# Strategies for Interactive Constraint Acquisition

Sarah O'Connell, Barry O'Sullivan, and Eugene C. Freuder

Cork Constraint Computation Centre
Department of Computer Science
University College Cork, Ireland
{s.oconnell | b.osullivan | e.freuder}@4c.ucc.ie

**Abstract.** In many practical applications users often find it difficult to articulate their constraints. While users can recognize examples of where a constraint should be satisfied or violated, they cannot articulate the constraint itself. In these situations we would like the computer to take an active role in acquiring the user's constraints. In this paper we present an approach to interactive constraint acquisition based on techniques from the field of machine learning. Constraint acquisition is modeled as search through an "hypothesis space" of constraints over which a general-to-specific ordering is known or is implicit in the hypothesis space representation. We have begun studying the issues involved in interactively acquiring constraints. During interactive constraint acquisition we would like to employ a strategy which minimizes the dialog length between the user and the computer. In this paper, we compare a number of different heuristics for interactive query generation. We demonstrate that the interactive strategy used to acquire constraints depends on the topology of the hypothesis space and the degree to which the human user is being helpful.

## 1 Introduction

The vast majority of research in the field of constraint processing assumes that either (a) the constraints of a problem are known in advance or (b) can be articulated precisely during an interactive session. This assumption is not always reasonable in the real-world.

In many practical applications users find it difficult to articulate their constraints. While users can recognize examples of where a constraint should be satisfied or violated, they cannot articulate the constraint itself. There are many reasons for this. For example, a customer may be trying to specify a constraint to an engineer, without being able to use the correct engineering terms for the relevant concepts. By pointing out to the engineer examples of what is acceptable, the customer can be assisted in finding the right term for what he is trying to articulate. This can be regarded as an instance of interactive constraint acquisition. The customer cannot articulate his constraint but, by interacting with the engineer, his examples of what he would accept or reject help to define his constraint.

Recently, researchers have become more interested in techniques for solving problems where users have difficulties articulating constraints. Freuder and Wallace have considered suggestion strategies for applications where a user cannot articulate all constraints in advance, but can articulate additional constraints when confronted with something which is unacceptable [2]. However, that work assumes that the user can actually articulate each of the constraints when required. In essence their approach is model focussed, in the sense that they are concerned with acquiring a partial model of the user's CSP which is sufficient to be able to find a solution that the user would find acceptable. Padmanabhuni *et al.* have proposed a framework for learning constraints [7]. However, they do not show how negative (unacceptable) examples can be handled by their approach. Rossi and Sperduti have studied the utility of using inductive learning, through the application of gradient descent techniques, for acquiring global and local preferences in the context of interactive soft constraint solving [8]. Their approach assumes that the constraints of the problem are known but that the local and global preferences are not. Freuder and O'Sullivan have begun studying the interactive acquisition of tradeoff constraints in interactive constraint-based configuration [1]. Their approach focuses on generating constraints which model tradeoffs between variables in problems which have become over-constrained during a interactive configuration session. Again, while some constraints in the problem are being acquired, it is assumed that the user can accurately articulate constraints, and that the issue is how to remove over-constrainedness in a model. Finally, a preliminary report on the research reported here has also been presented [6]. Therefore, while constraint acquisition, in one sense or another, is receiving some attention, there is a lack of work on techniques which can support the acquisition of individual constraints necessary to model the user's problem. It is this gap in the research which the work reported here is beginning to address.

In this paper we present an approach to interactive constraint acquisition based on techniques from the field of machine learning. In particular, constraint acquisition is regarded as search through an "hypothesis space" of constraints over which a general-to-specific ordering is explicitly known, or is implicit in its representation. Examples provided by the user can be used to identify a *version space* ([4]) of constraints that the user could be attempting to articulate. We employ an interactive model similar to that proposed by Sammut and Banerji for acquiring constraints from the user [9]. As the user provides examples of what he would find acceptable (positive examples), we attempt to generalize these examples by asking questions that attempt to qualify a hypothesized generalization. If the user finds a generalization unacceptable, this provides us with a negative example that can be used to further refine the version space of the constraint being acquired. We continue an interaction with the user until the constraint has been acquired. In addition, since we are concerned with acquiring constraints from a human user, we are also interested in strategies which attempt to minimize the length of the dialog between the human and the computer.

The contributions of this paper are as follows:

1. We present an approach to acquiring single constraints from user-specified examples;
2. We present an interactive model for acquiring constraints;
3. We present alternative strategies for generating queries to present to the human user and offer some insights into choosing the best strategy for a particular hypothesis space topology;

The remainder of the paper is organized as follows. Section 2 presents an overview of concept learning using version spaces, a model of interaction with the user, and an example of the approach used to acquire single constraints. Section 3 presents a number of alternative strategies for generating generalization queries that attempt to minimize dialog length with the user. Section 4 presents an evaluation of the alternative query strategies described in Section 3. Finally, in Section 5 a number of concluding remarks are made.

## 2 An Overview of Version Space Learning

Concept learning can be defined as inferring a boolean-valued function from training examples of its input and output [5]. Concept learning can be regarded as search through a space of hypotheses implicitly defined by the hypothesis representation. A number of algorithms for concept learning are known: the *Find-S* algorithm, the *List-Then-Eliminate* algorithm, the *Candidate-Elimination* algorithm [3, 5].

Here Mitchell's *List-Then-Eliminate* algorithm is used as the basis for an interactive constraint acquisition system. This algorithm outputs a description of the set of all hypothesis consistent with the training examples. An hypothesis is *consistent* with the training examples if it correctly classifies these examples: negative examples are classified as negative, positive examples are classified as positive. The term *version space* is used to describe the subset of the hypothesis space consistent with the training examples. Readers who are unfamiliar with version space learning are encouraged to refer to the literature [4, 5].

In Section 2.1 we present the model of interaction used in this work. An example of an interactive session involving the acquisition of a simple placement constraint is presented in Section 2.2.

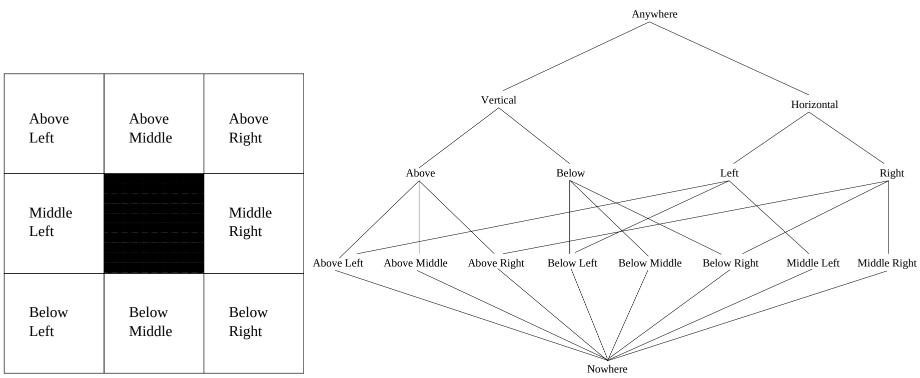
### 2.1 A Model for Interactive Constraint Acquisition

The model of interaction employed here is as follows. The user is assumed to be incapable of articulating his constraint, but can provide and recognize positive and negative examples of it. The interaction begins with the user providing one or more examples of the constraint that he wishes to articulate. Using an appropriate version space learning algorithm, such as the *List-Then-Eliminate* method (which is used here) or the *Candidate Elimination* method, the user's example can be used to determine a version space for the constraint. The system then attempts to generalize the user's examples. This generalization step involves the system choosing an hypothesis from the current version space and attempting to validate it. This attempt at generalization involves proposing a single *qualifying example* (a query) to the user based on the the candidate hypothesis. If the qualifying example is accepted by the user, the system attempts to generalize again. An accepted qualifying example is implicitly positive and is therefore used to further refine the version space for the constraint. Generalization continues until either the version space for the user's constraint has been reduced to a single hypothesis, or an example has been proposed to the user which he rejects. A rejected example is implicitly negative. Therefore, these negative examples are also used to further refine the version space for the constraint. If the user rejects an example he is asked to provide another positive example. Each example can only be used once during a constraint acquisition session, either as a user-specified example or as a system-generated qualifying question.

Below an example is presented which will illustrate the process of interactive constraint acquisition proposed in this paper.

### 2.2 An Example – Acquiring a Placement Constraint

Using the model to interaction presented in Section 2.1 let us consider a scenario where an hypothesis space related to the placement of a queens around a box is known. This can be regarded as the acquisition of a binary constraint. Figure 1 illustrates the scenario. A queen can be placed in any box, for example in the *Above Right* position.




(a) The placement grid

(b) An example version space

**Fig. 1.** An example constraint acquisition problem.

The hypothesis space presented in Figure 1(b) defines the various constraints that can be used to define the placement of the queens with respect to the box (indicated in Figure 1(a) as a black filled square). This hypothesis space is ordered general-specific from top to bottom. For example, if a queen was placed in the *Above Right* position, this indicates that the placement constraint is one of the following:

- the queen can be placed *Above Right* with respect to the box;
- the queen can be placed *Above* with respect to the box;
- the queen can be placed *Vertical* with respect to the box;
- the queen can be placed *Right* with respect to the box;
- the queen can be placed *Horizontal* with respect to the box;
- the queen can be placed *Anywhere* with respect to the box.

This is because each of these constraints are consistent with, and more general than or equal to, the the given example (*Above Right*). In addition, if the user was to state that the *Above Left* was acceptable, but that *Below Right* was not, then this would imply that the user is articulating the constraint *Above*, since it is the only constraint which is consistent with the set of examples he has given. The task in constraint acquisition using version spaces is to acquire the maximally-specific constraint which is consistent with all of the examples that the user presents.

From Figure 1(b) we can also see that the set of possible examples that the user and system can use comprises the hypotheses *Above Left*, *Above Middle*, *Above Right*, *Below Left*, *Below Middle*, *Below Right*, *Middle Left* and *Middle Right*.

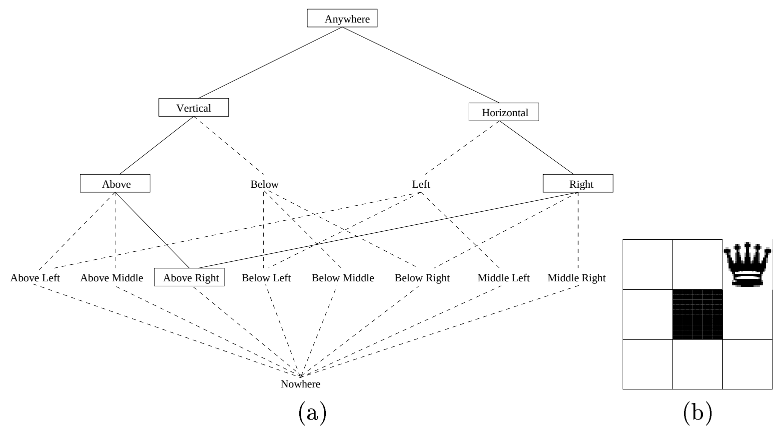
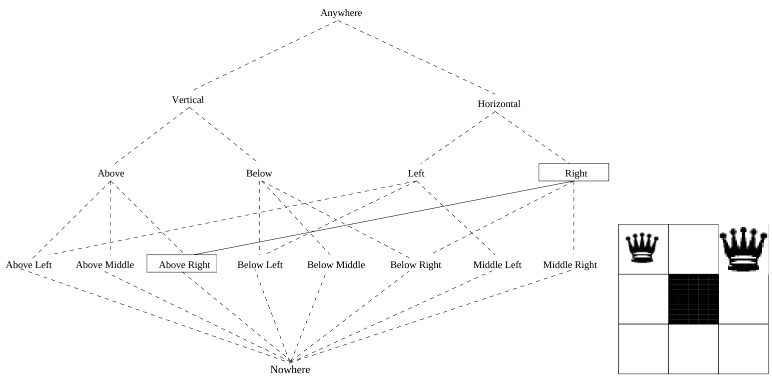



**Fig. 2.** The version space is illustrated in (a) that is associated with the user-specified example shown in (b).

For the purposes of our example, let us assume that the constraint that the user is attempting to articulate is that "*queens can be placed to the right of the box*". Let us also assume that the user's first (positive) example of the constraint to be acquired is that the queen can be placed in the *Above Right* position. This user-specified example is illustrated in Figure 2(b). The version space corresponding to this example is illustrated in Figure 2(a), using boxes to indicate the hypotheses of the original hypothesis space that are consistent with the user's example.

The set of maximally specific generalizations of the user's example is {*Above*, *Right*}. We assume for the purposes of the current discussion that the system selects, lexicographically, the hypothesis *Above*. This hypothesis can be qualified by asking the user if either *Above Left* or *Above Middle* are acceptable. The system selects, lexicographically, the qualifying example *Above Left*. This is illustrated in Figure 3(b).

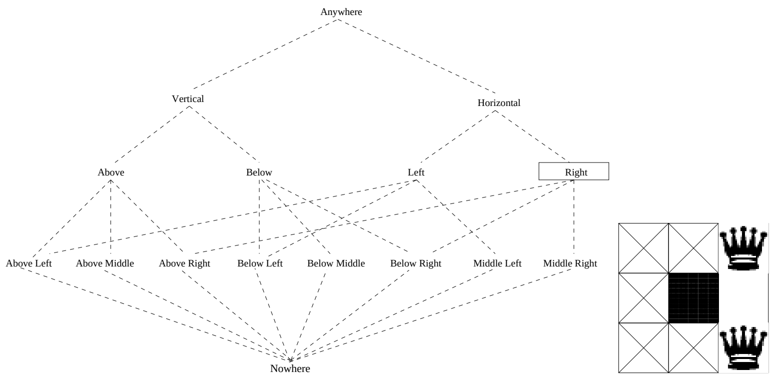



(a) (b)

**Fig. 3.** The version space is illustrated in (a) that is associated with the user's rejection of the qualifying example shown in (b) – that the hypothesized constraint *Above* is acceptable.

Since this example is inconsistent with the constraint that the user is trying to articulate he would reject it. The effect of this is that *Above Left* is now regarded as a negative example of the user's constraint. Therefore, the user's constraint cannot be *Above*, which also implies that it cannot be *Vertical* or *Anywhere*. In addition, the user's constraint cannot be *Left*, which also implies that it cannot be *Horizontal*. Therefore, the only remaining possible generalization is that the constraint is that the queen can either be placed to the *Above Right* or *Right* of the box. The version space corresponding to this scenario is illustrated in Figure 3(a).

Since the system-generated qualifying example was rejected by the user, the system requests another user-specified example. In Figure 4(b) the user presents *Below Right* as a positive example. The consequence of this example is that the version space is refined to contain the single hypothesis *Right* (Figure 4(a)). Therefore, since the version space has been reduced to a single hypothesis the




(a) (b)

**Fig. 4.** An example constraint acquisition problem.

system terminates the interaction having acquired the constraint that "*queens can be placed to the right of the box*".

The example interaction presented here is rather simplistic. However, in reality a user may provide us with several examples of what would be acceptable and what would not – multiple positive and negative examples. Also, we may wish to attempt to qualify generalizations at arbitrary positions in the version space. Such scenarios are trivial to handle with the version space approach.

When attempting to generalize the user-specified example the system has an opportunity to be intelligent. There are a number of strategies which the system could use to generate qualifying questions. In the remainder of this paper we will consider this issue in greater detail. We will also discuss the effect of user helpfulness on the dialog length between the human user and computer. In particular, in Section 3 we will present a number of alternative generalization strategies. In Section 4 we will present results of an empirical comparison of these strategies.

## 3 Generalization Strategies

In the work presented here, generalization is performed interactively through the computer's generation of a qualifying question and presenting it to the user. The machine learning community refer to this a query. It is generally accepted that the optimal query generation strategy to adopt when learning using version spaces is one that involves generating instances that satisfy exactly half of the hypotheses in the version space. For the purposes of this paper we will refer to this strategy as *50/50*.

Given that we are concerned with the interactive acquisition of constraints, one of the key metrics that is relevant to us is the total number of interactions with the user (the dialog length). Obviously, we would wish this number to be minimal. Therefore, query generation strategies which have the potential to minimize the length of the interaction (dialog) with the human are of significant value. One could imagine that in the real-world the difference between a user using an interactive constraint acquisition system and abandoning it maybe their lack of tolerance of one more interaction.

Thus, notwithstanding the general power of the 50/50 approach, there may be some characteristics of the acquisition problem which make it difficult for 50/50. Therefore, it is worthwhile posing the question: does the combination of acquisition problem, user profile and version space topology have an effect on the performance of different query generation strategies? In particular, can we tailor our query generation strategy so that the total number of interactions with the human user is minimized? We have begun to explore this issue and report some of our results here. The query generation strategies that we have studied are:

- 50/50 – our baseline case
- Random
- Least Generalization (Random)
- Least Generalization (Maximum Connectivity)
- Least Generalization (50/50)

Below we will briefly describe each of these query generation strategies. In Section 4 we will compare these empirically in a number of different constraint acquisition scenarios.

*50/50 (Baseline case)* – This heuristic represents the conventional wisdom in the machine learning field. The heuristic generates queries which satisfy half of the hypotheses in the current version space. This is the best known general-purpose query generation strategy for version space learning.

*Random* – In the real-world, strategies based on randomness can often perform surprisingly well. In the context of version space learning, this strategy selects a query at random from the set of examples that could be used to qualify a generalization.

*Least Generalization (Random)* – One disadvantage with the previous two strategies is that while they are based on sound principles, they can give rise to the generation of queries that are difficult for the user to explain from the perspective of perspecuity. Therefore, it may be worthwhile to use strategies that generate queries which the user would regard as "logical" generalizations. One such strategy would be to ask the user a question which attempts to generalize the user's examples in a minimal way. In other words, adopt a strategy which attempts to qualify the least generalized hypothesis. The *Least Generalization (Random)* strategy generates a query at random from the set of examples that could be used to qualify a minimally generalized hypothesis.

*Least Generalization (Maximum Connectivity)* – This strategy attempts to qualify a least generalized hypothesis from the user's examples by generating a query which, if rejected, would reduce the size of the version space by as much as possible. This strategy selects a query whose set of generalizations is maximal. This is a risky strategy since it relies on a negative response to the query to significantly reduce the size of the version space.

*Least Generalization (50/50)* – This strategy attempts to qualify a least generalized hypothesis from the user's examples by generating a query which, if either rejected or accepted, would reduce the size of the version space by the same amount. In other words, this strategy attempts a least generalization step based on the query which reduces the version space by a similar amount regardless of whether it is accepted or rejected.

## 4 Evaluation

The objective of our evaluation was to compare the length of dialog (number of interactions) between the user and computer required to acquire a constraint. The evaluation was performed by varying (a) the topology of the original hypothesis space and (b) the extent to which the user was trying to be helpful.

We studied two alternative hypothesis space topologies. Each topology was parameterized with a single integer value representing the number of atomic examples that formed the vocabulary of the interaction. Figure 5 presents the hypothesis-space topologies used for the purposes of our evaluation. We will refer to the topology illustrated in Figure 5(a) as the *tree topology*, while we will refer to the topology illustrated in Figure 5(b) as the *lattice topology*.

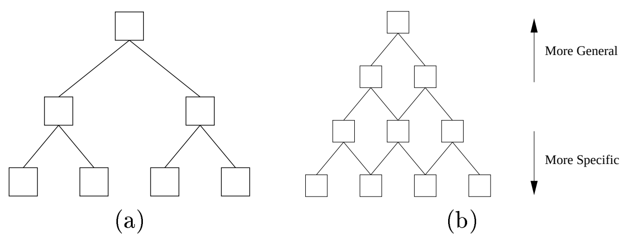



**Fig. 5.** The hypothesis space topologies used in our evaluation: (a) the tree topology and (b) the lattice topology. Note that we do not show the trivially most specific case, where nothing is acceptable.

For the purposes of our experiments, we developed an hypothesis-space based on the *tree topology* having 16 example nodes, giving a total of 30 possible constraints. We also generated an hypothesis space based on the *lattice topology* having 15 example nodes, giving a total of 120 possible constraints.

The extent to which the user was trying to be helpful was modeled by varying the number of examples that he would give to the computer when user-specified examples were required. An *unhelpful user* specified a single example to the computer, while a *helpful user* specified a set of examples. This set of user-specified examples is a subset of those examples from the hypothesis space that are consistent with the constraint the user is attempting to articulate. We select this subset with random cardinality and membership since doing so provides us with a mechanism for controlling the number of user-specified examples in a way which is representative of the generality of the constraint being articulated.

In the context of the hypothesis space based on the *tree topology* we compared the *50/50*, *Random* and *Least Generalization (Random)* strategies. The other two variants of the *Least Generalization* strategy, namely the *Least Generalization (Maximum Connectivity)* and *Least Generalization (50/50)* would behave in exactly way as *Least Generalization (Random)* the context of this hypothesis space topology, since there is never a situation where a principled choice could be made between alternative queries. In the context of the *lattice topology*, all strategies were compared.

The model of interaction used in the evaluation was based on that described in Section 2.1, varying the number of examples specified by the user depending on the extent to which he was being helpful. Each experiment involved counting the number of interactions, between the simulated user and our constraint acquisition system, required to acquire constraints from each of the hypothesis spaces described above. We define two types of *interaction*:

1. the user specifying one or more examples to the system, or
2. the system presenting a query to the user in an attempt to qualify an hypothesised generalization.

The total number of interactions was computed as the sum of the number of interactions used for user-specified examples (type (1) above) and the number of interactions for system-generated queries (type (2) above). Note that there is a difference between the number of examples that the user provides during the entire interaction and the number of interactions that involve the user specifying a set of examples. We count the latter, the number of interactions that involve the user specifying a set of examples, as a single interaction.

While not presented here, we have also run experiments counting the total number of examples that the user specified before the required constraint was acquired, which can be regarded as a measure of the work done by the user. We found that the number of examples that the user specified were similar for both types of user (helpful and unhelpful). This similarity is largely due to fact that the helpful user, as currently simulated, specifies random sets of examples. Obviously, this is not an intelligent approach for a user to adopt since some examples are redundant; in the sense that they do not help reduce the size of the version space. One current direction in our work is to associate with each possible example a measure of information gain which would reflect its usefulness in reducing the version space over examples already provided my the user. In this way, we can assist the user choose useful examples and reduce the total number of examples required to acquire a constraint. However, for the moment, we are concerned with reducing the length of the dialog between the user and computer. Thus, we acquired each constraint in each hypothesis space, recording the total number of interactions required to acquire it and its depth in the hypothesis space. Depth, in this context, is a measure of the generality of the constraint. The more general the constraint, the deeper it is in the hypothesis space.

The comparison presented below is based on averages over 25 runs of each experiment. In all the graphs, the x-axis represents the depth in the hypothesis space of the constraint being acquired.

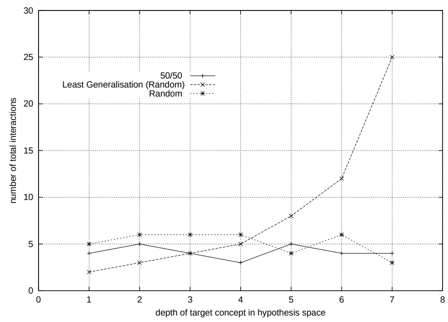
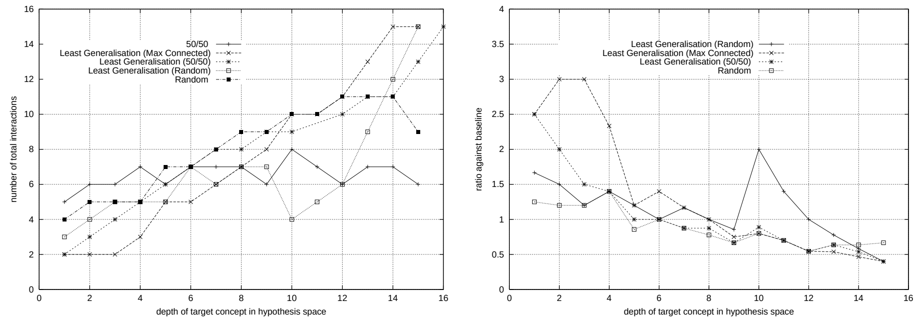



**Fig. 6.** Comparison of strategies on the tree topology when the user is not being helpful.

Figure 6 illustrates the average number of interactions required to acquire constraints at various depths in the hypothesis space. In this scenario, the hypothesis space is a (binary) tree and the user is not being helpful, only giving a single example at each point that a user-specified example is sought. It can clearly be seen that by far the best strategy to adopt is *50/50*. However, while the *Random* strategy is always beaten, the *Least Generalization* strategy performs well for less general constraints. We will return to this point later in the paper.

Figure 7(a) again illustrates the average number of interactions required to acquire constraints at various depths in the hypothesis space. In this scenario, the only change over the scenario illustrated in Figure 6 is that we have a *lattice* topology. Again, from Figure 7(a), it can be seen that the best strategy across the entire version space is 50/50. However, for constraints which are lower in depth in the hypothesis space, the *Least Generalization* strategies perform best. It is particularly interesting to see that the *Least Generalization (Random)* strategy does as well as *50/50* at most levels of the hypothesis space. However, it performed poorly on the deepest (most general) constraints.

Figure 7(b) presents a comparison of the the dialog lengths of each of the strategies in Figure 7(a), which shows the relative saving of the various strategies over *50/50*. On this graph, all points with a value greater than 1 represent an




(a) Dialog Length (b) Comparison against the 50/50 strategy

**Fig. 7.** Comparison of strategies on the lattice topology when the user is not being helpful.

improvement over the *50/50* baseline strategy. It is clear to see that the *Least Generalization (Random)* performs very well over most of the hypothesis space.

Figure 8 again illustrates the average number of interactions required to acquire constraints at various depths in the hypothesis space, for a *tree* topology. The user in this scenario is more helpful than in the previous two scenarios. Here the user is presenting a (random) set of examples. It can be seen clearly that the best strategy across the entire hypothesis space is *Least Generalization (Random)*. This is a significant improvement over the scenario illustrated in Figure 6. It is clear evidence that the choice of generalization strategy can be dramatically informed by knowledge of the nature of the interaction with the user.

Finally, Figure 9(a) illustrates the average number of interactions required to acquire constraints at various depths in the hypothesis space, for a *lattice* topology. Again the user in this scenario is helpful. It can be seen clearly that

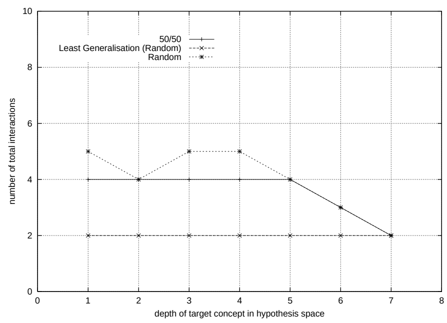
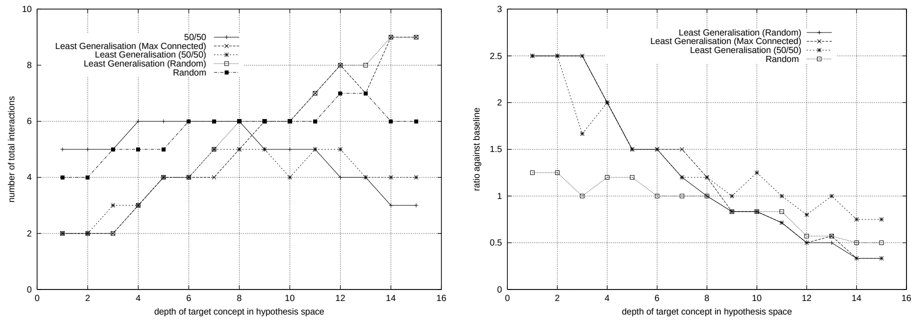



**Fig. 8.** Comparison of strategies on the tree topology when the user is being helpful.




(a) Dialog Length (b) Comparison against the 50/50 strategy

**Fig. 9.** Comparison of strategies on the lattice topology when the user is being helpful.

while 50/50 performs consistently well across the entire hypothesis space, the best strategy is again based on *Least Generalization*, in particular *Least Generalization (50/50)*. The improvement that this heuristic offers over 50/50 can be seen more clearly in Figure 9(b), which shows the relative saving of the various strategies over 50/50. At almost every depth, the *Least Generalization (50/50)* strategy saves on interactions over the regular 50/50 approach. This, again, is clear evidence that the choice of generalization strategy can be dramatically informed by knowledge of the nature of the interaction with the user. In general it can be seen that strategies based on *Least Generalization* are more powerful than *50/50* on constraints which are more specific in the version space.

**Table 1.** Comparison of learning rates against 50/50 for each strategy on each hypothesis space topology, for each degree of user helpfulness.

| *Heuristic* | *Tree* | *Lattice* | *Tree* | *Lattice* |
|---|---|---|---|---|
| | *One Ex.* | *One Ex.* | *Multiple Ex.* | *Multiple Ex.* |
| Random | -24% | -23% | -12% | -15% |
| Least Generalization | -103% | **-2%** | **44%** | -9% |
| Least Generalization + 50-50 | - | -23% | - | **20%** |
| Least Generalization + Max Conn. | - | -16% | - | -5% |

Table 1 presents a comparison of the learning rates of each of the generalization strategies discussed here over the existing 50/50 approach. The percentages in this table represent the savings in learning rate between each strategy compared against *50/50*. It can be seen that the *Random* strategy never beats *50/50* on any hypothesis space topology, regardless of the type of user from whom we are acquiring constraints. The performance of the various variants of the *Least Generalization* strategy are very interesting. While there was no benefit from using a *Least Generalization* strategy on the (binary) tree structured hypothesis space when the user is not being helpful, in almost every other situation, there was an improvement. In the case where we were acquiring constraints from an unhelpful user on the *lattice* structured hypothesis space, there was a very marginal difference between *Least Generalization* and *50/50* (2% in favor of the latter). However, once the user became helpful, there was a marked improvement in minimizing the number of interactions when using a *Least Generalization* approach. The most significant reason for this improvement is that when the user is being helpful, the set of user-specified examples allows us to reduce the depth in the version space of the required constraint; in other words, the required constraint becomes *relatively* more specific. We have clearly seen in all of the graphs above that *Least Generalization*-based strategies are more powerful on more specific constraints. As the user gives a set of examples, the net effect is that the required constraint becomes more specific in the new version space, after the set of user-specified examples has been taken into account.

These results are significant, since they demonstrate that the choice of generalization strategy is closely related to the degree to which the user is being helpful. The insights gained here inform our research agenda in user-friendly interactive constraint acquisition. Constraint acquisition is of critical importance to the constraint processing community, since without the ability to effectively acquire the problem practical application of the wealth of research in the field will be retarded.

## 5 Concluding Remarks

In many practical applications users often find it difficult to articulate their constraints. In this paper we presented an approach to interactive constraint acquisition based on techniques from the field of machine learning. We have begun studying the issues involved in interactively acquiring constraints and already make a number of novel contributions. In particular, the contributions of this paper are as follows:

1. We present an approach to acquiring single constraints from user-specified examples;
2. We present an interactive model for acquiring constraints;
3. We present alternative strategies for generating queries to present to the human user and offer some insights into choosing the best strategy for a particular hypothesis space topology;

We are currently developing the work reported here in a number of directions. In particular, some current directions we are exploring are:

- learning multiple constraints and entire CSPs;
- the benefits of hybridising consistency processing and version space learning for improved acquisition power;
- ways of assisting the user select good examples in order to minimize the number of user-specified examples;

- the effect of varying the number of examples that the user specifies on dialog length;
- the effect of varying the number of queries that the system presents to the user on dialog length and the number of user-specified examples.

We hope to make significant in-roads in this area in the short-term.

## Acknowledgments

The authors would like to thank Prof. Pat Langley of the Computational Learning Laboratory, at Stanford University, for his helpful discussions on this work.

Dr. O'Sullivan is supported by Enterprise Ireland, under their International Collaboration Programme (Grant No. IC/2001/058). Professor Freuder is supported by a Principal Investigator Award from Science Foundation Ireland.

## References

1. Eugene C. Freuder and Barry O'Sullivan. Generating tradeoffs for interative constraint-based configuration. In *Proceedings of the Seventh International Conference on Principles and Practice of Constraint Programming*, November 2001.
2. Eugene C. Freuder and Richard J. Wallace. Suggestion strategies for constraint-based matchmaker agents. In *Principles and Practice of Constraint Programming - CP98*, pages 192–204, October 1998.
3. Tom Mitchell. Version spaces: A candidate elimination approach to rule learning. In *Proceedings of the Fifth International Joint Conference in Artificial Intelligence*, 1977.
4. Tom Mitchell. Generalization as search. *Artificial Intelligence*, 18(2):203–226, 1982.
5. Tom Mitchell. Concept learning and the general-to-specific ordering. In *Machine Learning*, chapter 2, pages 20–51. McGraw Hill, 1997.
6. Barry O'Sullivan, Eugene C. Freuder, and Sarah O'Connell. Interactive constraint acquisition – position paper. In *Working Notes of the CP-2001 Workshop on User-Interaction in Constraint Satisfaction*, pages 73–81, December 2001.
7. Srinivas Padmanabhuni, Jia-Huai You, and Ghose Aditya. A framework for learning constraints. In *Proceedings of the Fourth Pacific Rim International Conference on Artificial Intelligence (PRICAI-96) Workshop on Induction of Complex Representations*, August 1996.
8. Francesca Rossi and Alessandro Sperduti. Acquiring both global and local preferences in interactive constraint solving via machine learning. In *Working Notes of the CP-2001 Workshop on User-Interaction in Constraint Satisfaction*, pages 83–95, December 2001. Paphos, Cyprus.
9. Claude Sammut and Ranan Banerji. Learning concepts by asking questions. In Ryszard S. Michalski, Jamie G. Carbonell, and Tom M. Mitchell, editors, *Machine Learning : An Artificial Intelligence Approach*, volume 2, chapter 7, pages 187–191. Morgan Kaufmann, Los Altos, California, 1986.