Conditional Lexicographic Orders in Constraint Satisfaction Problems

Richard J. Wallace

Cork Constraint Computation Center and Department of Computer Science
University College Cork, Cork, Ireland
email: r.wallace@4c.ucc.ie

Abstract. The lexicographically-ordered CSP ("lexicographic 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 variable k is more important than any change in assignment to any variable that comes after it in the ordering. In this paper, we show how this representation can be extended to handle conditional preferences. This can be done 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 basic ordering of variables. For problems of the first type, any of the algorithms originally devised for ordinary lexicographic CSPs can also be used when some of the domain orderings are dependent on the assignments to "parent" variables. For problems of the second type, we show that a branch-and-bound algorithm originally devised for ordinary lexicographic CSPs can be extended to handle CSPs with conditional domain orderings. Although bounding is necessarily compromised to a degree, our experiments suggest that this algorithm is still reasonably efficient.

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 is based on the assumption 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, 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. Although it is not entirely clear that contexts such as these should always be treated as elements in a preference ordering, this is surely a viable interpretation, especially with human beings who can represent contexts as entities. (And it also seems to be a generally effective manouver within prescriptive contexts.)

Recently, we have investigated the properties of *lexicographically ordered CSPs* [1]. This is a special kind of soft constraint system in which a total ordering is imposed on complete assignments, in terms of the variables and their assigned values. This ordering is lexicographic in form, with the further stipulation that variable selection is the primary factor and value assignment is secondary. This means that a good assignment for a more-preferred variable is more important than a good assignment for a less-preferred variable in deciding the overall ranking of solutions. The preference ordering is assumed to be independent of any constraints that may hold among these variables. The latter, therefore, restrict the alternatives given by an ideal preference ordering to those that can actually be realized.

Lexicographic CSPs are meant to represent problems in which preferences involve multiple objectives and attributes and where feasibility constraints also 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 due in part because of the radical decoupling of the preference structure from the feasibility conditions (as noted earlier by [1]; cf. a similar argument by [2] in connection with CP-nets).

While it is to be expected that there will be situations where simple lexicographic CSPs do not capture all the nuances of the preference relations, in other cases this may be a more appealing approach by virtue of its greater clarity and simplicity. This is suggested by the fact that lexicographic orderings are sometimes used in decision-making applications, despite their extreme assumptions, as noted by [3].

One of the appealing features of this form of lexicographic representation of preferences for CSPs is that it offers wide scope for developing optimization algorithms [1] [4]. In fact, ordinary CSP algorithms can be used as the foundation for such algorithms. If the variable and value orderings follow the lexicographic ordering, then this is all that is necessary because in this case the first solution found is guaranteed to be the optimal one. Otherwise, the algorithm must search for all solutions and by multiple comparisons determine which solution is optimal. Since this is inefficient when there are many feasible solutions, we have also developed a version of branch and bound and shown this to be reasonably effective. In addition, we have developed a specialized restart algorithm (a version of Junker's preference-based search [5]) in which on the kth restart, the kth variable is instantiated in lexical order while better ordering heuristics are used for the remaining variables; in this case, when the first solution is found, its kth assignment will be optimal. Finally, we have begun to consider specialized forms of weighted local search, which can be efficient for finding optimal solutions, although they cannot prove that a solution is optimal.

In this paper, this form of lexicographic ordering is extended to *conditional lexicographic orders*, thereby extending this form of representation to allow for conditional preferences. In this case, the same type of lexicographic ordering holds as in ordinary lexicographic CSPs, but domain orderings are conditional on assignments chosen from other domain. We consider two important classes of conditional lexicographic CSPs. In the first class, conditionalities always respect the priority ordering of the variables; in the second, they do not. As we will see, the latter extension greatly restricts the kinds of algorithms that can be used; however, a variation on the branch and bound algorithms not only remains sound but is also nearly as efficient in practice as the original algorithm.

As a motivating (and clarifying) example, consider a situation in which a customer is deciding among possible vacations. There are two seasons in which he can travel: spring and summer. And for simplicity we consider only two locations: Naples and Helsinki. In the first scenario (first type of conditional lexicographic ordering), location is more important than the time of travel and the preferred season depends on the location chosen. This is shown in Figure 1a, where following [6] the conditional preference is represented as a conditional preference table. The associated preference ordering 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 (second type of conditional lexicographic ordering), location is again the primary feature, but the preference for a location depends on the city chosen. Thus, our customer prefers Naples in the spring but Helsinki in the summer, but he prefers to take his vacation in spring instead of summer. In this case, the preference ordering is:

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

It is of interest to note that the second ordering is intuitively at least as reasonable as the first.

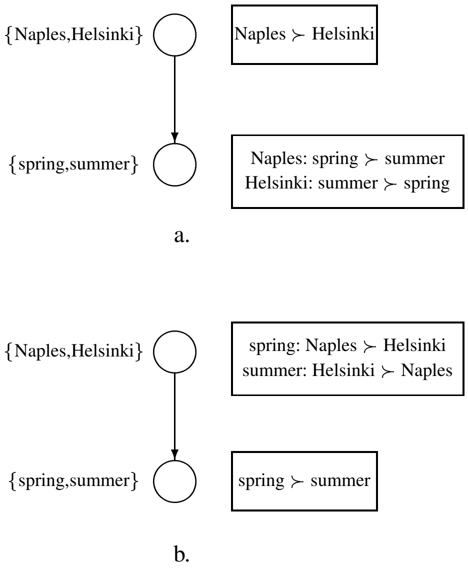

Fig. 1. Two examples of conditional lexicographic preference orderings. In (a) the conditions are consistent with the priority of variables; in (b) the conditions oppose the priority ordering.

An alternative representation of conditional preferences that has received much attention in recent years is the "CP-net" [6] [7]. Since CP-nets do not require total orders, they are in some respects a more flexible form of representation. However, the present formulation provides a novel kind of flexibility, in that it allows conditionalities to oppose the priority ordering. It may, therefore, be worth exploring the relations between these two forms of representation as well as their relative strengths and weaknesses.

The remainder of the paper is organized as follows. Section 2 gives formal definitions of lexicographic CSPs and CSPs with conditional lexicographic orders, and provides a short discussion on the relations between these systems and the more general soft constraint representations. Section 3 discusses relations to CP-nets and TCP-nets. Section 4 gives a summary of algorithms for solving ordinary lexicographic CSPs. Section 5 discusses algorithms that can handle conditional lexicographic CSPs of either type. Section 6 gives conclusions.

2 Background and Definitions.

2.1 Definitions

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 the labeling $\lambda(i)$ of V as $X_1, \ldots, X_n$. 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 an assignment α^* such that

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

Definition 2. Conditional lexicographic CSP. A *conditional lexicographic structure* over V is defined as a tuple $K = \langle\lambda, G, CPT\rangle$, where λ is a labelling of V, with $\lambda(i)$ being written X_i, G is a directed acyclic graph on V which is compatible with λ, i.e., $(X_i, X_j) \in G$ implies $i < j$. CPT is a function which associates a conditional preference table $CPT(X)$ to each $X \in V$. Each conditional preference table $CPT(X_i)$ associates a total order $>_u^{X_i}$ with each instantiation u of the parents U_i of X_i (with respect to G). 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 $u = \alpha(U_i) = \beta(U_i)$. It is easily seen that $\succ_K$ is a total order on assignments.

Definition 3. Extended Conditional Lexicographic CSP. An *extended conditional preference order* involves a function Q which assigns a number $Q(x|u)$ for every value x of X_i and assignment u to U_i. The conditional preference order is then defined as follows: to compare assignments α and β we find the first X_i where $Q(\alpha(X_i)|\alpha(U_i))$

is not equal to $Q(\beta(X_i)|\beta(U_i))$. If $Q(\alpha(X_i)|\alpha(U_i))$ is less than $Q(\beta(X_i)|\beta(U_i))$, we prefer α to β; else we prefer β to α.

Another way of viewing this is that we are converting each assignment $\alpha = (x_1, \ldots, x_n)$ to an n-tuple of numbers $\alpha' = (Q(x_1|u_1), \ldots, Q(x_n|u_n))$, where u_i is the assignment α makes to U_i. The conditional lexicographic order $>_L$ is then just the standard lexicographic order on these n-tuples of numbers: α is preferred to β if and only if α' is lexicographically less than β'. Hence $>_L$ is a total order.

2.2 Lexicographic CSPs and soft constraints.

The lexicographically-ordered CSP is a special case of the "lexicographic CSP" or "lex-VCSP" as defined in [8]. As these authors show, lex-VCSPs are in turn 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 [8] and refer to it by their term, "lexicographic CSP", it is a very special case of the class that 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 use the term "lex-VCSP" to refer to the more general category of CSPs whose evaluations can be ordered lexicographically.

An evaluation structure for CSPs involving a lexicographic ordering was originally developed within the fuzzy CSP context to avoid the limited discriminability between solution values due to the use of fuzzy min and max operations for combining and comparing evaluations. In this formulation, preferences for k-tuples associated with a given constraint are ordered by increasing magnitude, and two solutions are compared beginning with the first members of each ordering and proceeding up the lists until a difference is found [9]. In addition, constraint priorities can be incorporated by associating a priority level with each constraint, and making the evaluation for a tuple the maximum of its preference value and the complement of the priority value, i.e.

$$\mu_S(u_1, \ldots, u_k) \;=\; \max(1 - \alpha_C, \mu_R(u_1, \ldots, u_k))$$

where μ_S and μ_R are evaluations of the fuzzy relations S and R associated with constraint C, and α_C is C's priority level. A related and somewhat more general formulation falling within the valued CSP framework is in terms of orderings of constraint violations, where the combinator is multiset-union with an additional top ($\top$) value that acts as an absorbing element and can be used to represent violations of hard constraints. Comparison then involves sorting the multisets associated with each solution and comparing them lexicographically, beginning with the highest value and choosing the evaluation with the smaller value for the first difference found [8]. These authors also show that lexicographic CSPs of this form are equivalent to weighted CSPs, with positive ∞ serving as the top value.

Lexicographic CSPs as we have defined them fall under the second class described above (though not the first, in which priorities and preferences are balanced). For our purposes (and perhaps in general), the weighted CSP formulation is more straightforward. In this case, one avoids the oddities of combining $\top$ with multisets and comparing a single evaluation with a multiset when this element is involved. We can embed a lexicographic ordering within a weighted CSP framework as follows:

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 the weights associated to α is less than the sum associated to β if and only if $\alpha >_L \beta$.

3 Lexicographic CSPs and CP-nets.

In recent years, the most popular or at least the most advertised means of representing conditional preferences has been the conditional preference network with *ceteris paribus* assumptions, or CP-net [6]. A more recent variant, the TCP-net [10], includes elaborations to handle relations of importance between the features of user-selections. This corresponds to the ranking of variables in lexicographic CSPs.

CP-net structures are based on assignments of values to variables, or "features". A conditional preference is encoded in a "conditional preference table" (CPT) associated with a particular variable X_i. TCP-nets also encode importance relations between variables in terms of an ordering with lexicographic features, as well as representing conditional importance relations in a manner analagous to conditional preferences.

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 $a_1 b_1 >_N a_2 b_1$, $a_2 b_1 >_N a_2 b_2$, etc, but we cannot order $a_1 b_2$ and $a_2 b_1$ on this basis. As a result of this feature, preference orders can be established on the basis of "flipping sequences" (as illustrated in the last example). This is still true of TCP-nets, although in some cases adjacent outcomes in a sequence can be separated by two flips rather than one.

In this connection, it is worth noting 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 [11]. 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. In addition, for acyclic networks in which conditional preferences correspond to the priority ordering, it can be shown that CP-nets are dominated by conditional lexicographic orderings in that a given CP-net always implies a given conditional lexicographic ordering [4].

Perhaps the most important implication of these differences is that, while determining whether solution o is (necessarily) preferred to solution o' is easy for lexicographic orderings, since it is based on successive comparisons of indices, this can be difficult with (T)CP-nets, since it depends on finding flipping sequences for transforming one alternative into another [12]. On the other hand, CP-nets allow a weaker form of comparison, which indicates for two solutions o and o', that the preference of the latter over the former is *not* entailed by the CP-net structure. This form of comparison can be carried out in linear time [7].

Although (T)CP-nets do not encode feasibility constraints directly, the orderings that they represent can be combined with such constraints in somewhat the same way that lexicographic CSPs combine a particular preference ordering with a constraint representation. It has been shown that for CP-nets with feasibility constraints, a set of Pareto-optimal solutions can be obtained using the weaker comparisons that were just described [2]. In this case the CP-net implies a priority ordering that must be followed during of search. As shown below, a comparable requirement can be lifted in the case of some algorithms for conditional lexicographic CSPs.

4 Methods for Solving Ordinary Lexicographic CSPs

For reference, we outline four methods examined earlier in the context of ordinary lexicographic CSPs. (We omit the incomplete methods.) We also present some results of experimental tests with random problems to show comparative performance. We can simulate total orders with these problems, where variables and values are represented by integers, by using these labels as the required indices. In both cases, lower integer values represent preferred elements. Thus the solution, (1/1, 2/1, 3/1, ..., n/1), where x/y is the variable/value labeling, is the most preferred, (1/1, 2/1, 3/1, ..., n/2) the next-most preferred, etc. In keeping with the definition of a lexicographic ordering, a shift of value from k to $k+1$ for a given variable represents a greater change in preference than a shift from k to $k+r$ for any variable with a higher index number. Since in previous work a MAC-based algorithm proved to be much more effective than forward checking, the former is used in all tests in this paper.

As noted before, ordinary CSP algorithms based on a simple lexicographic ordering can be used to find optimal solutions to these problems. In addition, CSP algorithms with variable ordering heuristics can also be used, but in this case the all-solutions problem must be solved.

In addition to these two procedures, we consider a simple branch and bound procedure Here, we can also use effective CSP heuristics; in fact, this can be seen as a method of improving the original CSP algorithm that is based on these heuristics. The cost function gives large values for any but very small problems, but we do not need to calculate it directly. Rather, we simply compare successive values following the lexical variable ordering until we encounter a difference. Specifically, suppose that variable X_i is the variable currently being considered for instantiation, and this variable is the kth most important variable in the ordering. To evaluate the current partial solution, we start from the first variable in the lexical ordering. If a variable has an assignment, 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 domain against the best assignment. In either case, if we encounter a value greater than the best found so far, then search can back up.

The final procedure is specialized for this type of preference ordering; we refer to it as "staged lexical search". Search is done repeatedly, in each case until the first solution is found, and for each repetition, or stage, of search one more variable is chosen in lexical order. Values are always chosen according to the lexical ordering. Thus, in Stage 1 we first select variable X_1 according to the lexical ordering, and then use any

Table 1. Search Efficiency Comparisons

	hard problems			easy problems		
domain size	10	20	30	10	20	30
tightness	0.35	0.45	0.50	0.30	0.40	0.45
CSP lexical						
median nodes	149	4631	30,959	26	85	292
mean nodes	347	9084	121,786	37	174	1027
mean solns	1	1	1	1	1	1
CSP min domain						
median nodes	1402	7404	26,053	373,260	-	-
mean nodes	1599	7927	29,858	410,654	-	-
mean solns	193	35	8	196,731	-	-
branch and bound						
median nodes	165	1238	6252	322	945	2381
mean nodes	217	2020	9395	380	1189	2991
mean solns	2	1	1	6	6	6
staged lexical						
median nodes	325	1511	7152	230	338	506
mean nodes	390	2330	9778	237	375	607
mean solns	20	20	20	20	20	20

Notes. Twenty-variable problems, sample size 100. MAC algorithm. “hard problems” are near the critical complexity peak for lexical ordering. “easy problems” are near the edge of the hard region for lexical. “solns” is number of solutions found during the entire search; for CSP min domain this is the total number of solutions per problem. Branch and bound and staged lexical algorithms employed the min domain ordering.

heuristic to select the others. When we have found a feasible solution, we know that the assignment for X_1 is optimal, so we retain it for the remainder of search. In Stage 2, we first select variable X_2, so the first feasible solution found will include an optimal assignment for this variable. And so forth. Although developed independently, this algorithm is, in fact, a special case of preference-based search [5], where the criteria on which search is based form a total order.

Performance comparisons are given in Table 1. These are for problems with parameters $<20,10,0.50,0.20>$. (Note that the number of values per domain is large with respect to problems normally considered in this context and that there are numerous hard constraints.) The results suggest that for hard problems either branch and bound or staged lexical perform well, while throughout the range of easy problems an ordinary CSP algorithm with lexical variable (and value) ordering is the most efficient procedure.

5 Algorithms for Conditional Lexicographic CSPs.

When the parent-child order is compatible with the importance order of the variables, any of our methods for constrained optimization can be used to return a solution that is optimal for the conditional lexicographic CSP. However, results in the previous sections show that, if the CSP is strongly constrained, finding a single optimal solution can be made substantially more efficient by using an alternative algorithm such as the staged lexical or branch and bound. In particular, the staged lexical algorithm can be applied in exactly the same way as before to the conditional lexicographic case (since at stage i we know the ordering of the values of X_i, as its parents have been instantiated already). Either this algorithm or the branch and bound algorithm should be faster for highly constrained problems (cf. Table 1).

Lexicographic CSPs based on extended conditional preference orders are probably not amenable to search based on lexical ordering (and certainly not to any straightforward version of lexically-based search). This is because the preference ordering for a domain may not be known at the time of instantiation, and in such cases each value may potentially have any rank in the domain ordering. While one can select a value on the basis of the conditional ordering for the best value of the primary ancestor, if the latter becomes unavailable, then the original assignment must be revised. Under these conditions, it is not clear that a lexical order of search can be determined at all while still ensuring completeness, let alone maintaining efficiency.

For branch and bound, however, flexibility of variable ordering in search can be retained:

Proposition 1. *A branch and bound procedure whose cost function is the lexicographic ordering based on $Q(x|u)$ solves the (extended) conditional lexicographic CSP correctly under any order of variable instantiations.*

Proof Sketch. We argue this as follows. As usual, we refer to variables whose preference order depends on other variables' instantiations as "children", and the variables they depend on as their "parents". Since multiple children can be treated independently, treating the case of one child is sufficient. For more than one parent, the cases depend on the last parent instantiated, so it is sufficient to consider a singleton set. This gives four basic cases:

1. parent $\succ$ child $\bigwedge$ parent is instantiated first
2. parent $\succ$ child $\bigwedge$ child is instantiated first
3. child $\succ$ parent $\bigwedge$ parent is instantiated first
4. child $\succ$ parent $\bigwedge$ child is instantiated first

(Here, $X_i \succ X_j$ indicates that index $i <$ index j in the labeling λ.) Case 1 needs no comment. In Case 3, the child's preference order will always be fixed at the time of instantiation, so there is no special problem here, either. In Cases 2 and 4 the child's order is unknown at the time of instantiation, but an assignment can be chosen consistent with the best assignment of the parent. Then, if this assignment is available when the parent is instantiated, there is no problem. In the other situation, the problem occurs when the next assignment is considered for the child; here, if there is a possible assignment that $>^u_{X_i}$ the best-assignment-found, then search cannot be bounded.

The algorithm is complete because at any level of search all viable values of the current domain are tested in a particular order.

In addition to restrictions on when bounding can occur, the major difference from the branch and bound algorithm for simple lexicographically-ordered CSPs is that comparisons must use the rank for a value that held when the solution was found. The algorithm can be also enhanced by testing for special cases where bounding can still be done. For example, if the current variable being checked is uninstantiated and is a child-variable in some relation, but the domains of the uninstantiated parents in all such relations can be ordered, then the best values for those parents can be used to derive a tentative ordering for the domain of the variable being checked that together with these parent values gives a best partial solution.

To test the efficiency of a branch and bound algorithm for the extended lexicographic CSP, a problem generator was written. This program starts with an existing CSP and transforms it into a conditional lexicographic CSP by selecting variables for conditional preferences and building a CPT for each relation. The user specifies the following parameters:

- number of preference relations
- maximum number of parents per relation
- maximum number of children per relation
- maximum number of attempts to make a relation with p parents and c children (If this number is ever exceeded, the program writes a message to standard-output, but continues with the 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 is not a required restriction, but it prevents 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.

At present, 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 lexicographic CSP is of the simpler type.)
2. That the parents and children in a relation do not have parents in common.

The branch-and-bound procedure (Figure 2) relies heavily on the fact that for lexicographic orderings, value orderings can be indexed. This allows it to check bounds in terms of indexes, thereby comparing a candidate assignment with previous assignments even when the preference ordering for the past assignment is different from the present ordering. (In the current implementation, indexes are not stored as such; comparisons are made by comparing cardinalities of sublists beginning with the values compared.) Bounds checking proceeds lexicographically; if a variable has an assignment, this value is compared with the value in the best solution found so far - in terms of their indexes. If it does not have an assignment, a comparison can still be made between the best possible value in the current domain and the value in the best solution found. In addition, it is sometimes possible to determine a best value for an uninstantiated variable as indicated in the last two else-if clauses under the while in the bounds-check function.

```
conditional-bnb (partial-solution, remaining-variables)
    if remaining-variables ≡ nil
        save new best-solution
        and continue            //backtrack
    else
        select next variable and remove from remaining-variables
        for each value in its ordered domain
            if new instantiation gives an 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
        if next-variable ∉ any child-set
            compare curr-assign or best value in default-current-domain with next-best
        else if there is no current-preference order
            compare curr-assign or best value in default-current-domain with next-best
        else if candidate-var is a remaining uninstantiated parent
            get domain-order associated with parent values
            compare curr-assign or best value according to domain-order with next-best (using indexes)
        else if domains of remaining uninstantiated parents can be ordered
                  //parents not children or have current ordered domains
            compare curr-assign or best possible value given best possible parent-tuple
            with next-best (using indexes)
        if comparison has succeeded break  //one alternative was better
    if comparison succeeded and bound was exceeded
        return false
    else
        return true
```

Fig. 2. Pseudocode for branch and bound for CSP with conditional lexicographic orderings.

We present some preliminary results for problems derived from two sets of 20-variable problems listed in Table 1: (i) $|d|$ = 10, tightness = 0.35, (ii) $|d|$ = 20, tightness = 0.45. For generation, the maximum number of parents or children per relation was limited to two, in the first cases to limit CPT size, in the second to allow a sufficiently large number of relations (since the same variable cannot be a child in more than one relation). The number of preference relations was 7, and these included 70-80% of the variables in the problem. (This is probably a much more severe case than will be encountered in practice.) For problem set (i) the median and mean number of search nodes was 274 and 384, respectively, and 3 solutions were found on average including the optimal one. For set (ii) the median and mean were 2672 and 3526, respectively, with 2 solutions found. Given the potential for search to blow up under these conditions in comparison with ordinary lexicographic CSPs, these results are impressive. We conclude that this algorithm is still efficient despite the necessary restrictions on bounding.

6 Conclusions.

This work shows that conditional preferences can be incorporated into this type of lexicographic representation for CSPs, thus extending the scope of this form of representation in an important manner. This means that the desireable features of lexicographic CSPs, such as ease of comparison between solutions and the strict decoupling of preferences and feasibility constraints, can be carried over to the case of conditional preferences.

Algorithms for ordinary lexicographic CSPs can be extended to handle conditional orderings; somewhat surprisingly, this can be done in one case even when the conditionalities do not correspond to the ordering of variables. So to a large degree, efficiency of search for combinatorial optimisation can be maintained despite the added complexity of this form of representation.

Acknowledgment. This work was supported by Science Foundation Ireland under Grant 00/PI.1/C075. Definitions in Section 2 are from [4] and are largely due to N. Wilson.

References

1. Freuder, E.C., Wallace, R.J., Heffernan, R.: Ordinal constraint satisfaction. In: Fifth Internat. Workshop on Soft Constraints - SOFT'02. (2003)
2. Boutilier, C., Brafman, R.I., Domshlak, C., Hoos, H., Poole, D.: Preference-based constrained optimization with CP-nets. Computational Intelligence, Special Issue on Preferences (2004) 137–157
3. Keeney, R.L., Raiffa, H.: Decisions with Multiple Objectives. Preferences and Value Tradeoffs. Cambridge (1993)
4. Freuder, E.C., Heffernan, R., Prestwich, S., Wallace, R.J., Wilson, N.: Lexicographically-ordered constraint satisfaction problems. unpublished (2005)
5. Junker, U.: Preference-based search and multi-criteria optimization. In: Proc. Eighteenth Nat. Conf. on Artif. Intell., AAAI Press (2002) 34–40
6. Boutilier, C., Brafman, R.I., Hoos, H.H., Poole, D.: Reasoning with conditional *ceteris paribus* preference statements. In: Proc. Fifteenth Annual Conf. on Uncertainty in Artif. Intell., Morgan Kaufmann (1999) 71–80

7. Boutilier, C., Brafman, R.I., Domshlak, C., Hoos, H.H., Poole, D.: CP-nets: A tool for representing and reasoning with conditional *ceteris paribus* preference statements. Journal of Artificial Intelligence Research (2004) 135–191
8. Schiex, T., Fargier, H., Verfaillie, G.: Valued constraint satisfaction problems: Hard and easy problems. In: Proc. Fourteenth Internat. Joint Conf. on Artif. Intell., Morgan Kaufmann (1995) 631–637
9. Fargier, H., Lang, J., Schiex, T.: Selecting preferred solutions in fuzzy constraint satisfaction problems. In: Proc. First European Conf. on Fuzzy and Intelligent Technologies - EUFIT'93. (1993) 1128–1134
10. Brafman, R.I., Domshlak, C.: Introducing variable importance tradeoffs into CP-nets. In: Proc. Eighteenth Annual Conf. on Uncertainty in Artif. Intell. (2002)
11. Wilson, N.: Extending CP-nets with stronger conditional preference statements. In: Proc. Nineteenth Nat. Conf. on Artif. Intell. (2004)
12. Domshlak, C., Brafman, R.I.: CP-nets - reasoning and consistency testing. In: Proc. Eighth Conf. on Principles of Knowledge Representation and Reasoning, Morgan Kaufmann (2002) 121–132