Constraint Acquisition 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-
A constraint network is a set of constraints on the vocabulary \((X, D)\). An example \(e\) is a (partial/complete) assignment on a set of variables \(\forall e \subseteq X\). If \(e\) is rejected by a constraint \(c\) (i.e., \(e \not\models c\)), \(e\) is rejected by the constraint \(\forall c \in C\). A complete assignment \(e = \{x, y\} \subseteq X\) is a solution of \(C\), iff for all \(c \in C\), \(e\) does not reject \(c\). We denote by \(sol(C)\) 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 of constraints built from the constraint language \(\Gamma\) on the vocabulary \((X, D)\). Formally, \(B = \{e \mid (\forall e \subseteq X) \wedge (\forall rel(e) \in \Gamma)\}\).

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 \(Ask(e)\), with \(\forall e \subseteq X\), is a classification question asked to the user, where \(e\) is an assignment in \(D^{\forall} = \Pi_{x \in \forall} D(x)\). 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 \(Ask(e)\) is yes if and only if \(C\) accepts \(e\).

In this paper we introduce a new kind of query, recommendation queries \(AskRec(c)\), 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 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 \#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 score. In line 12, we reinitialize \#No to zero. If the user says ‘no’, we remove from \(B\) the constraint \((x, y), r\) (line 14) and we increment \#No (line 15). The loop ends when \(\Delta\) is empty or when \#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...
Theorem 1. Given a constraint basis $B$ built from a language $\Gamma$ of bounded arity, and a target network $C_T$, $P$-QUACQ uses $O(C_T. \log |X| + \Gamma) + |B|$ 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, $O(C_T. \log |X| + \Gamma) + |B|$ [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 $O(C_T. (\log |X| + \Gamma) + |B|)$. \qed

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 $\Delta$ of all candidate constraints that may belong to the target network. $\Delta = \{((x_1, x_3), r_1), ((x_1, x_4), r_1), ((x_2, x_3), r_1), ((x_2, x_4), r_1), ((x_2, x_5), r_1), ((x_3, x_5), r_1)\}$. Then, PREDICT&ASK assigns to each candidate constraint in $\Delta$ a score. We sort the elements of $\Delta$ 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 $\alpha = 1$, 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 $\Delta$, 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:

- $AskRec((x_2, x_4), r_1) = yes$ ($\#No = 0$)
- $AskRec((x_2, x_5), r_1) = yes$ ($\#No = 0$)}
recommendation systems [Li and Chen, 2009], security domain in data mining. Link prediction can be used for predicting type similarity or type proximity of variables could we expect that when variable types are not known in advance, 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 \(O(n^3)\). We will see in our experiments that this never takes more than a few milliseconds.

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.

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:

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

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:

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

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 \(\times\) 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 cuboid, 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 murder occurred. The suspects are: Miss Linda Ablaze,
We also observe that AA and LHN outperform Random for the murder problem. We first observe that the number of queries asked by the basic Q\#AskRec serves as baseline selector as it randomly picks a candidate constraint from the description of the problem. We fed Q\#AskRec and Q\#AskRec with a basis \( B \) of 1440 binary constraints based on the language \( \Gamma = \{ =, \neq, >, = \} \). 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 Q\#AskRec and Q\#AskRec 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 Q\#AskRec, Q\#AskRec, and Q\#AskRec. For Q\#AskRec 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 Q\#AskRec, the number of (recommendation) queries asked by Q\#AskRec, and the number of queries asked by Q\#AskRec. For Q\#AskRec we report the number \#AskRec of standard 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 = \#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 Q\#AskRec is always significantly lower than with Q\#AskRec or Q\#AskRec, whatever the way we predict links in Q\#AskRec. We also observe that AA and LHN outperform Random for all values of \( \alpha \), which means that their predictions are correlated 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 number of queries required for convergence. On the contrary, Q\#AskRec + LHN is better with cutoff: it reaches its lower number of queries to learn the RLFAP network when \( \alpha = 2 \) (851 queries instead of 1653 for basic Q\#AskRec and 1575 for Q\#AskRec). The good performance of LHN on RLFAP can be explained by the structure of that problem. The RLFAP structure contains bicliques and cliques. The constraints that belong to the same clique can be easily predicted by both the neighborhood-based method, AA, or the path-ensemble-based method, LHN. However, constraints in bicliques cannot be predicted by AA because variables of the same constraint do not share any neighbor (see Figure 2a).

Table 2 reports the results on the Vessel Loading problem. The structure of this problem is quite similar to the structure of RLFAP. Thus, the results follow the same trend as on the RLFAP (Q\#AskRec + LHN with \( \alpha = 2 \) is the best). However, we see that as opposed to the RLFAP, Q\#AskRec + AA benefits from the cutoffs.

Table 3 reports the results on the Murder problem. The structure of that problem is essentially composed of cliques, as we can see in Figure 2b. In this case, we observe that Q\#AskRec + AA with a cutoff equal to 1 is the best. It requires 367 queries to get the model instead of 585 queries for Q\#AskRec and 1050 for Q\#AskRec. The good performance of the AA predictor can be explained by the fact that neighborhood-based predictors are effective in detecting bicliques.

Table 4 reports the results on the Zebra problem. Again, Q\#AskRec with AA or LHN predictors outperforms Q\#AskRec, Q\#AskRec, and Q\#AskRec + Random. When comparing AA to LHN, we observe that, interestingly, Q\#AskRec + AA and Q\#AskRec + LHN give exactly the same results for all values of the cutoff \( \alpha \). This can be explained by the fact that only the relation \( \neq \) shows a structure in the network, and that structure is such that all cliques are isolated, as illustrated in Figure 2c.

### Table 1: P-Q\#AskRec on RLFAP.

<table>
<thead>
<tr>
<th>( \alpha )</th>
<th>#query</th>
<th>#AskRec</th>
<th>#AskRec</th>
<th>#yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1653</td>
<td>1575</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>964</td>
<td>560</td>
<td>404</td>
<td>322</td>
</tr>
<tr>
<td>3</td>
<td>1017</td>
<td>676</td>
<td>341</td>
<td>268</td>
</tr>
<tr>
<td>4</td>
<td>1129</td>
<td>817</td>
<td>312</td>
<td>250</td>
</tr>
<tr>
<td>5</td>
<td>1281</td>
<td>1015</td>
<td>268</td>
<td>229</td>
</tr>
<tr>
<td>6</td>
<td>1553</td>
<td>1390</td>
<td>183</td>
<td>161</td>
</tr>
<tr>
<td>7</td>
<td>1028</td>
<td>784</td>
<td>244</td>
<td>178</td>
</tr>
<tr>
<td>8</td>
<td>1229</td>
<td>1055</td>
<td>174</td>
<td>128</td>
</tr>
<tr>
<td>9</td>
<td>1252</td>
<td>1087</td>
<td>174</td>
<td>114</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 is who was 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 Q\#AskRec and Q\#AskRec with a basis \( B \) of 1140 binary constraints based on the language \( \Gamma = \{ =, \neq, >, = \} \). 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 Q\#AskRec and Q\#AskRec with a basis \( B \) of 4450 unary and binary constraints taken from a language with 24 basic arithmetic and distance constraints.
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 problems 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


