \succ_K \beta$ if and only if $\alpha \neq \beta$ and $\alpha(X_i) >_u^{X_i} \beta(X_i)$, where X_i is the first variable (i.e., with smallest i) such that $\alpha(X_i) \neq \beta(X_i)$, and $u = \alpha(U_i)$ (which also equals $\beta(U_i)$). It is easily seen that $\succ_K$ is a total order on assignments. In this definition, it is essential that the graphical structure G is compatible with the importance ordering of the variables λ, as this ensures that $\alpha(U_i) = \beta(U_i)$ in the above definition, since U_i consists only of more important variables (i.e., variables X_j with $j < i$), and α and β agree on such variables.

The example in Fig. 1a can be represented with λ labelling location as X_1 and season as X_2, graph G containing the arc from X_1 to X_2, and using the pair of conditional preference tables in Fig. 1a, including for example, the following ordering: spring $>_{\text{Naples}}^{X_2}$ summer.

2.2 Extended conditional lexicographic orders

In this section we construct an extended form of conditional lexicographic order in which the preferences regarding a variable can depend on the values of less important variables, as well as on the values of more important variables. We again assume that an importance ordering of the variables is part of the input, and also a directed graph to indicate which variables each variable depends on. However, we require stronger input information than conditional preference tables, namely the functions Q_i defined below, where Q_i is used to represent an ordering of assignments to [X_i and its parents]. (A method of generating function Q_i from a conditional preference table is described in Sect. 3.1.)

Definition 3 (Conditional lexicographic CSP with extended conditional preference orders) An *extended conditional preference network* involves a directed graph G and a labelling λ which represents an ordering $X_1, \ldots, X_n$ of the variables. It also involves, for each variable X_i, a function Q_i which assigns a number $Q_i(x, u)$ for every value x of X_i and assignment u to the parents U_i of X_i. If α is a complete assignment, we write also $Q_i(\alpha)$ for $Q_i(\alpha(X_i), \alpha(U_i))$, where e.g., $\alpha(X_i)$ is the value that α assigns to X_i. The "Q" is short for "quality". For example, if X_i represents the location of a holiday, Q_i can be considered as representing the *degree of quality* of the location, that is, how good it is (with lower values of Q_i representing better quality). $Q_i(\alpha)$ tells us about the quality of the choice $\alpha(X_i)$ of the location variable X_i. This depends not just on X_i, but on other variables (the parents of X_i); for example, how good a location is depends on the season.

The associated ordering, *the extended conditional preference order,* is then defined as follows: to compare complete assignments α and β we find the first X_i (i.e., with smallest i) such that $Q_i(\alpha)$ is not equal to $Q_i(\beta)$; if $Q_i(\alpha)$ is less than $Q_i(\beta)$, we prefer α to β; else we prefer β to α. If, on the other hand, there exists no such X_i—so that for all $i = 1, \ldots, n$, $Q_i(\alpha) = Q_i(\beta)$—then α and β are equivalent with respect to the ordering; neither is (strictly) preferred to the other. We write $\alpha \succ_E \beta$ if α is strictly preferred to β, and we write $\alpha \succeq_E \beta$ if α is either strictly preferred to β, or α and β are equivalent. So $\alpha \succ_E \beta$ holds if and only if $\beta \not\succeq_E \alpha$.

Therefore, we compare assignments α and β by first seeing how good their values of X_1 are, i.e., by comparing $Q_1(\alpha)$ and $Q_1(\beta)$, and then considering X_2, and so on.

Suppose, as in the example in Sect. 1, we are choosing a holiday destination. We consider that the quality of the location is more important than the season, so we let variable X_1 represent the city, and let X_2 be the season. We prefer spring to summer unconditionally,

which can be represented by setting, for example, $Q_2(\text{spring}) = 1$, $Q_2(\text{summer}) = 2$ (the numbers are not important apart from the relative ordering, that $Q_2(\text{spring}) < Q_2(\text{summer})$). We might set $Q_1(\text{Naples, spring}) = Q_1(\text{Helsinki, summer}) = 1$ and $Q_1(\text{Helsinki, spring}) = Q_1(\text{Naples, summer}) = 2$ to indicate that, as far as the criterion of location is concerned, Naples in the spring is equally good as Helsinki in the summer, and both are better than Naples in the summer, which is equally good as Helsinki in the spring. The uniquely most desirable alternative is then Naples in the spring. It is preferred to Helsinki in the summer since they are equally good regarding the quality of the location Q_1 (the more important criterion), but ⟨Naples, spring⟩ has a better season (the less important criterion).

Alternatively, we might represent a different view on the quality of the location by defining, for example, $Q_1(\text{Helsinki, summer}) = 1$, $Q_1(\text{Naples, spring}) = 2$, $Q_1(\text{Helsinki, spring}) = Q_1(\text{Naples, summer}) = 3$. This gives the second ordering shown in the Introduction under the assumptions of Fig. 1b.

An important difference between the approaches for the extended conditional lexicographic case and that for the simple case, is that in the simple case the conditional preference statements respect a *ceteris paribus* interpretation which is like that used in CP-nets and related formalisms. In Fig. 1a the unconditional preference for Naples over Helsinki implies that an alternative involving Naples is preferred to an alternative involving Helsinki which agrees with the first alternative on other variables; this would hold even if location were a less important variable. However, the preference for spring over summer in Fig. 1b is not interpreted in the same *ceteris paribus* manner: we may well have Helsinki in summer being preferred to Helsinki in spring. The preference for spring over summer only comes into play if two alternatives are *equally good* with regards to the more important location variable: *equal quality of location* replaces the *equality of value of the location variable* in the usual *ceteris paribus* interpretation.

It is revealing—in particular, for computational reasons—to consider another way of viewing the construction of the ordering $\succeq_E$: we are converting each assignment $\alpha = (x_1, \ldots, x_n)$ to an n-tuple of numbers $\alpha^* = (Q_1(\alpha_1), \ldots, Q_n(\alpha_n))$, i.e., $(Q_1(x_1, u_1), \ldots, Q_n(x_n, u_n))$, where u_i is the assignment α makes to U_i. The extended conditional lexicographic order $\succeq_E$ is then essentially the standard lexicographic order on these n-tuples of numbers: α is strictly preferred to β if and only if α^* is lexicographically better than β^*, and they are equivalent in the order if and only if $\alpha^* = \beta^*$. This implies that $\succeq_E$ is a total pre-order, i.e., it is reflexive, transitive and complete. It is not necessarily a total order since we can have $\alpha \succeq_E \beta \succeq_E \alpha$ for $\alpha \neq \beta$, which happens when $\alpha^* = \beta^*$.

Because the order on tuples of assignments depends only on the ordering of the values of each Q_i, each function Q_i in the definition of an extended conditional preference network could be replaced by a total pre-order $\geq_i$ on the set of assignments to $\{X_i\} \cup U_i$, where $xu \geq_i x'u'$ if and only if $Q_i(x, u) \leq Q_i(x', u')$. The values of Q_i are not important in themselves: only the relative ordering of them.

3 Derivations of Q-values for extended conditional lexicographic orders

This section considers a situation where we would like to generate an extended conditional lexicographic network (and hence its associated ordering on solutions), but instead of the Q_i functions we are given similar input as for a simple conditional lexicographic network, i.e., an importance ordering of the variables $X_1, \ldots, X_n$ (expressed by labelling λ), a DAG G and a conditional preference table function CPT. To obtain the extended conditional preference network we will need to generate, for each variable X_i, a function Q_i from this input information.

Now, a function Q_i gives stronger information than a conditional preference table CPT_i: the function Q_i orders (using a total pre-order) all tuples of the form (x, u), where x is an assignment to X_i, and u is an assignment to U_i, the parents of X_i. On the other hand, CPT_i orders, for each u, all the values of X_i using relation $>_u^{X_i}$. There is therefore a basic coherence condition between (the input) CPT_i and (the desired output) Q_i:

(a) Underlying Order Rule. For any values x, x' of X_i, and any assignment u to U_i, if $x >_u^{X_i} x'$ then $Q_i(x, u) < Q_i(x', u)$. (Recall that smaller values of Q_i indicate more preferred tuples.) Thus, the ordering of all tuples of $\{X_i\} \cup U_i$ expressed by Q_i, must be consistent with each ordering based on a specific set of parent values.

Thus, given the information in Fig. 1b, we must have $Q_1(\text{Naples, spring})$ less than $Q_1(\text{Helsinki, spring})$, since given spring, the city Naples is preferred to Helsinki in the conditional preference table.

For the case where the importance ordering is compatible with the dependency graph G, so that parents are always more important than children, suppose we choose Q_i in any way such that the Underlying Order Rule is satisfied. It is easy to see that any such choice leads to the same ordering on solutions, which is equal to that generated by the simple conditional lexicographic network $\langle \lambda, G, \text{CPT} \rangle$.

More generally, when the children can be more important than the parents, there can be many non-equivalent functions Q_i which satisfy the Underlying Order Rule, and which generate very different orderings on solutions. In the next section we consider one way of generating Q_i from CPT_i. In Sect. 3.2 we discuss some further conditions that one might make on Q_i, which are compatible with the approach of Sect. 3.1.

3.1 Q-values based on ranking of values

There is a straightforward way to derive Q-values directly from the importance ordering and the CPTs, that we refer to as the index- or rank-based interpretation. The method is to assign Q-values according to the rankings of values in their respective conditional preference orderings. Therefore we set $Q_i(x, u) = 1$ if x is the best value of X_i according to $>_u^{X_i}$, we set $Q_i(x, u) = 2$ if x is the second best value of X_i, and so on. In the example in Fig. 1b, where Naples $>_{u_1}^{X_1}$ Helsinki when it is spring and Helsinki $>_{u_2}^{X_1}$ Naples when it is summer, the ranking is 1 for Naples and 2 for Helsinki in the first case and 1 for Helsinki and 2 for Naples in the second. For spring and summer, the respective ranking is 1 and 2 unconditionally. So, we have $Q_1(\text{Naples, spring}) = Q_1(\text{Helsinki, summer}) = 1$ and $Q_1(\text{Naples, summer}) = Q_1(\text{Helsinki, spring}) = 2$, and $Q_2(\text{spring}) = 1$ and $Q_2(\text{summer}) = 2$.

When ranks are listed according to the importance ordering of the variables, one obtains an ordering on the tuples of ranks associated with each assignment that is lexicographic. In the present example, the rank tuples are: ⟨Naples, spring⟩ (1,1), ⟨Helsinki, summer⟩ (1,2), ⟨Helsinki, spring⟩ (2,1), ⟨Naples, summer⟩ (2,2). Ordering by ranks from left to right gives the ordering. This is the first ordering of choices shown in the Introduction under the assumptions of Fig. 1b.

By construction, the ranking interpretation satisfies the Underlying Order Property (a). Since it generates a unique tuple of Q_i values for each assignment, it generates a total order on complete assignments.

However, this indexing derivation involves a strong assumption, that the values of a child variable are regarded as equivalent with respect to preference when they have the same rank with respect to their respective orderings. (Thus, Helsinki in summer is treated as equivalent to Naples in spring in the example.) This doesn't require that the decision maker consider

them identical in quality; a weaker interpretation would be that when their ranks are equal one cannot decide between two tuples on the basis of these values alone.

3.2 Q-values based on orderings of child-parent tuples

As remarked at the end of Sect. 2.2, Q-values can be viewed as an ordering of the tuples formed by each value of a child together with each possible set of values of its parent variables. In our running example, this would simply consist of ordering the two sets of (two) tuples associated with each parent value. In doing this, the user would assign numbers to each tuple consistent with the ordering; these would form the Q-values for the child variable.

A potential issue with the tuple-order derivation stems from the fact that it involves an ordering that goes beyond those in the basic definition of a LO-CSP. In this case, the tuple ordering could come into conflict with the other evidence regarding importance and dependency.

To illustrate the problem, we consider several possible orderings using our running example:

(i) ⟨Naples, summer⟩ > ⟨Helsinki, spring⟩ > ⟨Naples, spring⟩ > ⟨Helsinki, summer⟩

(ii) ⟨Helsinki, summer⟩ > ⟨Naples, summer⟩ > ⟨Naples, spring⟩ > ⟨Helsinki, spring⟩

(iii) ⟨Naples, spring⟩ > ⟨Helsinki, spring⟩ > ⟨Helsinki, summer⟩ > ⟨Naples, summer⟩

(iv) ⟨Naples, spring⟩ > ⟨Helsinki, summer⟩ > ⟨Helsinki, spring⟩ > ⟨Naples, summer⟩

(v) ⟨Helsinki, summer⟩ > ⟨Naples, spring⟩ > ⟨Helsinki, spring⟩ > ⟨Naples, summer⟩

(vi) ⟨Helsinki, summer⟩ > ⟨Naples, spring⟩ > ⟨Naples, summer⟩ > ⟨Helsinki, spring⟩

In each case, if the ordering is deemed acceptable, we assign increasing Q-values, e.g. 1, 2, 3 and 4, respectively, to successive tuples. This contrasts with the assignment of Q-values from the ranks associated with entries in the CPTs, which would only involve two distinct values. (In our running example, this completely orders the outcomes, but this is not the case in general.) But before doing this, we need to decide on the acceptability of the tuple ordering, given other preference information, in particular, the CPTs.

In fact, the first tuple ordering can be dismissed at once because it contradicts the conditional preference orders shown in Fig. 1b; it disobeys Rule (a) above. The second tuple ordering is not only consistent with season being more important than city, which might seem to conflict with the original importance ordering, but it also favours summer over spring, in contrast with the specified preference ordering for season. The third tuple ordering favours spring over summer, but it is still compatible with season being more important than city. Orderings (iv)–(vi) are compatible with Rule (a) and the importance ordering, although the order of assignments is different in each case.

These examples suggest two things: (i) if Q-values are derived directly from tuple orders, further conditions must be specified to ensure coherence among different sources of information, in particular between the tuple orders and the CPTs, (ii) when Q-values are derived in this fashion, the ordering that they are derived from need not be unique, in contrast to the rank-based interpretation.

One way of addressing these issues is by stipulating further "coherence rules" that limit the kinds of tuple orderings that are deemed acceptable. The underlying rationale for these rules is that these orderings should be compatible with the other inputs. The further rules make use of the notion of rank of a value in a total order, introduced in Sect. 3.1. The rank of x in total order $>$ is equal to 1 if x is the best value according to $>$, and equal to 2 if x is the second best value, and so on.

(b) Equal Child Values Rule. For any value x of X_i, and any assignments u, u' to U_i, if the rank of x in the total order $>_u^{X_i}$ is less than the rank in $>_{u'}^{X_i}$, then it should not be the case that $Q_i(x, u) > Q_i(x, u')$. In other words, when a child value occurs in different places in the orderings of the domain of X_i under two sets of parent values, the Q-values of the two corresponding child-parent tuples should be consistent with the rankings of the child value in the respective orders. (This rule is violated by orderings (i), (ii), and (iii) above.)

(c) Incomparability Rule. For any values x, x' of X_i, and any assignments u, u' to U_i, if the rank of x in $>_u^{X_i}$ is the same as the rank of x' in $>_{u'}^{X_i}$, and for any assignment z to the parents of U_i, we have $u >_z^{U_i} u'$, then $Q_i(x, u) \leq Q_i(x', u')$. (For simplicity, we are restricting ourselves to the case where $|U_i| = 1$, although the specification can be extended.) The basic thrust of this rule is that, when two values occur in the same location in the orderings of the domain of X_i, they are considered incomparable and are excluded from consideration in deciding whether the ordering under consideration is acceptable. For example, in comparing ⟨Naples, spring⟩ and ⟨Helsinki, summer⟩, the ordering cannot be justified on the basis of the difference in cities. In this case, we reject the ordering if information about preferences between parent values contradicts the tuple ordering unconditionally. (This rule is violated by orderings (i), (ii), (v), and (vi) above.)

A further rule is considered in Wallace (2006); since it is subsumed by Rule (b), the same results obtain when the present set of rules is applied as in the earlier paper.

It is easily shown that each of the rules is consistent with the rank-based method described in Sect. 3.1 cf. Wallace (2006). Rules (b) and (c) are closely related to a rank-based interpretation of Q-values, which may limit their acceptability given the assumptions required by the latter. Nonetheless, they have a certain force, especially in extreme cases.

Rule (b) pertains to cases where there are two child domain orderings, and in one instance a child value is judged to be better than in another *with respect to the same set of other child values*. It says that in such situations, according to the ordinal style of evaluation, the tuple of child-and-parent values (X_i, U_i values) in the first instance should be preferred to those in the other instance, *ceteris paribus*. Consider, for example, a case where in one of the two tuples the child value in question is top-ranked (e.g. Naples in the spring). This means that in this context it is taken to be the best in the set, while in the other context it is only the kth best (e.g. Naples in summer). In an extreme case, ignoring this rule would mean that a tuple in which the child value is the worst value for that variable under one set of conditions could be preferred to a tuple in which the same child value is the best value for that variable.

Rule (c) is related to Rule (b), most obviously in cases with identical child values that are in the same position in their respective rankings. In this case, Rule (b) does not apply, and because both the values and the rankings are the same, it seems reasonable to refrain from choosing between the tuples on the basis of these values. In that case, acceptability can be determined by the values of other, less important variables, if their domains are ordered unconditionally. Without Rule (c), however, such conformance is not guaranteed. The same

argument holds, though perhaps with less force, when the child values are not identical. For our vacation example this rule prevents the following,

$$\langle \text{Helsinki, summer} \rangle > \langle \text{Naples, spring} \rangle,$$

since in this case season is the only basis for comparison and spring $>$ summer unconditionally.

In this case, in fact, use of the three rules leads to a unique outcome ordering:

$$\langle \text{Naples, spring} \rangle \succ \langle \text{Helsinki, summer} \rangle \succ \langle \text{Helsinki, spring} \rangle \succ \langle \text{Naples, summer} \rangle$$

However, this is not true generally (Wallace 2006). Hence, unless one requires that the tuple ordering conform completely to the ranking of domain values, which could be considered a stronger and less justifiable rule, derivations of extended conditional lexicographic orderings from tuple-orderings can allow more than one ordering of assignments. This situation does not seem unreasonable, and all such tuple orderings result in an extended conditional lexicographic order, by virtue of the manner of constructing Q-values.

4 Conditional lexicographic CSPs and CP-nets

In recent years, the most widely discussed method for representing conditional preferences within the AI community has been the conditional preference network with *ceteris paribus* assumptions, or CP-net (Boutilier et al. 1999, 2004b). A more recent variant, the TCP-net (Brafman and Domshlak 2002), Boutilier et al. (2006), includes elaborations to handle relations of importance between the features of user-selections (this relates to, but is weaker than, the ranking of variables in lexicographic CSPs, Wilson 2004b); another variant, CP-theories (Wilson 2004b), generalises CP-nets, TCP-nets and lexicographic orders.

CP-net structures are based on assignments of values to variables, or "features". The conditional preferences related to a variable X_i are encoded in a "conditional preference table" (CPT) (see Sect. 2), examples of which are shown in Fig. 1. TCP-nets also encode a form of conditional importance relations between variables (one could also extend lexicographic CSPs to handle conditional importance cf. Wilson 2006).

A critical feature of (T)CP-nets is that preferences are only defined under "*ceteris paribus*" conditions. If, for example, features A and B each have two values, a_1, a_2 and b_1, b_2, respectively, and $a_1 >_{X_A} a_2$ and $b_1 >_{X_B} b_2$, then we can deduce from *ceteris paribus* assumptions that $a_1b_1 >_N a_2b_1$ and $a_2b_1 >_N a_2b_2$, etc, but we cannot order a_1b_2 and a_2b_1 on this basis. As a result of this feature, preference orders can be established on the basis of "flipping sequences"; e.g., the sequence of two "flips" $a_1b_1 \succ a_2b_1 \succ a_2b_2$ enables us to deduce the preference $a_1b_1 \succ a_2b_2$. This is still true of TCP-nets, although in this case adjacent outcomes in a sequence can be separated by a "double flip" of two variables.

An acyclic CP-net over variables $V = \{X_1, \ldots, X_n\}$ is a directed acyclic graph G over V whose nodes are annotated with conditional preference tables $\text{CPT}(X_i)$ for each $X_i \in V$. Associated with an acyclic CP-net is a partial order $\succ_N$ on complete assignments. $\alpha \succ_N \beta$ holds if and only if there exists a sequence of worsening flips from α to β. There is a *worsening flip* from α to γ if α and γ differ on exactly one variable X_i, and $\alpha(X_i) >_u^{X_i} \gamma(X_i)$, where $u = \alpha(U_i) = \gamma(U_i)$ and U_i is the set of parents of X_i with respect to G.

It was shown in Wilson (2004a) that, except in some trivial cases, the order on assignments generated by a CP-net, or by a TCP-net, is never a lexicographic order. The reason for this is that flipping sequences require that consecutive elements in the ordering differ

by at most one (CP-nets) or two (TCP-nets) elements. However, consecutive elements in a lexicographic ordering can differ by up to $|V|$ elements.

Perhaps the most important implication of these differences is that, while determining whether solution α is preferred to solution β is easy for lexicographic orderings, since it is based on successive comparisons of values of a variable, this is much more problematic with (T)CP-nets, since it depends on finding flipping sequences that transform one alternative into another. Although there are special cases where this problem is polynomial Domshlak and Brafman (2002), Boutilier et al. (2004b), it appears, in general, to be an extremely hard problem Goldsmith et al. (2005). On the other hand, CP-nets allow a weaker form of comparison, which can sometimes be used to show that for two solutions α and β, the preference of the latter over the former is *not* entailed by the CP-net structure. If the CP-net is acyclic, this comparison can be carried out in low order polynomial time (Boutilier et al. 2004b).

In considering the algorithmics of CP-nets, the emphasis has been on cases where the set of dependencies forms an acyclic graph. In particular, algorithms developed for solving constrained optimisation problems are based on this assumption (Boutilier et al. 2004a; Brafman and Domshlak 2002). In many cases, this seems natural, but as we have seen, when importance relations are mixed with conditional preference relations it is sometimes reasonable to consider cases where the two do not always correspond. As we will see, for conditional lexicographic orderings this does not appear to be as important as for CP-nets, since the basic algorithms can be extended, in some cases without marked effects on performance.

For acyclic networks, a CP-net ordering can be extended to a conditional lexicographic ordering:[1] we choose a conditional lexicographic network with the same conditional preference tables and the same directed acyclic graph G as the CP-net, ordering the variables in any way compatible with the graphical structure G of the CP-net. (Similarly, the preference ordering corresponding to an acyclic conditional preference theory (Wilson 2004a) can be extended to a conditional lexicographic ordering.)

Proposition 1 *Let N be a CP-net with associated DAG G and conditional preference table function* CPT. *Let λ be any labelling of V compatible with G. Define conditional lexicographic order $K_{N,\lambda}$ to be the tuple $\langle \lambda, G, \text{CPT} \rangle$. Then, relation $\succ_{K_{N,\lambda}}$ contains relation $\succ_N$, i.e., $\alpha \succ_N \beta$ implies $\alpha \succ_{K_{N,\lambda}} \beta$.*

The idea is simple: clearly, if one can reach β from α by a worsening flip in the CP-net, then α is better according to the conditional lexicographic order. If α is better than β according to the CP-net, then, by definition, β can be reached from α by a sequence of worsening flips, and so, α is better than β according to the conditional lexicographic ordering, by transitivity of the latter.

Proof Suppose complete assignment δ can be reached by a worsening flip from complete assignment γ. Then there exists variable X_i such that for all $j \neq i$, $\gamma(X_j) = \delta(X_j)$, and $\gamma(X_i) >_u^{X_i} \delta(X_i)$, where $u = \gamma(U_i) = \delta(U_i)$. Therefore, $\gamma \succ_{K_{N,\lambda}} \delta$ holds, because γ and δ agree on variables $X_1, \ldots, X_{i-1}$. Now suppose that $\alpha \succ_N \beta$. By definition, there exists a worsening flipping sequence $\alpha_1, \ldots, \alpha_k$, with $\alpha = \alpha_1$ and $\alpha_k = \beta$, and, for $j = 1, \ldots, k-1$,

[1] Similar techniques are used in Boutilier et al. (2004b), for ordering queries, and proving consistency of an acyclic CP-net.

assignment α_{j+1} being a worsening flip from α_j. So, by the above argument, $\alpha_j \succ_{K_{N,\lambda}} \alpha_{j+1}$. Hence, by transitivity of $\succ_{K_{N,\lambda}}$, we have $\alpha \succ_{K_{N,\lambda}} \beta$, as required. □

Suppose we want to generate a solution α to a CSP on V which is maximal with respect to CP-net partial order $\succ_N$, i.e., it is such that there does not exist solution β with $\beta \succ_N \alpha$. One way of generating such a solution is to choose any labelling λ compatible with the DAG G associated with N, and find the solution α which is maximal with respect to the conditional lexicographic order $\succ_{K_{N,\lambda}}$. This α will be maximal also with respect to $\succ_N$ since, if β were a solution such that $\beta \succ_N \alpha$, then, by the last proposition, $\beta \succ_{K_{N,\lambda}} \alpha$, contradicting the $\succ_{K_{N,\lambda}}$-maximality of α. This means that if one wants to find a solution of a set of constraints C which is optimal with respect to a CP-net, one can generate an associated conditional preference order, as described above, and find the optimal solution with respect to this. The above result implies that this solution will also be optimal with respect to the CP-net. Hence, *constrained optimisation algorithms for conditional lexicographic orders can be used for finding a single optimal solution of a constrained optimisation problem for an acyclic CP-net.*

5 Constrained optimisation algorithms for conditional LO-CSPs

5.1 Methods for solving ordinary LO-CSPs

In earlier work, we showed that this form of lexicographic representation of preferences for CSPs offers wide scope for developing optimisation algorithms (see Freuder et al. 2003, 2007 for detailed descriptions). The most successful algorithms were,

1. *Lexical search*: ordinary CSP algorithms that follow the importance ordering of the variables and the preference ordering for each domain. These work very well when problems are not too strongly constrained, but are inefficient for problems in the critical complexity region.
2. *Branch and bound* algorithm: This is intended for problems in the critical complexity region. It is assumed that this algorithm will be used with a variable ordering heuristic that is non-lexical, i.e. one that does *not* follow the importance ordering; otherwise, there is no need for branching and bounding.
3. *"Staged lexical"* algorithm: This is a specialized iterative algorithm in which, on the kth restart, the kth variable in the importance ordering is instantiated according to the preference order on its domain, while typical variable ordering heuristics such as minimum domain size are used to select the remaining variables. This algorithm works almost as well as branch and bound for problems in the phase transition region and tends to be more effective than the latter as the number of solutions increases.

For the branch and bound procedure, the cost function (based on the representation in terms of weighted constraints; cf. the discussion of embeddings in Sect. 2) gives large values for any but very small problems; however we do not need to calculate it directly. Instead, we simply compare successive values following the importance ordering of the variables until we encounter a difference. Suppose that variable X_k is the variable currently being considered for instantiation and is the kth most important variable in the ordering. To evaluate the current partial solution, we start from the first variable X_1 with respect to the importance ordering. If X_1 has been assigned, we check this against its instantiation in the best assignment found so far; if it does not yet have an assignment, we check the best remaining value in its

Fig. 2 Pseudocode for staged lexical algorithm with LO-CSP

```
k = 0
while k ≤ n
  assign optimal values to variables 1 to k − 1 and propagate
  level = k
  while remaining-variables     //search for next solution
    if level == k
      select kth variable in importance order
    else
      select next-variable according to some heuristic
    while values remain and viable assignment not found
      if level == k
        select value according to preference ordering
      else
        select value according to some heuristic
    if all assignments failed
      backtrack and decrement level
    else
      level = level + 1
      if level > n
        save assignment made at level k     //this is optimal
  k = k + 1
```

domain against the best assignment. If this favours the best assignment found so far, then search can back up; if it favours the other value, we know that search cannot be bounded at this point, so we stop checking. (In either case, no further combination of cost differences can match the present difference by virtue of the properties of lexicographic ordering.) Otherwise, if they are equal on X_1 we perform a similar check on X_2 (and so on). (In addition to the references cited above, more detail on bounding conditions is given in Sect. 6.3.)

In staged lexical search, search is done repeatedly, in each case until the first solution is found, and for each repetition, or stage, one more variable is chosen according to the importance ordering, beginning with X_1 at stage 1. Values of the current variable X_k are chosen in best-first order. After the kth stage, when we have found a feasible solution, we know that the assignment for X_k is optimal, so we retain it for the remaining stages. Pseudocode for this procedure in shown in Fig. 2. Although developed independently, this algorithm is, in fact, a special case of preference-based search (Junker 2002), where the criteria on which search is based form a total order.

5.2 Algorithms for simple conditional lexicographic CSPs

When the parent-child order is compatible with the importance order of the variables, any of our methods for constrained optimisation can be used to return a solution that is optimal for the conditional LO-CSP. In particular, the staged lexical algorithm can be applied in exactly the same way as before to the conditional lexicographic case, since at stage k we know the ordering of the values of X_k, as its parents have already been instantiated. For branch and bound, if a child is chosen for instantiation before its parents, bounding can be done provided the ordering of the parent values can be established. This is because for any ancestor X_i of the child variable, either the ordering is the same on its domain for the candidate and current best assignment, or its ancestor is different, in which case this latter difference overrides the values of X_i by virtue of the importance ordering.

6 Search algorithms for extended conditional lexicographic orders

By adding extra variables and constraints (doubling the number of variables), an extended conditional lexicographic order can be related to an ordinary lexicographic order; this enables one to generalise the algorithms for lexicographic orders to the conditional case.

Auxiliary variables representation Let R_i be the set of all values taken by Q_i, i.e., the set of numbers $Q_i(x,u)$ over all values x of X_i and all assignments u to the parents U_i of X_i. For each variable X_i, create a new variable Y_i with domain R_i. Variable Y_i can be considered as telling us how good the value of X_i is. We create a constraint with scope $U_i \cup \{X_i, Y_i\}$ consisting of all tuples uxq with $Q_i(x,u) = q$, where x is a value of X_i, u is an assignment to U_i and $q \in R_i$. Let B be the set of these extra constraints. Let $V' = V \cup \{Y_1, \ldots, Y_n\}$. Each assignment α to V clearly extends uniquely to an assignment α' to V' satisfying these extra constraints B: we define, for each $i = 1, \ldots, n$, $\alpha'(X_i) = \alpha(X_i)$ and $\alpha'(Y_i) = Q_i(\alpha)$. Hence α' is essentially α extended with α^* (as defined in Sect. 2).

There is a natural lexical order on assignments to V' defined by variables $Y_1, \ldots, Y_n$ in that order of importance, and where variables $X_1, \ldots, X_n$ are all of less importance. Assignments to V' which agree on all variables Y_i—so differ only on X_i variables—are considered to be equal in the lexical order. $\alpha \succ_E \beta$ holds if and only if α' is lexically better than β' according to this lexical order (which is if and only if α^* is lexically better than β^*). In particular, α is an optimal solution of C with respect to the extended conditional lexicographic network if and only if α' is a lexically optimal solution of constraints $C \cup B$. Therefore, the strategies mentioned earlier for finding optimal solutions with respect to a lexical order can be elaborated to produce algorithms for finding optimal solutions for extended conditional lexicographic orders, as we describe in the following sections.

6.1 Extending lexical search

We can adapt lexical search for extended conditional lexicographic optimisation by using auxiliary variables to represent Q-values and adding constraints to represent acceptable parent-child assignments given a particular value for the child. The key idea is that the Q-values can serve as the basis for a lexically-ordered search. In other words, search can be done in a way that is lexicographic on the Q-values rather than on the decision variables themselves, even though the domains of the latter have no a priori preference order.

Recall that R_i is the set of all values taken by Q_i. For any $q \in R_i$, we define constraint c_i^q on variables $U_i \cup \{X_i\}$ to be the constraint $Q_i = q$, i.e., xu is a tuple in c_i^q if and only if $Q_i(x,u) = q$.

The search tree for the lexical search can be defined as follows: A node N at level j, for $j = 0, \ldots, n$ has an associated set of constraints C^N of the form $C \cup \{c_1^{q_1}, \ldots, c_j^{q_j}\}$, where for each $i \leq j$, q_i is an element of R_i. The initial node, which is the root node of the search tree, is at level 0. A node at level n is said to be a *complete node*; the other nodes are said to be *partial nodes*. At each partial node we will need to maintain some form of partial consistency. If we deduce that the associated set of constraints C^N is inconsistent then we can backtrack at this point. Otherwise, we branch on constraints c_{j+1}^q, for $q \in R_{j+1}$; that is, for each $q \in R_{j+1}$ we generate a child node of N with associated set of constraints $C^N \cup \{c_{j+1}^q\}$. The child corresponding to the smallest element q of R_{j+1} is explored first, in a depth-first manner. When we reach a complete node N, we determine if C^N has a solution; if it does, we return the solution and stop; otherwise we backtrack. This algorithm will return an optimal solution, given that the initial set of constraints C has a solution.

If the Q_i's are one-to-one functions, each constraint of the form $c_i^{q_i}$ just contains a single tuple which is an instantiation of X_i and the parents of X_i. The algorithm then behaves in a fairly similar way to standard lexical search.

On the other hand, if the Q_i's have many 'ties' so that added constraints of the form c_i^q include several tuples, then we may find that, e.g., maintaining arc consistency is not sufficiently strong to prune the search effectively, since we are not directly instantiating the variables X_i. For example, it could happen that an initial constraint $c_1^{q_1}$ is inconsistent with constraints C, but we might only discover this at complete nodes, when we have generated values of all the other q_i, and we test the consistency of the associated set of constraints $C \cup \{c_1^{q_1}, \ldots, c_n^{q_n}\}$. This will tend to make the algorithm extremely slow in such situations. For this reason, it is natural to consider the use of stronger forms of consistency checking at each node. In particular, we can use a search to check global consistency at a node; this then becomes an extended staged lexical algorithm, as described in the next section.

6.2 Extending the staged lexical algorithm

Unlike the lexical search algorithm, the staged lexical algorithm can be done without backtracking over Y_i variables. These are instantiated in order over successive stages, in each stage before any X_i variables are instantiated. The latter can be instantiated using any heuristic ordering.

Let C_0 be the original set of constraints C, which we assume to be satisfiable. Let α be an optimal solution, and let α^* be the corresponding n-tuple of numbers, as defined in Sect. 2. The fact that C_0 is satisfiable implies that there exists some $q \in R_1$ such that $C_0 \cup \{c_1^q\}$ is satisfiable, since R_1 includes all possible values of Q_1. We find minimal value $q_1 \in R_1$ such that $C_0 \cup \{c_1^{q_1}\}$ is satisfiable. (By definition of α^*, we have $q_1 = \alpha^*(X_1)$, since q_1 is the best feasible value of Q_1.) This can be found by starting with the lowest (i.e., best) value in R_1 and continuing until we find q_1 with $C_0 \cup \{c_1^{q_1}\}$ satisfiable. The checks of satisfiability can be performed by a search using a dynamic variable ordering (they don't need to instantiate variables in order of importance).

We then add this constraint $c_1^{q_1}$ to the set of constraints, setting $C_1 = C_0 \cup \{c_1^{q_1}\}$ (and we will not backtrack over this decision). (With the auxiliary variable representation this amounts to setting $Y_1 = q_1$.) We move on to optimising Q_2: we find minimal value $q_2 \in R_2$ such that $C_1 \cup \{c_2^{q_2}\}$ is satisfiable. We set $C_2 = C_1 \cup \{c_2^{q_2}\}$. We continue this until we have generated minimal $q_n \in R_n$ such that $C_{n-1} \cup \{c_n^{q_n}\}$ is satisfiable; we let $C_n = C_{n-1} \cup \{c_n^{q_n}\}$, which, by construction, is satisfiable.

It is easy to see that any optimal solution α is a solution of C_n (or else α would be worse than a solution of C_n). Also, if β is any other solution of C_n then α and β have exactly the same Q_i-values, so β is also optimal. This leads to the following result which shows that a solution of the set of constraints C_n (in particular, the one found when checking that C_n is satisfiable) is an optimal solution of the constraints C.

Proposition 2 *With the above notation, complete assignment α is an optimal solution of C if and only if α is a solution of the set of constraints C_n.*

Proof Consider $i \in \{1, \ldots, n\}$. By definition, complete assignment β satisfies the constraint $c_i^{q_i}$ if and only if $Q_i(\beta) = q_i$, which is if and only if $\beta^*(X_i) = q_i$. Therefore β is a solution of C_n if and only if β satisfies the input constraints C_0, and for each X_i, $\beta^*(X_i) = q_i$.

Let α be an optimal solution. We claim that for all i, $\alpha^*(X_i) = q_i$. Suppose otherwise, and let j be the smallest i with $\alpha^*(X_i) \neq q_i$. Let $q_i' = \alpha^*(X_j)$. Then α is a solution of C_{j-1},

since for all $i < j$, $\alpha^*(X_i) = q_i$; and α satisfies constraint $c_j^{q'_j}$, so α satisfies $C_{j-1} \cup \{c_j^{q'_j}\}$. By minimality of q_j, we have $q_j \leq q'_j$ and so $q_j < q'_j$. Let β be any solution of C_n. Then $\beta^*(X_i) = \alpha^*(X_i)$ for all $i < j$ and $\beta^*(X_j) = q_j < q'_j = \alpha^*(X_j)$. Hence β is a better solution than α, contradicting the optimality of α.

Putting these parts together, we have that any optimal solution α satisfies $\alpha^*(X_i) = q_i$ for all $i = 1, \ldots, n$, and hence satisfies C_n. Conversely, any solution β of C_n satisfies the input constraints C_0 and is such that for all i, $\beta^*(X_i) = q_i$, and so $\beta^* = \alpha^*$ for any optimal solution α, showing that β is also optimal. □

6.3 Branch and bound algorithm

Like the lexically-based search algorithms, this variation of branch-and-bound (Fig. 3) relies on the fact that for lexicographic orderings, value orderings can be indexed by Q-values. This means that bounds can be checked in terms of Q-values, thereby comparing a candidate assignment with previous assignments even when the preference ordering for the past assignment is different from the present ordering.

```
conditional-bnb (partial-solution, remaining-variables)
  if remaining-variables ≡ nil
    save partial-solution as new best-solution
    and continue    //backtrack
  else
    select next-variable and remove from remaining-variables
    for each value in its ordered domain
      if instantiating next-variable with this value gives arc consistent problem
      and
      bounds-check(next-variable, next-value) returns true //under bound
        conditional-bnb (new-partial-solution, remaining-variables)
    continue    //backtrack

bounds-check (candidate-var, candidate-value)
  while variables remain to be compared
    select next-variable in order
    get value next-best for this variable from current best-solution
    if next-variable == candidate-var
      curr-assign = candidate-value
    else if next-variable is instantiated
      curr-assign = current assignment of next-variable
        //perform comparisons
    if next-variable ∉ any child-set
      compare curr-assign or best value in current-domain with next-best
    else if domain of next-variable can be ordered
      compare curr-assign or best value in current-domain with next-best
    else if candidate-var is the only uninstantiated parent of next-variable
      get domain-order associated with parent values
      compare curr-assign or best value in current-domain with next-best
    else
      set comparison to succeeded and bound to not-exceeded
    if comparison has succeeded break
  if comparison succeeded and bound was exceeded
    return false
  else
    return true
```

Fig. 3 Branch and bound pseudocode for CSP with extended lexicographic ordering

Regardless of the search order, bounds testing always proceeds according to the priority ordering of the variables (until the current variable is reached), and the decision to bound depends on the first difference found between the current partial assignment and the best solution α. Note that for the latter all $Q_i(x, u)$ are known.

Testing is restricted to the following conditions:

- Variable k has no parents, in which case either its current assignment or the best assignment available can be used for comparing with the current best assignment for this variable (in this case, the $Q_k(x, u)$ are independent of other assignments).
- The ordering for variable k has been specified (because its parents have assignments), in which case the $Q_k(x, u)$ are known.
- Variable k has one uninstantiated parent, which is the current variable, in which case the $Q_k(x, u)$ can be determined for a candidate value of variable k.

Bounding is not done unless these conditions are met for all comparisons up to the first difference found. Then, if the first difference found favours the current best solution, since this is also the first difference in the lexicographic order, no extension of the current assignment can produce a solution β such that $\beta \succ_E \alpha$. (If there is no such difference, then, of course, we cannot bound search at this point.) This guarantees the correctness of the present algorithm. (A minor point. In the actual code, bounds checking is avoided until a first solution has been found.)

The present branch and bound algorithm has one difference from that used for CSPs with simple conditional lexicographic orderings. In the latter case, comparisons can also be made during bounds testing if the domain of each remaining uninstantiated parent of the variable being compared ("next-variable" in Fig. 3) can be ordered. In this case, a comparison is made using an ordering for next-variable based on the best possible parent values. Because the parents will have been checked already, their best values must have been equal to the corresponding values of best-solution, since a bounding decision could not be made. In this case, for the variable currently under consideration the preference ordering over the domain is the same for the best solution and the current assignment. For extended conditional lexicographic orders, where the parent is not necessarily more important than the child, this implication does not always hold.

7 Experimental tests

We present results of experimental tests with random binary CSPs and with problems based on a real-world configuration problem to show comparative performance of the different algorithms described above. Since in previous work (Freuder et al. 2003, 2007) a MAC-based algorithm proved to be much more effective than forward checking, the former is used in all tests reported here. Algorithms were coded in lisp and run using Xlisp on a Dell Work Station PWS 330 running at 1800 MHz. For each condition, solutions were compared for the different algorithms to verify that the implementations were correct.

7.1 Problem generator

For these tests, CSPs with conditional lexicographic orders were generated with a program written by the first author. This program starts with an existing CSP and transforms it into a conditional LO-CSP by selecting variables for conditional preferences and building a CPT for each relation. Each CPT is built by generating all possible tuples of parent values (ordered by descending importance of the parent variables), and for each parent tuple, and for

each child, randomly re-ordering the domain of child values. (Re-ordering is done independently for each parent tuple and child.) Q-values are derived from successive positions of successive domain elements within an ordering, so they do not have to be generated explicitly. (In other words, the generator (and solvers) implicitly use the rank-based interpretation of Sect. 3.1, although, of course, they could be generalised.)

Before creating conditional lexicographic orderings for a set of problems, the user specifies the following parameters: (i) number of preference relations, (ii) maximum number of parents per relation, (iii) maximum number of children per relation, (iv) maximum number of attempts to make a relation with p parents and c children, since at some point in generation it may not be possible to do this under the given restrictions. (If this number is ever exceeded, the program writes a message to standard-output, but continues with problem generation.) In addition, the following restrictions are made during generation:

1. A child-variable only appears as such in one preference relation (otherwise the CPT is ill-defined).
2. The graph of conditional relations is directed-acyclic, so there is no *directed* path from a node back to itself.
3. A variable occurs in no more than one single-parent relation. This restriction was made to prevent selection from undermining the maximum-child specification since k singleton-parent relations involving the same parent variable are indistinguishable from a single relation with one variable and k children.

There are two further restrictions that the user can specify optionally:

1. That parent-child relations always correspond to the priority ordering of the variables. (This specifies that the conditional LO-CSP is of the simpler type.)
2. That the parents and children in a relation do not have parents in common. (This option was not used in the experiments reported here.)

An example of output from the generator is shown in Fig. 4. In this case, there is one relation, where variable 1, with domain {0,1,2,3} is the parent, and variables 3 and 4, with domains {0,1,2} and {0,1,2 3} are the children. (The parent-child relation is indicated on the line labelled "conpref", which also tags the relation (here with a 1) for bookkeeping purposes.)

For problems used in the present tests, there was a maximum of two parents and two children per relation. The number of relations per problem was set to be 3, 7 or 11 in different experiments. With 7 relations, 70–80% of the variables were included in at least one conditional relation (i.e. were either parents or children in at least one relation); with 11 it was approximately 100%.

```
(TITLE (GRAPH-1 CONFIGTEST3.1))
(PARAMS NUMVARS 4 DMAX 4 TYPE GAC-CONFIG)
(PARAMS-PREF NUM-RELS 1 MAX-PARENTS 2 MAX-CHILDREN 2)
(conpref 1 (1) (3 4))
(CPT 1 (0)((0 2 1) (0 1 2 3))
       (1)((2 1 0) (2 3 0 1))
       (2)((1 2 0) (1 3 2 0))
       (3)((0 2 1) (3 0 2 1)))
```

Fig. 4 Output from generator of conditional relations for LO-CSPs

7.2 Results with random CSPs

Performance comparisons are given in Table 1. (Note that the number of values per domain is large in comparison with problems typically considered in this area and that there are numerous hard constraints. In addition, since the same Q-values were used for all domains, these constitute particularly difficult problems for algorithms like the staged lexical method where constraint size is related to number of different Q-values (cf. Sect. 6.1).) There are some striking differences due to problem difficulty and to the character of the conditional lexicographic ordering, and no single algorithm is superior overall.

Results for simple conditional lexicographic orderings were quite similar to those found for similar problems for the unconditional case. In addition, increasing the number of conditional relations had very little effect on search efficiency. For hard problems of this type, branch and bound and staged lexical were comparable, and both were better than straight lexical as domain size increased. For easy problems, either of the lexically-based search strategies outperformed branch and bound.

For the extended lexicographic orderings, performance differences depended on the number of constraints and number of conditional preference relations, as well as the heuristic used to order decision variables. For hard problems, the difference was decisive with branch and bound outperforming the staged lexical search by about an order of magnitude. For easy problems, staged lexical search was sometimes more efficient than branch and bound. This depended on the search order heuristic as well as the number of conditional preference relations (Table 1).

7.3 Results with configuration problems

These tests are based on a configuration problem obtained from the Configuration Benchmarks Library maintained by the Computational Logic and Algorithms Group at the University of Copenhagen (ESVS Benchmark, #7[2]). The original problem has 26 variables (with domain sizes ranging from two to 61) and 11 constraints with a maximum arity of five.

In constructing preference problems, we first discarded six variables that were disconnected. (They were not of interest in the present context since the most preferred value would always be available.) Then the remaining 20 variables were labelled lexically (following the order of listing in the source file), and 100 LO-CSPs were constructed with different lexical orders on these variables by renaming the variables according to random permutations of the original labels. (This particular strategy allowed us to continue using the lexical labels as indicators of importance in the solver code.) Doing this, we were able to test our algorithms over a large sample of possible importance orderings with respect to configuration components. This also served to test the ease of finding optimal solutions with each method when variables with very different domain sizes and constraint relations were either lower or higher in the importance ordering. Finally, the problem generator described above was used to create problems with conditional preference relations, using the same parameter values as those used with the random CSPs. For each algorithm, the solver code was the original code updated with code for generalised arc consistency; hence the algorithm employed a generalised form of MAC3.

The results of these experiments are shown in Table 2. Constraint checks are not included because of differences in the manner of checking binary and n-ary constraints. Instead, we

[2] http://www.itu.dk/research/cla/externals/clib/esvs.pm

Table 1 Search efficiency comparisons

Domain/tight		Hard problems						Easy problems		
		10/.35			20/.45			20/.40		
		Simple conditional lexicographic								
		lex	stg	bb	lex	stg	bb	lex	stg	bb
P3	Nodes	348	390	217	9083	2334	2020	276	363	1346
	ccks(000)	161	128	115	6505	2260	2124	118	148	949
P7	Nodes	349	390	219	9063	2329	2020	269	365	1297
	ccks(000)	161	128	116	6492	2255	2123	115	149	927
P11	Nodes	348	391	218	9098	2330	2022	269	362	1408
	ccks(000)	161	128	116	6514	2256	2125	117	148	985
		Extended conditional lexicographic								
		stg/lex	stg/dm	bb	stg/lex	stg/dm	bb	stg/lex	stg/dm	bb
P3	Nodes	2396	3547	439	29445	27743	3496	12085	73652	47428
	ccks(000)	1104	1645	203	18905	30294	3660	6833	47508	17219
P7	Nodes	5283	4397	587	103380	43790	5078	98031	106701	87209
	ccks(000)	1594	2043	267	57692	46330	5241	36832	66849	29578
P11	Nodes	14072	6038	957	140285	38344	6801	213311	97095	123614
	ccks(000)	4045	2927	416	66466	40415	7050	59348	62312	44385

Notes. 20-variable problems with density = 0.5. Data are means for 100 problems. "Hard problems" are near the critical complexity peak. "Easy problems" are near the edge of the hard region. Branch and bound employed min domain variable ordering; this was also used for staged lexical search for simple conditional lexicographic orderings. (For the extended case, decision variables, X_i, were either ordered in this way or ordered lexically, as indicated.) "Pk" is number of conditional preference relations per problem

show runtimes. The pattern of results corresponds in some respects to those of the previous experiments, although the differences are more striking. Again, when problems have many feasible solutions (3.1 million in this case), branch and bound algorithms are relatively inefficient. In addition, with these problems certain importance orderings wreak havoc with ordinary variable ordering heuristics, leading to very high means for performance measures.

7.4 Discussion

Taken together, these results indicate that simple conditional lexicographic orders pose no special problems for these algorithms. In the extended conditional lexicographic case, it is important to have some form of lexically-based search—because branch and bound is adversely affected when the number of feasible solutions becomes large, while the effects on lexical or staged lexical search are often not as severe. This is shown most strikingly in the experiments using a real-world configuration problem, and this effect is possibly quite general for these kinds of problems. With our present suite of algorithms, we are thus able to solve problems efficiently under a wide range of conditions despite the reduced restrictions on conditional relations.

These algorithms can also accommodate cases where preference relations are restricted to a subset of the CSP variables, as demonstrated in work with LO-CSPs (Freuder et al.

Table 2 Search efficiency comparisons—configuration problems

		Simple conditional		
		lex	stage	bb
P3	mn. nodes	20	210	40606
	mdn. nodes	20	210	822
	mn. time	.01	.04	9.62
P7	mn. nodes	20	210	40747
	mdn. nodes	20	210	761
	mn. time	.01	.04	9.85
P11	mn. nodes	20	210	43165
	mdn. nodes	20	210	1364
	mn. time	.01	.05	10.69
		Extended conditional		
		stage/lex	stage/dom	bb
P3	mn. nodes	1198	459400	370576
	mdn. nodes	904	10499	6407
	mn. time	0.23	80	57
P7	mn. nodes	1957	195753	1066066
	mdn. nodes	1180	6492	46290
	mn. time	0.33	36	174
P11	mn. nodes	2373	354420	1070914
	mdn. nodes	1905	10769	345247
	mn. time	0.43	51	178

Notes. 20-variable configuration problem with 100 different importance orderings. "mn" and "mdn" are means and medians, respectively, Branch and bound employed min domain variable ordering. This was also used for staged lexical search with simple conditional lexicographic orderings, while for the extended case, decision variables, X_i, were chosen either lexically or by (min) domain size. "Pk" is number of conditional preference relations per problem. Times are seconds

2007). In this case, lexically ordered search considers the non-preference variables after the others using any CSP heuristic, while branch and bound only bounds search when the current variable is in the preference set (and only considers the k preference variables when bounding). Again, after choosing the kth variable by importance, staged lexical search can use any appropriate search heuristic until it has found a solution for this stage.

8 Conclusions

Lexicographic orders allow a very simple and basic representation of preferences in combinatorial problems. The assumptions are strong, but the user inputs are of an easily understandable form, and there are powerful algorithmic approaches for constrained optimisation.

In many situations, preferences are naturally conditional, i.e., context-dependent, and there has been a good deal of recent work in the AI literature on qualitative frameworks for conditional preferences, especially, CP-nets and their extensions. In this paper we define conditional lexicographic orders, which gives a simple approach for reasoning with

conditional preferences, and which has computational advantages over more sophisticated methods. We also show how algorithms for constrained optimisation for ordinary LO-CSPs can be extended to handle conditional orderings. Somewhat unexpectedly, this can be done even when conditional preference relations do not correspond to the relative importance of the variables. Therefore, efficiency of search in combinatorial optimisation can be maintained despite the additional complexity of this form of representation, in some cases to a surprising degree. These algorithms can also be used for finding a single optimal solution of a constrained optimisation problem for an acyclic CP-net.

Acknowledgements This work received support from Science Foundation Ireland under Grant 00/PI.1/C075. We are grateful to the editors and referees for their comments and suggestions, which helped improve the exposition of this paper.

References

Boutilier, C., Brafman, R. I., Hoos, H. H., & Poole, D. (1999). Reasoning with conditional *ceteris paribus* preference statements. In *Proc. fifteenth annual conf. on uncertainty in artif. intell.* (pp. 71–80).

Boutilier, C., Brafman, R. I., Domshlak, C., Hoos, H., & Poole, D. (2004a). Preference-based constrained optimization with CP-nets. *Computational Intelligence*, *20*(2), 137–157.

Boutilier, C., Brafman, R. I., Domshlak, C., Hoos, H. H., & Poole, D. (2004b). CP-nets: A tool for representing and reasoning with conditional *ceteris paribus* preference statements. *Journal of Artificial Intelligence Research*, *21*, 135–191.

Boutilier, C., Domshlak, C., & Shimony, E. (2006). On graphical modeling of preference and importance. *Journal of Artificial Intelligence Research*, *25*, 389–424.

Brafman, R. I., & Domshlak, C. (2002). Introducing variable importance tradeoffs into CP-nets. In *Proc. eighteenth annual conf. on uncertainty in artif. intell.* (pp. 69–76).

Domshlak, C., & Brafman, R. I. (2002). CP-nets—reasoning and consistency testing. In *Proc. eighth conf. on principles of knowledge representation and reasoning* (pp. 121–132).

Fishburn, P. (1974). Lexicographic orders, utilities and decision rules: A survey. *Management Science*, *20*(11), 1442–1471.

Freuder, E. C., Wallace, R. J., & Heffernan, R. (2003). Ordinal constraint satisfaction. In *Fifth internat. workshop on soft constraints—SOFT'02*.

Freuder, E. C., Heffernan, R., Wallace, R. J., & Wilson, N. (2007). Lexicographically-ordered constraint satisfaction problems (submitted).

Goldsmith, J., Lang, J., Truszczynski, M., & Wilson, N. (2005). The computational complexity of dominance and consistency in CP-nets. In *Proc. nineteenth internat. joint conf. on artif. intell.* (pp. 144–149).

Junker, U. (2002). Preference-based search and multi-criteria optimization. In *Proc. eighteenth nat. conf. on artif. intell.* (pp. 34–40).

Keeney, R. L., & Raiffa, H. (1993). *Decisions with multiple objectives. Preferences and value tradeoffs*. Cambridge: Cambridge University Press.

Schiex, T., Fargier, H., & Verfaillie, G. (1995). Valued constraint satisfaction problems: Hard and easy problems. In *Proc. fourteenth internat. joint conf. on artif. intell.* (pp. 631–637).

Wallace, R. J. (2006). Interpretation of preferences based on extended conditional lexicographic orders. In U. Junker, W. Kiessling (Eds.), *ECAI 2006 multidisciplinary workshop on advances in preference handling* (pp. 141–147).

Wallace, R. J., & Wilson, N. (2006). Conditional lexicographic orders in constraint satisfaction problems. In J. C. Beck & B. M. Smith (Eds.), *Lecture notes in computer science: Vol. 3990. Proc. third internat. conf. on integration of AI and OR techniques in constraint programming* (pp. 258–272). Berlin: Springer.

Wellman, M. P., & Doyle, J. (1991). Preferential semantics for goals. In *Proc. ninth nat. conf. on artif. intell.* (pp. 698–703).

Wilson, N. (2004a). Extending CP-nets with stronger conditional preference statements. In *Proc. nineteenth nat. conf. on artif. intell.* (pp. 735–741).

Wilson, N. (2004b). Consistency and constrained optimisation for conditional preferences. In *Proc. sixteenth Europ. conf. on artif. intell.* (pp. 888–892).

Wilson, N. (2006). An efficient upper approximation for conditional preference. In *Proc. seventeenth Europ. conf. on artif. intell.* (pp. 472–476).