Constraint Acquistion with Recommendation Queries

Abderrazak Daoudi
U. Mohammed V
Rabat, Morocco

Younes Mechqrane
U. Mohammed V
Rabat, Morocco

Christian Bessiere
U. of Montpellier
France

Nadjib Lazaar
U. of Montpellier
France

El Houssine Bouyakhf
U. Mohammed V
Rabat, Morocco

Abstract

Constraint acquisition systems assist the non-expert user in modeling her problem as a constraint network. Most existing constraint acquisition systems interact with the user by asking her to classify an example as positive or negative. Such queries do not use the structure of the problem and can thus lead the user to answer a large number of queries. In this paper, we propose PREDICT&ASK, an algorithm based on the prediction of missing constraints in the partial network learned so far. Such missing constraints are directly asked to the user through recommendation queries, a new, more informative kind of queries. PREDICT&ASK can be plugged in any constraint acquisition system. We experimentally compare the QUACQ system to an extended version boosted by the use of our recommendation queries. The results show that the extended version improves the basic QUACQ.

1 Introduction

Constraint programming (CP) allows effective solving of combinatorial problems in many areas, such as planning and scheduling. However, modeling a combinatorial problem using constraints is a fastidious task that requires significant expertise in CP [Freuder, 1999]. To make constraint programming accessible to novices, several constraint acquisition systems have been introduced in the last decade [Bessiere et al., 2005; 2007; Beldiceanu and Simonis, 2012; Lallouet et al., 2010; Shchekotykhin and Friedrich, 2009]. These systems either need an expert to validate the learned model or need an exponential number of queries to converge on the target constraint network [Bessiere et al., 2015]. Recently, a system polynomial in terms of queries, called QUACQ, has been proposed [Bessiere et al., 2013]. QUACQ iteratively generates partial queries (that is, partial assignments of the variables) and asks the user to classify them. When the answer of the user is yes, QUACQ reduces the search space by removing constraints that reject the positive example. In the case of a negative answer, QUACQ focuses on a constraint in a number of queries logarithmic in the size of the example. This key component allows QUACQ to converge on the target network in a polynomial number of queries. Despite this good theoretical bound, QUACQ may require a lot of queries to learn the target constraint network, especially when the problem is highly structured and involves a large number of constraints. For instance, the user has to classify more than 8000 queries to get the Sudoku model.

The next challenge to constraint acquisition is to reduce the dialog length between the user and the learner, that is, to reduce the number of asked queries to get the target model. This paper presents a generic approach to constraint acquisition which is centered on the following question: Given the constraint graph learned so far, can we infer which new constraints are more likely to belong to the target constraint network? We formalize this question as a link prediction problem in the partial constraint graph learned so far. We also introduce a new concept of query, called recommendation query. Borrowing techniques from the link prediction field, a recommendation query asks the user whether or not a predicted constraint belongs to the target constraint network. To deal with recommendation queries, we propose a constraint recommender algorithm called PREDICT&ASK, which we plugged into the QUACQ constraint acquisition system leading to the P-QUACQ algorithm. We experimentally evaluated the benefit of our approach on several benchmark problems. The results show that P-QUACQ significantly improves the basic QUACQ algorithm in terms of number of queries.

The rest of the paper is organized as follows. Section 2 gives the necessary material to understand the technical presentation. Section 3 describes the constraint recommender algorithm. We illustrate the idea behind our constraint recommender algorithm through an example in Section 4. In Section 5, several predictor techniques are presented. Section 6 presents the experimental results we obtained when comparing P-QUACQ to the basic QUACQ. Section 7 presents the related work. Section 8 concludes the paper.

2 Background

The common knowledge shared between the user and the learner is the vocabulary. This vocabulary is repre-
sentenced by a (finite) set of variables \(X\) and domains \(D = \{D(x_1), \ldots, D(x_n)\}\) over \(\mathbb{Z}\). A constraint \(c\) is defined by a pair \((\operatorname{var}(c), \operatorname{rel}(c))\), where \(\operatorname{rel}(c)\) is the relation specifying which sequences of \(\operatorname{var}(c)\) values are allowed for the variables \(\operatorname{var}(c)\). \(\operatorname{var}(c)\) is called the scope of \(c\) and \(\operatorname{var}(c)\) the arity of \(\operatorname{rel}(c)\). Without loss of generality, we restrict ourselves to binary constraints. Combinatorial problems are represented with constraint networks. A constraint network is a set \(C\) of constraints on the vocabulary \((X, D)\). An example \(e\) is a (partial/complete) assignment on a set of variables \(\operatorname{var}(c) \subseteq X\). \(e\) is rejected by a constraint \(c\) (i.e., \(e \not\models c\)) iff \(\operatorname{var}(c) \subseteq \operatorname{var}(e)\) and the projection \(e[\operatorname{var}(c)]\) of \(e\) on \(\operatorname{var}(c)\) is not in \(c\). A complete assignment \(e\) of \(X\) is a solution of \(C\) iff for all \(c \in C\), \(e\) does not reject \(c\). We denote by \(\operatorname{sol}(C)\) the set of solutions of \(C\).

In addition to the vocabulary, the learner owns a language \(\Gamma\) of relations from which it can build constraints on specified sets of variables. A constraint basis is a set \(B\) of constraints built from the constraint language \(\Gamma\) on the vocabulary \((X, D)\). Formally, \(B = \{c \mid (\operatorname{var}(c) \subseteq X) \land (\operatorname{rel}(c) \in \Gamma)\}\).

In terms of machine learning, a concept is a Boolean function over \(D^X = \Pi_{x \in X} D(x)\), that is, a map that assigns to each example \(e \in D^X\) a value in \(\{0, 1\}\). We call target concept the concept \(f_T\) that returns \(1\) for \(e\) if and only if \(e\) is a solution of the problem the user has in mind. In a constraint programming context, the target concept is represented by a target network denoted by \(C_T\). A query \(\operatorname{Ask}(c)\), with \(\operatorname{var}(c) \subseteq X\), is a classification question asked to the user, where \(e\) is an assignment in \(D^\operatorname{var}(c) = \Pi_{x \in \operatorname{var}(c)} D(x)\). A set of constraints \(C\) accepts an assignment \(e\) if and only if there does not exist any constraint \(c \in C\) rejecting \(e\). The answer to \(\operatorname{Ask}(c)\) is yes if and only if \(C_T\) accepts \(e\).

In this paper we introduce a new kind of query, recommendation queries \(\operatorname{AskRec}(c)\), which ask the user whether or not the constraint \(c\) belongs to the target constraint network \(C_T\). It is answered yes if and only if \(c\) belongs to \(C_T\).

## 3 Predict&Ask Algorithm

In this section, we present our constraint recommender Predict&Ask algorithm. The idea behind this algorithm is to predict missing constraints in the partial network learned so far, and then to recommend the predicted constraints to the user through recommendation queries.

### 3.1 Description of Predict&Ask

The algorithm Predict&Ask takes as argument the set of constraints \(C\) learned so far, a relation \(r\), and the predictor \(\operatorname{score}\) that corresponds to the strategy used to assign a cost to a candidate constraint for recommendation. The algorithm uses the local data structure \(\Delta\) which contains all constraints that are candidate for recommendation.

Predict&Ask starts by initializing \(L\) to the empty set (line 1). The set \(L\) will contain the output of Predict&Ask, that is all constraints learned by prediction plus recommendation query. In line 4, we build the constraint graph \(G = (Y, E)\) restricted to the relation \(r\). The counter \(\#\text{No}\) counts the number of consecutive times recommendation queries have been classified negative by the user. It is initialized to zero at line 5. We put in \(\Delta\) all constraints that are candidate for recommendation.

In the main loop of Predict&Ask (line 7), for each iteration, we pick a constraint from \(\Delta\) such that its score is maximum (line 8). A constraint with a high score means that it is likely that this constraint belongs to the target constraint network. Hence, Predict&Ask asks a recommendation query on \((x, y), r\) (line 9). If the user says ‘yes’, \(\((x, y), r\)\) is a constraint of the target network. Hence, we put \(\((x, y), r\)\) in \(L\) (line 10). We also add the edge \((x, y)\) to \(E\) to be taken into account in the next iteration when computing the \(\operatorname{score}\). In line 12, we reinitialize \(\#\text{No}\) to zero. If the user says ‘no’, we remove from \(B\) the constraint \(\((x, y), r\)\) (line 14) and we increment \(\#\text{No}\) (line 15). The loop ends when \(\Delta\) is empty or when \(\#\text{No}\) reaches the given threshold \(\alpha\), and we return \(L\) (line 16).

### 3.2 Using Recommendation in QUACQ

Predict&Ask is a generic constraint recommender algorithm that can be plugged into any constraint acquisition system. In this section, we present P-QUACQ (Algorithm 2) where we incorporate Predict&Ask into the QUACQ system.

P-QUACQ initializes the constraint network \(C_L\) to the empty set (line 1). When \(C_L\) is unsatisfiable (line 3), the space of possible networks collapses because there does not exist any subset of the given basis \(B\) that is able to correctly classify the examples already asked to the user. In line 4, P-QUACQ computes a complete assignment \(e\) satisfying \(C_L\) and violating at least one constraint from \(B\). If such an example does not exist (line 5), then all constraints in \(B\) are implied by \(C_L\), and the algorithm has converged. Otherwise, we propose the example \(e\) to the user, who will answer by yes or no (line

---

**Algorithm 1: Predict&Ask**

**Input:** \(C\): a set of constraints, \(r\): a relation,

\(\operatorname{score} \in \{AA, LHN\}\): a prediction strategy

**Output:** \(L\): a set of predicted constraints

```plaintext
1 \(L \leftarrow \emptyset;\)
2 \(Y \leftarrow \bigcup \operatorname{var}(c) \text{ s.t. } (c \in C \land \operatorname{rel}(c) = r)\)
3 \(E \leftarrow \{(x, y) \mid c \in C \land \operatorname{rel}(c) = r \land \operatorname{var}(c) = (x, y)\}\)
4 \(G \leftarrow (Y, E)\)
5 \(\#\text{No} \leftarrow 0\)
6 \(\Delta \leftarrow \{(x, y), r\} \in B \mid (x, y) \in Y^2 \setminus E\)
7 \textbf{while} \(\Delta \neq \emptyset \land \#\text{No} < \alpha\) \textbf{do}
8 \hspace{1em} \textbf{pick} \((x, y), r\) \text{ in } \Delta \text{ that maximizes } \operatorname{score}((x, y), G)
9 \hspace{1em} \textbf{if} \: \operatorname{AskRec}((x, y), r) = \text{yes} \textbf{then}
10 \hspace{2em} \(L \leftarrow L \cup \{(x, y), r\}\)
11 \hspace{2em} \(E \leftarrow E \cup (x, y)\)
12 \hspace{2em} \#\text{No} \leftarrow 0\)
13 \hspace{1em} \textbf{else}
14 \hspace{2em} \(B \leftarrow B \setminus \{(x, y), r\} \mid r \subseteq r'\)
15 \hspace{2em} \#\text{No} ++
16 \textbf{return} \(L;\)
```

---

721
Algorithm 2: p-QUACQ = QUACQ + PREDICT&ASK

Input: score ∈ {AA, LHN}; a predictor strategy
Output: C_L; a set of learned constraints

1. C_L ← ∅;
2. while true do
3. if sol(C_L) = ∅ then return "collapse";
4. choose e in D^X accepted by C_L and rejected by B;
5. if e = nil then return "convergence on C_L";
6. if Ask(e) = yes then
7. B ← B \ κ_B(e);
8. else
9. c ← FindC(e, FindScope(e, ∅, X, false));
10. if c = nil then return "collapse";
11. else
12. C_L ← C_L \ {c};
13. C_L ← C_L ∪ PREDICT&ASK(C_L, rel(c), score);

6). If the answer is yes, we can remove from B the set κ_B(e) of all constraints in B that reject e (line 7). If the answer is no, we are sure that e violates at least one constraint of the target network C_T. We then call the function FindScope to discover the scope of one of these violated constraints. Here, FindScope acts in a dichotomous manner and asks a number of queries logarithmic in the size of the example. FindC selects which constraint with the given scope is violated by e (line 9). If no constraint is returned (line 10), this is a condition for collapsing as we could not find in B a constraint rejecting one of the negative examples. Otherwise, we know that the constraint e returned by FindC belongs to the target network C_T, we then add it to the learned network C_L (line 12). Note that FindScope and FindC functions are used exactly as they appear in [Bessiere et al., 2013]. Afterwards, we call PREDICT&ASK to mine the learned constraint network C_L in order to predict and recommend missing constraints that may belong to the target network. p-QUACQ updates C_L by adding all learned constraints (line 13).

3.3 Complexity Analysis

Let us now give the theoretical upper bound of the new constraint acquisition system p-QUACQ.

Theorem 1. Given a constraint basis B built from a language Γ of bounded arity, and a target network C_T, p-QUACQ uses O(C_T.(log|X| + Γ) + |B|) queries to prove convergence or to collapse.

Proof. By construction, p-QUACQ inherits the correctness of QUACQ, and thus, it always finishes by proving convergence or collapsing. As for its complexity, p-QUACQ asks partial queries (line 6 of p-QUACQ) and recommendation queries (line 9 of PREDICT&ASK). By construction, the number of partial queries in p-QUACQ is bounded above by the number of partial queries of pure QUACQ, that is, O(C_T.(log|X| + Γ) + |B|) [Bessiere et al., 2013]. Concerning recommendation queries, we know that they are asked on constraints that are in B and not in C_L (Algorithm 1, lines 6 and 8). Furthermore, a recommendation query cannot be asked twice on the same constraint as, whatever the answer, the constraint is put in C_L (yes answer, Algorithm 1, line 10 and Algorithm 2, line 13) or removed from B (no answer, Algorithm 1, line 14). As a result the number of recommendation queries asked by PREDICT&ASK is in O(|B|) and the number of queries asked by p-QUACQ is in O(C_T.(log|X| + Γ) + |B|).

4 An Illustrative Example

In this section, we illustrate our constraint recommender algorithm PREDICT&ASK through an example. Figure 1(a) shows the constraint network of the problem that the user has in mind. This problem involves 10 variables and 21 binary constraints. Two relations are used, noted r_1 and r_2 in Figure 1. Figure 1(b) shows the constraint network partially learned by QUACQ. Suppose that the last constraint learned using QUACQ was (x_1, x_2, r_1). At that point, we want to recommend potential constraints on which the relation r_1 may hold. PREDICT&ASK builds a partial network limited to the relation r_1 (Figure 1(c)), and then computes the set Σ of all candidate constraints that may belong to the target network. Σ = {((x_1, x_3), r_1), ((x_1, x_4), r_1), ((x_2, x_3), r_1), ((x_2, x_4), r_1), ((x_2, x_5), r_1), ((x_3, x_5), r_1)}. Then, PREDICT&ASK assigns to each candidate constraint in Σ a score. We sort the elements of Σ in decreasing order of their score. Suppose that we have the following order (((x_1, x_3), r_1), ((x_2, x_4), r_1), ((x_2, x_5), r_1), ((x_2, x_3), r_1), ((x_3, x_5), r_1), ((x_1, x_3), r_1)). Suppose that α = 1, which means that we have to exit PREDICT&ASK after one negative answer. We pick the first constraint ((x_1, x_3), r_1) in Σ, and we ask the user the recommendation query AskRec((x_1, x_3), r_1), which will be answered yes, as the constraint ((x_1, x_3), r_1) belongs to the target network. The other questions are as follows:

- AskRec((x_2, x_4), r_1) = yes (#No = 0)
- AskRec((x_2, x_5), r_1) = yes (#No = 0)
At the end, thanks to PREDICT&ASK three (out of four) constraints are added to the current constraint network (see Figure 1(d)).

5 Prediction Strategies

The way PREDICT&ASK computes the score has not been detailed in Section 3. In this section, we present the two techniques that we have used to predict missing constraints. Bessiere et al. (2014) have shown that when a constraint network has some structure, variables of the same given type are often involved in constraints with the same relation. Hence, we expect that when variable types are not known in advance, predicting type similarity or type proximity of variables could be done by prediction link techniques.

Link prediction in dynamic graphs is an important research field in data mining. Link prediction can be used for recommendation systems [Li and Chen, 2009], security domain [Krebs, 2002], social networks [Liben-Nowell and Kleinberg, 2003], and many other fields. Several techniques have been proposed in the literature for link prediction. All these techniques compute and assign a score to pairs of nodes \((x, y)\), based on the input graph and then produce a ranked list in a decreasing order of scores. They can be viewed as computing a measure of proximity or similarity between nodes \(x\) and \(y\), with respect to the network topology. Most of these techniques are based either on node neighborhood or on path ensemble [Lu and Zhou, 2010]. In our experiments we selected one link prediction technique representative of node-neighborhood-based techniques (Adamic/Adar –AA), and one representative of path-ensemble-based techniques (Leicht-Holme-Newman Index –LHN). Both of these techniques have a time complexity in \(O(n^2)\). We will see in our experiments that this never takes more than a few milliseconds.

5.1 Adamic-Adar Index (AA)

Adamic and Adar (2003) proposed a measure in the context of deciding when two personal home pages are strongly "related". They compute features of the pages and define the similarity index between two pages to be:

\[
\text{score}(x, y) = \sum_{z \in Z} \frac{1}{\log(|\text{frequency}(z)|)}
\]

where \(Z\) is the set of features shared by \(x\) and \(y\). This refines the simple counting of common features by weighting rarer features more heavily. This suggests the measure

\[
\text{score}(x, y) = \sum_{z \in N(x) \cap N(y)} \frac{1}{\log|N(z)|}
\]

where \(N(x)\) denotes the neighborhood of \(x\), that is, the set of variables with whom it shares a constraint.

5.2 Leicht-Holme-Newman Index (LHN)

Leicht-Holme-Newman (2006) proposed to compute vertex similarity, or proximity, based on the concept that two nodes are similar when their neighbors are similar. This index can be expressed into a matrix form as:

\[
S = 2m\lambda_1D^{-1}(I - \frac{\phi A}{\lambda_1})^{-1}D^{-1}
\]

where \(m\) is the number of links in the network, \(\lambda_1\) is the largest Eigenvalue of the adjacency matrix \(A\), \(D\) is a diagonal degree matrix, \(I\) is the identity matrix, and \(\phi (0 < \phi < 1)\) is a free parameter that assigns higher weights to shorter paths if it is closer to 0 and to longer paths if it is closer to 1 [Lu and Zhou, 2010]. In all our experiments we have set \(\phi\) to 0.5 to assign the same weight to both shorter and longer paths.

6 Experimental Evaluation

We made experiments to evaluate the impact of using PREDICT&ASK in constraint acquisition. We first present the benchmark problems we used for our experiments. Then, we report the results of acquiring these problems with the basic version of QUACQ, with a brute-force algorithm using only recommendation queries (denoted by ONLYREC), and with our p-QUACQ using the Adamic/Adar (AA) and Leicht-Holme-Newman (LHN) indexes to recommend constraints to the user. ONLYREC makes a brute-force use of recommendation queries: it asks recommendation queries on constraints from \(B\) and removes redundant constraints from \(B\) each time a new constraint is learned, until convergence is reached. Our tests were conducted on an Intel Core i5-3320M CPU @ 2.60GHz × 4 with 4 Gb of RAM.

6.1 Benchmark Problems

Radio Link Frequency Assignment Problem. The RLFAP is to provide communication channels from limited spectral resources [Cabon et al., 1999]. Here we build a simplified version of RLFAP that consists in distributing all the frequencies available on the base stations of the network. The constraint model has 36 variables with domains of size 36, and 210 binary constraints. We fed QUACQ and p-QUACQ with a basis of 1800 binary constraints taken from a language of 6 arithmetic and distance constraints.

Vessel Loading. Supply vessels transport containers from site to site. The deck area is rectangular. Containers are cubic, and are laid out in a single layer. All containers are positioned parallel to the sides of the deck. The contents of the containers determine their class. Certain classes of containers are constrained to be separated by minimum distances either along the deck or across the deck. The constraint model has 25 variables with domains of size 25, and 210 binary constraints. We fed QUACQ and p-QUACQ with a basis of 2610 binary constraints taken from a language of 6 arithmetic and distance constraints.

Murder. Someone was murdered last night, and you are summoned to investigate the murder. The objects found on the spot that do not belong to the victim include: a pistol, an umbrella, a cigarette, a diary, and a threatening letter. There are also witnesses who testify that someone had argued with the victim, someone left the house, someone rang the victim, and some walked past the house several times about the time the murder occurred. The suspects are: Miss Linda Ablaze,
We also observe that AA and LHN outperform Random for... We first observe that the number of queries asked by... and generating recommendation queries is not reported be-

directly... queries, the number... standard queries, the number... asked by the basic Q-
based method, LHN. However, constraints in bicliques cannot... all values of \( \alpha \), which means that their predictions are corre-
lated to the probability of having a constraint at the selected link. When predicting links with AA, we observe that cutoffs... hurt the acquisition: the smaller the cutoff, the greater the
to RLFAP. Thus, the results follow the same trend as on the

Table 1: \( p \)-QUACQ on RLFAP.

<table>
<thead>
<tr>
<th>( \alpha )</th>
<th>#query</th>
<th>#Ask</th>
<th>#AskRec</th>
<th>#no</th>
<th>#yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>-∞</td>
<td>1575</td>
<td>1575</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1575</td>
<td>1575</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1052</td>
<td>1575</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1052</td>
<td>1575</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Mr. Tom Burner, Ms. Lana Curious, Mrs. Suzie Dulles, and Mr. Jack Evilson. Each suspect has a different motive for the murder, including: being harassed, abandoned, sacked, promotion and hate. Under a set of additional clues given in the description, the problem who is the Murderer? And what was the motive, the evidence-object, and the activity associated with each suspect. The target network of Murder has 20 variables with domains of size 5, and 53 binary constraints. We fed QUACQ and \( p \)-QUACQ with a basis \( B \) of 1140 binary constraints based on the language \( \Gamma = \{ =, \neq, \geq, <, \leq, > \} \).

**Zebra problem.** The Lewis Carroll’s Zebra problem is formulated using 25 variables, with 5 cliques of \( \neq \) constraints and 14 additional constraints given in the description of the problem. We fed QUACQ and \( p \)-QUACQ with a basis \( B \) of 4450 unary and binary constraints taken from a language with 24 basic arithmetic and distance constraints.

### 6.2 Results

We compare QUACQ, ONLYREC, and \( p \)-QUACQ. For \( p \)-QUACQ we report results when predicting links with AA or LHN, without cutoff (i.e., \( \alpha = +\infty \)) and also with four values for the cutoff \( \alpha \) (from 1 to 4). We also report results when predicting links with a Random strategy, which serves as baseline selector as it randomly picks a candidate constraint from \( \Delta \), and recommends it to the user. For all our experiments we report the number of (standard) queries asked by the basic QUACQ, the number of (recommendation) queries asked by ONLYREC, and the number of queries asked by \( p \)-QUACQ. For \( p \)-QUACQ we report the number \#Ask of standard queries, the numbers \#AskRec of recommendation queries, the numbers \#no and \#yes of negative and positive recommendation queries (i.e., \#AskRec = \#no + \#yes), and the total number \#query of queries (i.e., \#query = \#Ask + \#AskRec). The time overhead of computing scores and generating recommendation queries is not reported because it takes a few milliseconds.

Table 1 reports the results of acquiring the RLFAP problem. We first observe that the number of queries asked by \( p \)-QUACQ is always significantly lower than with QUACQ or ONLYREC, whatever the way we predict links in \( p \)-QUACQ. We also observe that AA and LHN outperform Random for...
Table 2: P-QUAC on Vessel Loading.

<table>
<thead>
<tr>
<th>QUACQ</th>
<th>α</th>
<th>#query</th>
<th>#Ask</th>
<th>#AskRec</th>
<th>#no</th>
<th>#yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ONLYRec</td>
<td>1.0</td>
<td>2595</td>
<td></td>
<td></td>
<td>2595</td>
<td></td>
</tr>
<tr>
<td>P-QUACQ+Random</td>
<td>1.0</td>
<td>1505</td>
<td>864</td>
<td>641</td>
<td>501</td>
<td>140</td>
</tr>
<tr>
<td>P-QUACQ+AA</td>
<td>1.0</td>
<td>1385</td>
<td>889</td>
<td>496</td>
<td>357</td>
<td>139</td>
</tr>
<tr>
<td>P-QUACQ+LHN</td>
<td>1.0</td>
<td>1240</td>
<td>852</td>
<td>388</td>
<td>266</td>
<td>122</td>
</tr>
</tbody>
</table>

Table 3: P-QUAC on Murder.

<table>
<thead>
<tr>
<th>QUACQ</th>
<th>α</th>
<th>#query</th>
<th>#Ask</th>
<th>#AskRec</th>
<th>#no</th>
<th>#yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ONLYRec</td>
<td>1.0</td>
<td>1050</td>
<td></td>
<td></td>
<td>1050</td>
<td></td>
</tr>
<tr>
<td>P-QUACQ+Random</td>
<td>1.0</td>
<td>433</td>
<td>267</td>
<td>166</td>
<td>138</td>
<td>28</td>
</tr>
<tr>
<td>P-QUACQ+AA</td>
<td>1.0</td>
<td>496</td>
<td>380</td>
<td>116</td>
<td>100</td>
<td>16</td>
</tr>
<tr>
<td>P-QUACQ+LHN</td>
<td>1.0</td>
<td>497</td>
<td>406</td>
<td>91</td>
<td>78</td>
<td>13</td>
</tr>
</tbody>
</table>

Table 4: P-QUAC on Zebra.

<table>
<thead>
<tr>
<th>QUACQ</th>
<th>α</th>
<th>#query</th>
<th>#Ask</th>
<th>#AskRec</th>
<th>#no</th>
<th>#yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ONLYRec</td>
<td>1.0</td>
<td>1195</td>
<td></td>
<td></td>
<td>1195</td>
<td></td>
</tr>
<tr>
<td>P-QUACQ+Random</td>
<td>1.0</td>
<td>498</td>
<td>428</td>
<td>70</td>
<td>40</td>
<td>30</td>
</tr>
<tr>
<td>P-QUACQ+AA</td>
<td>1.0</td>
<td>625</td>
<td>4142</td>
<td>4142</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>P-QUACQ+LHN</td>
<td>1.0</td>
<td>625</td>
<td>4142</td>
<td>4142</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

Such a structure is perfectly well detected both by AA and LHN. This explains that they behave the same and that the shorter the cutoff, the better.

This experimental analysis clearly shows that the use of prediction strategies with recommendation queries can significantly reduce the number of queries asked to the user. The brute-force use of recommendation queries (ONLYREC) is always close to the worst case (i.e., close to |B| queries). The AA prediction strategy seems to be particularly well-suited to problem containing cliques of constraints, whereas the LHN can be highly efficient to predict biclique structures.

7 Related Work

Several papers have already proposed to use the structure of the constraint graph to decrease the number of examples needed to learn the target constraint network. Beldiceanu and Simonis (2012) have proposed MODELSEEKER, a passive constraint acquisition system devoted to problems having a regular structure. MODELSEEKER learns global constraints from the global constraints catalog (Beldiceanu et al., 2007) whose scopes are the rows, the columns, or any other structural property MODELSEEKER can capture. The counterpart is that it misses any constraint that does not belong to one of the structural patterns it is able to capture. Bessiere et al. (2014) introduced a new concept of query, called generalization query. By using some background knowledge, namely types of variables, a generalization query asks the user whether or not a learned constraint can be generalized to other scopes of variables of the same types as those of the learned constraint. The drawback of such queries is that they require types of variables to be provided by the user. To overcome this weakness, Daoudi et al. (2015) have proposed to learn types of variables during the constraint acquisition process, and then to use the learned types to generate generalization queries. The advantage of such an approach is that there is no need for the user to provide the types. Of course learning the types requires extra queries that were not needed when types are given for free at the beginning of the learning process. In addition, generalization queries do not work on all problems. They work only for the problems for which variables can be grouped into types.

By contrast, in our work, recommendation queries are generic and do not require any background knowledge to be generated. By using techniques borrowed from link prediction in dynamic graphs, we infer constraints that are more likely to belong to the target constraint network, and that are validated by asking recommendation queries to the user.

8 Conclusion

We have proposed a new kind of queries, called recommendation queries. To deal with these queries, we have proposed a generic constraint recommender algorithm, PREDICT&ASK, which uses techniques borrowed from link prediction to predict constraints that are likely to belong to the target network. Finally, we have plugged PREDICT&ASK into QUACQ to have a boosted version called P-QUACQ. Our experiments on several benchmark problems show that our new technique outperforms the basic QUACQ. An interesting direction would be to use a reinforcement learning to decide on the use of neighborhood-based predictions or path-ensemble-based predictions.
References


