Learning Qualitative Constraint Networks

Malek Mouhoub[1]
University of Regina 3737 Wascana Parkway, Regina SK S4V 0A2, Canada
mouhoubm@uregina.ca
https://orcid.org/0000-0001-7381-1064

Hamad Al Marri
University of Regina 3737 Wascana Parkway, Regina SK S4V 0A2, Canada
almarrih@uregina.ca

Eisa Alanazi
Um Al Qura University Aif Road, 21955, Mecca, Makkah Province, Saudi Arabia
eaanazi@uqu.edu.sa

Abstract

Temporal and spatial reasoning is a fundamental task in artificial intelligence and its related areas including scheduling, planning and Geographic Information Systems (GIS). In these applications, we often deal with incomplete and qualitative information. In this regard, the symbolic representation of time and space using Qualitative Constraint Networks (QCNs) is therefore substantial.

We propose a new algorithm for learning a QCN from a non expert. The learning process includes different cases where querying the user is an essential task. Here, membership queries are asked in order to elicit temporal or spatial relationships between pairs of temporal or spatial entities. During this acquisition process, constraint propagation through Path Consistency (PC) is performed in order to reduce the number of membership queries needed to reach the target QCN. We use the learning theory machinery to prove some limits on learning path consistent QCNs from queries. The time performances of our algorithm have been experimentally evaluated using different scenarios.

2012 ACM Subject Classification Computing methodologies → Temporal reasoning

Keywords and phrases Temporal Reasoning, Qualitative Constraint Network (QCN), Constraint Learning, Path Consistency, Constraint Propagation

Digital Object Identifier 10.4230/LIPIcs.TIME.2018.19

1 Introduction

Reasoning with time and space is a fundamental task in artificial intelligence and its related areas including scheduling, planning, Geographic Information Systems (GIS) and computational linguistics. Handling the qualitative aspects of time and space is an important matter when designing such systems especially when dealing with incomplete information. Several research works have therefore been proposed in order to represent and reason on symbolic temporal and spatial information. One of the most known approaches is the Allen Algebra [1], based on the notion of time intervals and binary relations on them. A time interval I is an ordered pair (I^-, I^+) such that $I^- < I^+$, where I^- and I^+ are points on the time line. There are thirteen basic relations (Allen primitives) that can hold between intervals. A binary relation between two intervals is represented by the disjunction of some

[1] Natural Sciences and Engineering Research Council of Canada (NSERC), RGPIN-2016-05673

Allen primitives and expresses the incompleteness of the temporal information. Any problem under temporal constraints can be converted into an Interval Algebra (IA) network (also called qualitative temporal network) where nodes correspond to intervals and each arc represents the binary relation between the corresponding intervals. Note that an IA network is a particular case of a Qualitative Constraint Networks (QCN) which is a general network for representing problems under qualitative temporal or spatial constraints. Given an IA network, the main reasoning task is to decide its consistency and return one or more solutions (consistent scenarios satisfying all the temporal constraints) if it is the case. These tasks can be achieved with a backtrack search algorithm enhanced with constraint propagation techniques.

One of the challenges when dealing with problems under temporal or spatial constraints is the modelling task. This latter requires some expertise in temporal and spatial representation and reasoning, such as a good knowledge of the Allen Algebra. In this regard, we propose a new algorithm for learning a QCN network from a non expert with a focus on the IA network. The learning process includes different cases where querying the user is an essential task. Here, membership queries are asked in order to elicit the temporal or spatial relationships between pairs of temporal or spatial entities. During this acquisition process, constraint propagation is performed in order to reduce the number of membership queries needed to reach the target QCN. The time performance of our algorithm has been experimentally evaluated using different randomly generated scenarios. Note that, while the focus of this paper is on temporal constraints, our proposed algorithm can be easily generalized to any QCN.

2 Qualitative Constraint Networks (QCNs)

A Qualitative Constraint Network (QCN) is a pair (V, C) in which V is a finite set of variables representing temporal or spacial entities and C is a finite set of constraints on these variables. Each constraint c_i is expressed as a disjunction of binary relations between the involved variables. Each of these relations is defined on a language set $\mathcal{B} = \{b_1, b_2, \ldots, b_p\}$ where $p > 0$. A particular case of such networks is the IA network [1] where V is a set of temporal events, each representing an assertion over a time interval; and $\mathcal{B}$ is the set of Allen primitives depicted in Table 1 [2]. For instance, let us consider two temporal events, E_1 and E_2. The following constraint expresses the fact that both events are mutually exclusive: E_1 $(B \vee Bi)$ E_2. Note that a universal relation, corresponding to the disjunction of all relations within $\mathcal{B}$ (I, representing the 13 Allen primitives in the case of the Interval Algebra), is used to express the fact that there is no constraint between the involved entities (the knowledge on such constraint is completely unknown).

Given a QCN, the main reasoning task that can be performed consists of checking the consistency of the network and, if it is the case, returns one or more consistent scenarios satisfying all the constraints of the related problem. This task can be done using a backtrack search algorithm. Constraint propagation, before and during search, can be enforced in order to prevent late failure earlier. The most known constraint propagation technique used in QCN networks and especially in IA networks is Path Consistency (PC) [8, 14, 15, 20]. This technique (also called transitive closure) applies local consistency on every subset of three variables which results in removing some inconsistent relations from the network. This

[2] Note that, six of the seven primitives listed have inverse relations. This will bring the total of number of possible relations to 13.

Table 1 Allen's primitives.

Relation	Symbol	Inverse	Meaning
X precedes Y	B	Bi	XXXX YYYY
X equals Y	E	E	XXXX YYYY
X meets Y	M	Mi	XXXXYYYY
X overlaps Y	O	Oi	XXXX YYYY
X during Y	D	Di	XXXX YYYYYYYY
X starts Y	S	Si	XXXX YYYYYYYY
X finishes Y	F	Fi	XXXXX YYYYYYYYY

will lead to the reduction on the search space. Assuming X, Y and Z are three temporal events, if X precedes Y and Y meets Z, then it must be that X precedes Z as well. Any other relation belonging to the constraint between X and Z should be removed. Local consistency is enforced using a $|\mathcal{B}|^2$ composition table (13 x 13 composition table between Allen's primitives in the case of IA networks). Note that PC can also infer new relations by refining universal relations into a more specific one. For instance, if we take the previous example and assume that there is no constraint between Y and Z then PC will infer a new relation (precedes) between these two events.

Partial Path Consistency (PPC) is an extension of PC based on chordal graphs. A chordal graph is a graph such that, for every connected four vertexes and higher there must be one chordal edge [8]. A chordal graph is also called triangulated graph since the graph can be printed in a map in which the vertices are connected in triangle shapes. Partial Path Consistency assures that the chordal graph is consistent such as when applying regular PC [8]. Partial Path Consistency is much efficient than PC when working with chordal graphs. It is useful in case there are universal edges in the graph that can be removed to form a chordal graph.

3 Constraint Acquisition

Constraint acquisition is the process of inducing, from a set of examples, a constraint network representing a given problem. The main goal of constraint acquisition algorithms is to minimize the number of examples needed to converge to a target constraint network. In this regard, several strategies have been explored and a passive learning algorithm (CONACQ.1 or the passive version of the CONACQ acquisition system [7]) using unit propagation to speed up the convergence test has been proposed in [5]. In [6], a proposed active constraint acquisition technique, called QUACQ, works by asking the user to classify partial queries (those queries not involving all variables) in addition to membership queries (where the user is asked to answer "yes" or "no" depending on whether the provided assignment is a solution or not). Partial queries are asked when given a negative example and allow QUACQ to converge in a number of queries that is logarithmic in the number of variables. An extension to this latter algorithm has been proposed in [3] where multiple constraints are learned (and not only one as in QUACQ) using Minimal Unsatisfiability Subsets (MUS).

In [7], the active version of the CONACQ architecture, called CONACQ.2, is proposed as shown in Algorithm 1. CONACQ.2 starts with a bias B (similar to our language set $\mathcal{B}$) and its corresponding background language K. K is a set of declarative rules (definite

Algorithm 1 CONACQ.2 [7].

```
1:  procedure CONACQ.2
2:  Input: a bias $B$, a background knowledge $K$, a strategy $Strategy$
3:  Output: a clausal theory $T$ encoding the target network
4:      $T \leftarrow \emptyset$; $converged \leftarrow false$; $N \leftarrow \emptyset$
5:      while $\neg converged$ do
6:          $q \leftarrow QueryGeneration(B, T, K, N, Strategy)$
7:          if $q = nil$ then
8:              $converged \leftarrow true$
9:          else
10:             if $Ask(q) = no$ then
11:                 $T \leftarrow T \wedge (\bigvee_{c \in K(q)} a(c))$
12:             else
13:                 $T \leftarrow T \wedge \bigwedge_{c \in K(q)} \neg a(c)$
14:             end if
15:         end if
16:     end while
         return $T$
17: end procedure
```

Horn clauses) expressing some properties (such as the transitivity property) that can be used to enforce local consistency between constraints. In the case of QCNs, the set K basically corresponds to the composition table we mentioned in the previous section. Given these two inputs B and K, in addition to a given strategy to follow for generating the queries, the algorithm iterates by asking the user a new generated query at each time and expands the theory set T (initially set to the empty set) according to the answer provided. More precisely, if the user answers "no" to a given query q then the algorithm removes all the concepts that support this query (as shown in line 11 of the Algorithm). In case the answer is "yes" then all the concepts rejecting q must be removed (as per line 13 of the Algorithm). The algorithm terminates when there is no query to generate. The target network, in the form of the clausal theory T is then returned.

4 Proposed Learning Algorithm for QCNs

In this section, we present an algorithm for learning QCNs in the particular case of AI networks. Note that our algorithm can be easily generalized to any QCN. We consider the following three cases, each with a different target QCN.

1. Learning consistent and complete scenarios: the target QCN is a complete graph where each edge (constraint) contains exactly 1 relation (Allen primitive in the case of IA networks).
2. Learning consistent but incomplete scenarios: same as case 1 but in the case of incomplete graphs (some constraints do not exist between pairs of variables).
3. Learning QCNs: the target QCN is a graph (can be complete) where each edge contains 1 or more primitives (each constraint has one or more relations). Note that, in this particular case we are learning a QCN problem rather than a consistent scenario (solution) as it is in cases 1 and 2.

Following CONACQ.2 [7], our algorithm learns through membership queries and uses PC to reduce the number of queries. Note that when using PC for case 3, our learning algorithm will return a path consistent QCN problem.

4.1 Proposed Learning Algorithm

The algorithm starts with all the constraints completely unknown (corresponding to the universal relation I in the case of IA networks). The user is then asked a membership query for each relation within each constraint. If the answer is "yes" for a given query, the corresponding constraint will be replaced by the relation that has been confirmed (all the other relations will be eliminated). Otherwise (if the answer is "no"), the related relation will be removed from the constraint. After every query, PC is applied on the graph to remove path inconsistent relations due to this recent update. This will reduce the number of relations per constraint which will reduce the number of subsequent queries and helps getting the target QCN scenario (solution) sooner.

The pseudo-code of our method is listed in Algorithm 2. G_t is initially set to a complete constraint graph with universal relations. $\mathcal{B}$ and CT are respectively the set of possible relations and composition table of the QCN to learn. *QueryGeneration* is a function that generates a membership query each time it is called. We use two implementations of this function. In the first one, the constraint and its related relation (to confirm) are picked randomly. In the second implementation, the relations to confirm are picked according to the first fail principle used when solving general Constraint Satisfaction Problems (CSPs) [9]. In this regard we use the ordering heuristics proposed in [20], namely "weight" and "cardinality". The idea behind the "weight" heuristic is to quantify the restriction imposed by a relation, when assigned to a given edge, on the temporal constraints of the other edges. The "cardinality" heuristic is a special case of the "weight" heuristic when considering that all relations have the same weight (equal to 1). The algorithm iterates by processing a query at each time, until a solution (target QCN) is found or a path inconsistency is detected.

Given that the constraint graph is not necessarily complete, our algorithm for case 2 differs from the previous one, as follows. If the answer is "yes" for a given query, the relation is confirmed but we do not remove the other relations as the corresponding constraint can be the universal relation, I. Instead, we ask the user a second query for the same constraint in order to check if this latter is I. If the answer is "yes" then the constraint is confirmed to be universal, otherwise (the answer is "no") we replace the constraint by the relation confirmed with the first query. The rationale is that, in case 2 we either have a single relation or a universal relation for each constraint.

In case 3, given that the number of relations per constraint varies from 1 to $|\mathcal{B}|$ (13 in the case of IA networks) then we will have to ask the user subsequent queries for the same constraint regardless of the answer. More precisely, if the answer is "yes" then we need to ask the user for the remaining relations as we can have more than one relation per constraint.

4.2 Dealing with Inconsistent Answers

As stated in Algorithm 2, our learning method collapses (fails to return the target QCN) if PC detects an inconsistency after one of the constraints becomes empty (has no relations). In this case, we need to identify the query that has been answered incorrectly. We address this task using a backtrack search algorithm to go backward and let the user confirm previous answers until we reach the state where the user changes the answer of a query. Given that we can have more than one incorrect answer, every time a response to a query is changed, we

Algorithm 2 Learning QCN for Case 1.

```
procedure LearningQCN
Input: a language set $\mathcal{B}$, a composition table $CT$
Output: a target QCN $G_t$
    $G_t \leftarrow$ complete graph with universal relations
    $q \leftarrow QueryGeneration(G_t)$
    while $q \neq nil$ do
        $r \leftarrow Relation(q)$
        if $Ask(q) =$ "$yes$" then
            $ConfirmRelation(G_t, r)$
        else
            $RemoveRelation(G_t, r)$
        end if
        $status \leftarrow PC(G_t, CT)$
        if $status =$ "$inconsistent$" then
            return "$collapse$"
        end if
        $q \leftarrow QueryGeneration(G_t)$
    end while
    return $G_t$
end procedure
```

resume the normal querying starting back from the state where the user fixed the incorrect answer. For example, assume that after a query q_j is answered, an inconsistency is detected. Our algorithm will then backtrack until it identifies the query q_m causing this inconsistency. The user will then change the answer and our learning algorithm will resume from query q_{m+1}. The pseudo code of our method is listed in Algorithm 3. This algorithm is very similar to Algorithm 2 except that we save the query and the current G_t at each time in the stack s.

5 Theoretical Limits on Learning QCNs

In this section we use the learning theory machinery to prove some limits on learning Path Consistent QCNs from queries. In particular, we are interested in the number of queries required by the best possible algorithm. The problem addressed in this work can be viewed as a concept learning problem [2]. In concept learning, a concept is a subset of a given universe $\mathcal{X}$ and a concept class $\mathcal{C}$ is a set of concepts. We assume the user has a hidden target $c^* \in \mathcal{C}$ and we try to exactly identify it with the minimum number of queries. There are different types of queries and we are interested in what is known as membership queries [2, 12]. In learning with membership queries, the learning algorithm, having access to the set of concepts $\mathcal{C}$, picks an instance $x \in \mathcal{X}$ and asks the user *"Is $x \in c^*$?"*. The answer is either "yes" or "no". This process continues until the algorithm exactly identifies c^*. The main challenge here is to have a sequence of instances $x, x', \ldots$ of minimum size that exactly identifies any hidden concept. Let $\mathcal{I} = \{1, 2, \ldots, n\}$ be the set of entities and $\mathcal{B} = \{b_1, b_2, \ldots, b_p\}$ be the set of $p > 0$ relations. In this work, we are interested in the learnability of the three cases of QCNs presented in the previous Section and defined over $\mathcal{I}$ and $\mathcal{B}$ as follows:

1. The set of complete scenarios $\mathcal{Q}_{cmp}$.
2. The set of incomplete scenarios $\mathcal{Q}_{inc}$.
3. The set of all QCNs $\mathcal{Q}_{all}$.

Algorithm 3 Learning with Mistakes.

```
1: procedure LearningWithMistakesQCN
2: Input: $\mathcal{B}$: Language set. $CT$: Composition Table
3: Output: $G_t$ : target QCN
4:     $G_t \leftarrow$ complete graph with universal relations
5:     $s \leftarrow \emptyset$
6:     $q \leftarrow QueryGeneration(G_t)$
7:     $stackpush(s, < G_t, q >)$
8:     while $q \neq nil$ do
9:         $r \leftarrow Relation(q)$
10:        if $Ask(q) =$ "$yes$" then
11:            $ConfirmRelation(G_t, r)$
12:        else
13:            $RemoveRelation(G_t, r)$
14:        end if
15:        $status \leftarrow PC(G_t, CT)$
16:        while $status =$ "$inconsistent$" do
17:            $stackpop(s, < G_t, q >)$
18:            if $Ask(q) =$ "$yes$" then
19:                $ConfirmRelation(G_t, r)$
20:            else
21:                $RemoveRelation(G_t, r)$
22:            end if
23:            $status \leftarrow PC(G_t, CT)$
24:        end while
25:        $q \leftarrow QueryGeneration(G_t)$
26:        $stackpush(s, < G_t, q >)$
27:    end while
28:    return $G_t$
29: end procedure
```

Clearly, $\mathcal{Q}_{cmp} \subsetneq \mathcal{Q}_{inc} \subsetneq \mathcal{Q}_{all}$. Let $\mathcal{S}$ be the set of two-element subsets of $\mathcal{I}$. An instance x is a pair (s, b) where $s \in \mathcal{S}$ and $b \in \mathcal{B}$. For simplicity, we write the triple (i, j, b) to denote the instance (s, b) where $s = \{i, j\}$. The instance space $\mathcal{X}$ is then defined simply as $\mathcal{X} = \mathcal{S} \times \mathcal{B}$ and clearly $|\mathcal{X}| = \binom{n}{2} \times p$. A concept c is a subset of $\mathcal{X}$ or equivalently it is a mapping from $\mathcal{X}$ to $\{0, 1\}$ where $c(x) = 1$ iff $x \in c$ for any $x \in \mathcal{X}$. For a set $\mathcal{Q}$ of QCNs, c is representable by $\mathcal{Q}$ if there exists a QCN $N \in \mathcal{Q}$ where $c(x) = 1$ if and only if x holds in N. The concept class $\mathcal{C}$ is then defined as the set of all concepts that are representable by $\mathcal{Q}$. We consider the three concept classes $\mathcal{C}_{cmp}, \mathcal{C}_{inc}$ and $\mathcal{C}_{all}$ that represent respectively the set of concepts that are representable by the set of QCNs in $\mathcal{Q}_{cmp}$, $\mathcal{Q}_{inc}$ and $\mathcal{Q}_{all}$. One of the important parameters in learning theory is the teaching dimension [11]. The teaching dimension of a concept c w.r.t. a class $\mathcal{C}$, TD$(c, \mathcal{C})$, is the smallest number of examples (or instances) that distinguishes c from every other concept in the class. The teaching dimension of a class $\mathcal{C}$ (denoted as TD$(\mathcal{C})$) is the teaching dimension of the hardest concept to teach, i.e., TD$(\mathcal{C}) = \max_{c \in \mathcal{C}}$ TD$(c, \mathcal{C})$ [11]. It is known that the number of membership queries required by the best possible algorithm is lower bounded by TD [11]. Therefore, any learning algorithm would need at least TD queries in the worst case scenario. In the following, we show the teaching dimension of the three classes of QCNs. We believe such results would give some insights into the learnability of QCNs in general.

▶ **Proposition 1.** $\mathrm{TD}(\mathcal{C}_{cmp}) = \binom{n}{2}$.

Proof. For the upper bound, every concept $c \in \mathcal{C}_{cmp}$ represents a complete scenario with $\binom{n}{2}$ edges and we can teach any edge (i,j) in c by at least one instance (i,j,b_k) where $b_k \in \mathcal{B}$ is the singleton relation that holds between i and j. To see why $\mathrm{TD}(\mathcal{C}_{cmp})$ is at least $\binom{n}{2}$, consider the concept c where, for any two edges (i,j) and (j,r) in c, we cannot infer any useful information on the relation between (i,r) from the transitive closure table. Thus, there exists no edges in c where we can infer their relation from the other two edges and one instance per edge is required. ◀

Note that the fact that $\mathcal{C}_{cmp} \subset \mathcal{C}_{inc}$ does not necessarily mean $\mathrm{TD}(\mathcal{C}_{cmp}) \leq \mathrm{TD}(\mathcal{C}_{inc})$ i.e., TD is not monotonic. There could be concepts that are hard to teach on small classes but their teaching becomes easy on large classes. Therefore, the above result gives no clue over the TD of the set of incomplete scenarios.

▶ **Proposition 2.** $\mathrm{TD}(\mathcal{C}_{inc}) = n(n-1) - 1$.

Proof. For the upper bound, at least two examples $x = (i,j,b_k)$ and $x' = (i,j,b_{k'})$ for $b_k \neq b_{k'}$ suffice to teach the relation between any pair (i,j) of entities. In particular, if there exists an edge (i,j) with relation b_k, then $c(x) = 1$ and $c(x') = 0$ otherwise if there exists no edge between i and j then $c(x) = 1$ and $c(x') = 1$ for any concept $c \in \mathcal{C}_{inc}$. As any concept in $\mathcal{C}_{inc}$ has at least one edge in its graph, it follows that we need at least two examples per pair and for at least one pair (i,j) we need exactly one instance (since the QCN where all the edges are universal is not included in this class). Therefore, $\mathrm{TD}(\mathcal{C}_{inc}) \leq n(n-1) - 1$. For the lower bound, consider the concept $c^{\emptyset}$ that represents a QCN with empty edge set in its graph. It is easy to see that we cannot teach such concept with fewer than $n(n-1)$ examples as we need to confirm that every pair (i,j) holds a universal relation. ◀

▶ **Proposition 3.** $\mathrm{TD}(\mathcal{C}_{all}) = |\mathcal{X}|$.

Proof. The instance space size $|\mathcal{X}|$ is a trivial upper bound on the teaching dimension of any class. For the lower bound, consider the concept $c^{\emptyset}$ again. Our argument is that $\mathrm{TD}(c^{\emptyset}, \mathcal{C}_{all}) > |\mathcal{X}| - 1$. To see this, consider any set T of instances where $|T| = |\mathcal{X}| - 1$ and w.l.o.g. assume $\mathcal{X} = T \cup \{(i,j,b_k)\}$ for an arbitrary instance (i,j,b_k). We can always have another concept $c' \neq c^{\emptyset}$ where it represents a QCN with only one edge (i,j) and its relation equals to $\mathcal{B} \backslash \{b_k\}$. Thus, T cannot distinguish $c^{\emptyset}$ from c' and $\mathrm{TD}(c^{\emptyset}, \mathcal{C}_{all}) > |\mathcal{X}| - 1$. ◀

6 Experimentation

We report on the experiments conducted in order to assess the effect of PC on the number of queries needed to learn an IA network. In this regard, we compare our proposed learning algorithm with and without PC or PPC (we call the method without PC, the "Naive" method) for each of the three cases listed in Section 4 as well as the case of incorrect answers. In the case where PC is used, we consider 2 situations: the case where the *QueryGeneration* function generates the queries randomly as well as using an ordering heuristic as described in Section 4. Moreover, for case 2 we consider PPC in addition to PC, given that we have incomplete graphs in this case. All the experiments are conducted on a Dell XPS 8900, i7-6700K, 32 GB RAM, running Linux. All the algorithms are coded in Java. For each of these cases, we first generate a random consistent temporal scenario (that we call G_t). We then add primitives randomly to get an initial graph $G_{problem}$ with the goal of achieving the following, at the end of the learning process: $G_{problem} = G_t$. Each query is generated with a probability P_y of getting a "yes" answer. The general approach is similar for all the cases we consider in the following, and differs in terms of the target G_t).

In case 1, we use the $\mathbf{S}(n,p)$ model [20] that starts by generating n numeric random intervals (pairs of natural values) assigned to each temporal variable in the graph. Allen primitives are then deduced from pairs of these numeric intervals. For example, let us assume the following assignments to two temporal events X and Y: $X = (3, 11)$ and $Y = (7, 18)$. The corresponding Allen primitive is then: (XOY) i.e. X overlaps Y. The model $\mathbf{S}(n,p)$ ensures a consistent solution since all variables have numeric intervals. A random consistent scenario is then build and considered as our target graph G_t. Our algorithm then starts from a complete graph $G_{problem}$ where each edge contains the universal relation I (the disjunction of all the 13 primitives).

Case 2 is similar to case 1 with the following difference when generating the target network. Some edges in G_t will be removed based on a parameter P_u corresponding to the percentage of universal relations in G_t. These relations (edges to be removed) will be in a set E_u that is used in the query generation. $Q_{solution}$ is generated by traversing all the edges in G_t, and by adding all the universal relations in E_u. In case 3, we first generate G_t as in case 1 and then randomly add more primitives to each constraint (X, Y). G_t is therefore a complete graph. $Q_{solution}$ contains queries for the primitives within the constraint (X, Y). The query generation for inconsistent scenarios is produced as follows. Mistakes are picked randomly with a percentage ratio P_m. Query generators are similar to the query generators in case 1, case 2, and case 3. However, queries are labelled with $isMistake$ as a boolean indicator which is checked when answering each query. If $isMistake$ is true, then the answer would be incorrect, otherwise, the answer must be correct. Figures 1, 2, and 3 show the comparative results for the 3 cases when 100 variables are considered. It is clear from the chart corresponding to case 1 that PC has a significant effect on reducing the number of queries especially when there is a small percentage of "yes" answers. For instance, in the extreme case where there is no "yes" answers, the Naive method requires about 60000 queries while the learning method using PC only needs about 2500 queries to reach the target IA scenario. By increasing the percentage of "yes" answers, the number of queries starts to drop down to 4950 queries for 100% of "yes" answers using the Naive method, which is very close to the 3731 queries that are asked using PC. The "cardinality descending" heuristic is the most effective one as we can easily see from the chart, while the other two do not seem to have an effect when compared to PC without heuristics.

The situation in case 2 is similar to the one in case 1. All the methods using path consistency are better than the naive method. These methods have however similar performance.

For case 3, we can easily see that there is a significant difference between the 3 methods considered. PC with "cardinality" ordering heuristic is the winner in this situation and is followed by the PC method without heuristic. These results are justified by the fact that we are dealing with complete graphs with much more relations per constraint than cases 1 and 2.

Tables 2, 3 and 4 show the results for inconsistent scenarios due to mistakes for each of the 3 cases when 100 variables are considered. As we can notice, in all cases the number of queries, mistakes and related backtracks is considerably reduced when path consistency is used.

In Table 2, we can easily see that there is a significant difference between the Naive method and PC in terms of number of queries, number of mistakes detected and running time. There are 14 times more mistakes detected by the Naive method than the PC method. This is explained by the fact that PC removes inconsistent relations at each time leaving less chances for the user to choosing them. Also, the ordering heuristic does not seem to do a good job in this case.

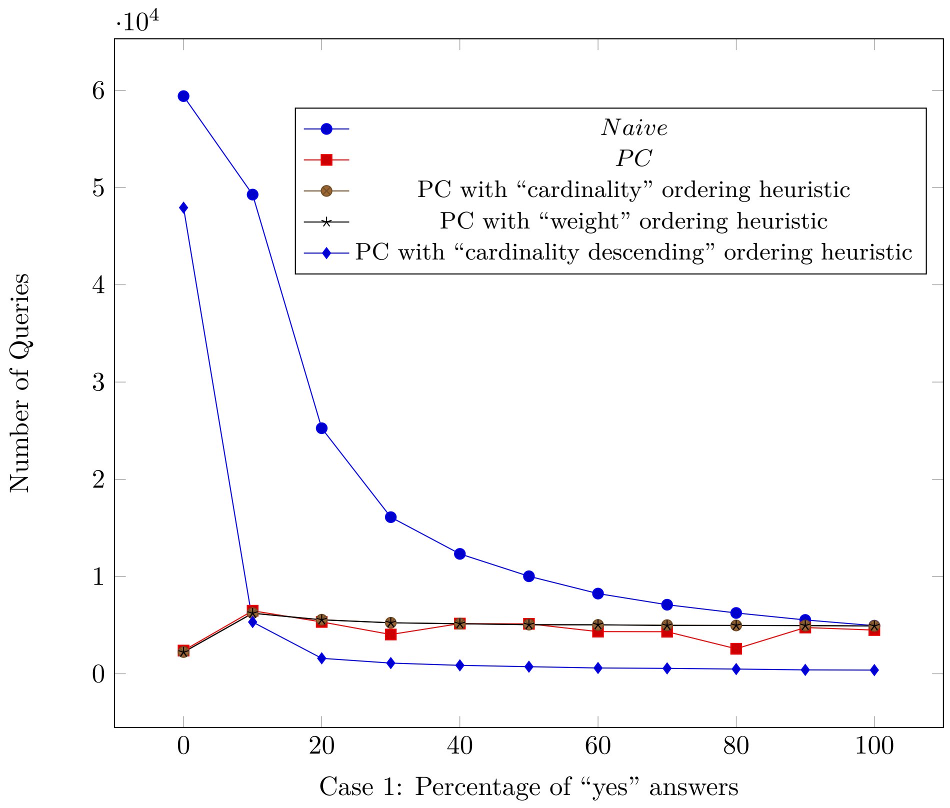

Figure 1 Test results for case 1.

In Table 3, we notice that the performance PPC is better than the one of PC and this is due to the fact that we are dealing with incomplete graphs. Also, the "weight" ordering heuristic is effective in this case for both PC and PPC. This is again explained by the nature of problems we are dealing with in case 2.

Table 4 clearly shows that both the "cardinality" and "weight" ordering heuristics are effective given that the objective is to produce a path consistent QCN.

7 Conclusion

The symbolic representation of time and space is very relevant especially when dealing with incomplete information. We have proposed a new algorithm for learning QCNs by queries. The proposed algorithm is enhanced with path consistency to reduce the number of queries needed to reach a target QCN. In order to assess the effect of PC and ordering heuristics on reducing this number, in practice, we have conducted several experiments on randomly generated IA networks, considering several scenarios. The results of these tests are very promising and encouraging. In this regard, we plan to pursue this work by considering other QCNs such as the Region Connection Calculus (RCC) [13], the rectangle algebra [4] and the n-intersections [10]. We will as well consider temporal constraint networks involving both quantitative and qualitative information [17,18]. Another future direction, we will consider, is to extend our algorithm to learning preferences as these often co-exist with constraints in many real world applications [16,19].

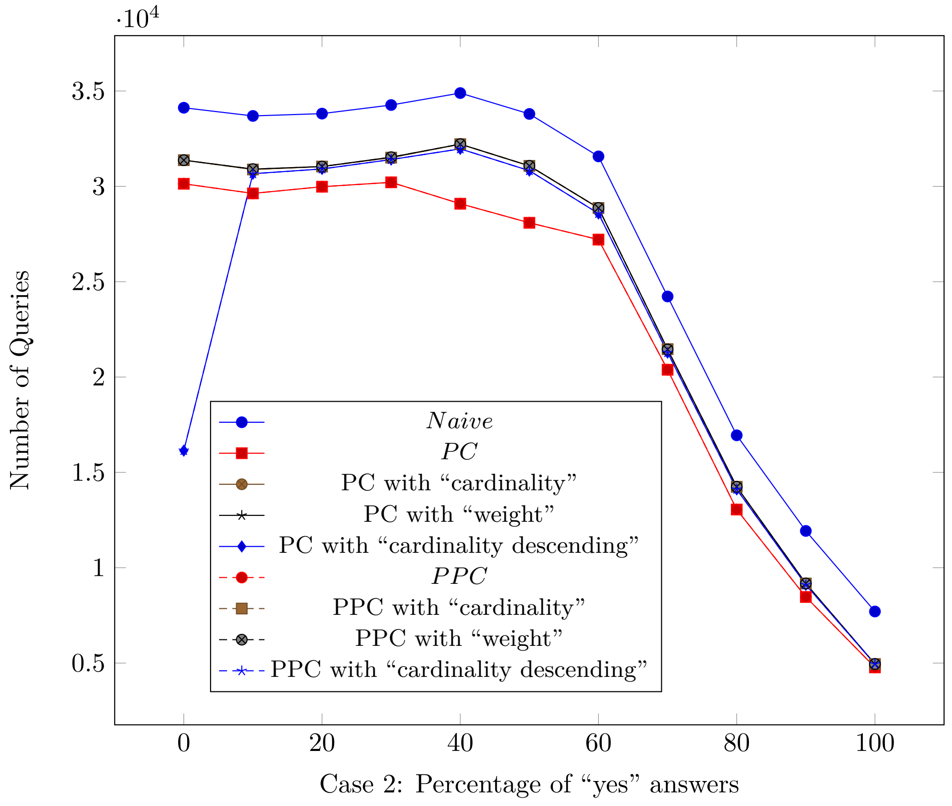

Figure 2 Test results for case 2.

Table 2 Results for inconsistent scenarios for case 1.

Method	# of Queries	Time(s)	Mistakes	Backtracks
Naive	13005708	802.175	399	398
PC	4444	55.061	28	28
PC with "cardinality"	6879	82.741	43	43
PC with "weight"	7819	122.455	36	36
PC with "cardinality descending"	38367	524.589	75	75

Table 3 Results for inconsistent scenarios for case 2.

Method	# of Queries	Time(s)	Mistakes	Backtracks
Naive	63206804	3430.845	6676	6666
PC	36662	458.491	4	5
PPC	10598	114.350	3	2
PC with "cardinality"	28906	327.168	5	0
PPC with "cardinality"	88593	986.297	14	14
PC with "weight"	8786	198.325	1	1
PPC with "weight"	8655	109.046	2	2
PC with "cardinality descending"	132225	2470.913	19	20
PPC with "cardinality descending"	67066	706.155	8	9

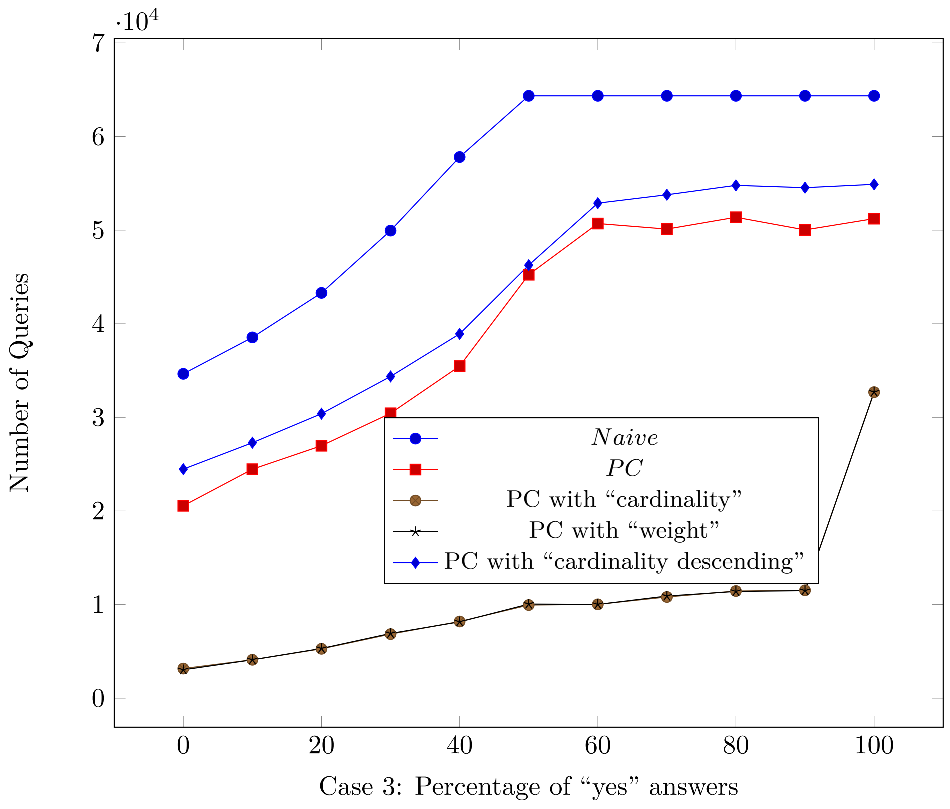

Figure 3 Test results for case 3.

Table 4 Results for inconsistent scenarios for case 3.

Method	# of Queries	Time(s)	Mistakes	Backtracks
Naive	2384321	123.455	66	66
PC	40917	542.344	41	1
PC with "cardinality"	11514	46.772	10	6
PC with "weight"	11280	54.366	11	2
PC with "cardinality descending"	330215	5323.526	69	61

References

1 James F Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

2 Dana Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1988.

3 R. Arcangioli and N. Lazaar. Multiple Constraint Acquisition. In *Workshop on Constraints and Preferences for Configuration and Recommendation, the Twenty-Forth International Joint Conference on Artificial Intelligence*, 2015.

4 Philippe Balbiani, Jean-François Condotta, and Luis Farinas del Cerro. A new tractable subclass of the rectangle algebra. In *Proceedings of the 16th international joint conference on Artifical intelligence - Volume 1*, pages 442–447, 1999.

5 C. Bessiere, R. Coletta, B. O'Sullivan, and M. Paulin. Query-Driven Constraint Acquisition. In *Proceedings of the Twentiest International Joint Conference on Artificial Intelligence*, pages 50–55, 2007.

6 Christian Bessiere, Remi Coletta, Emmanuel Hebrard, George Katsirelos, Nadjib Lazaar, Nina Narodytska, Claude-Guy Quimper, and Toby Walsh. Constraint acquisition via partial queries. In *IJCAI'2013: 23rd International Joint Conference on Artificial Intelligence*, page 7, 2013.

7 Christian Bessiere, Frédéric Koriche, Nadjib Lazaar, and Barry O'Sullivan. Constraint acquisition. *Artificial Intelligence*, 244:315–342, 2017. `doi:10.1016/j.artint.2015.08.001`.

8 Christian Bliek and Djamila Sam-Haroud. Path consistency on triangulated constraint graphs. In *IJCAI*, volume 99, pages 456–461. Citeseer, 1999.

9 Rina Dechter. *Constraint Processing*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

10 Max J. Egenhofer and Robert D. Franzosa. Point-set topological spatial relations. *International Journal of Geographical Information Systems*, 5(2):161–174, 1991.

11 Sally A. Goldman and Michael J. Kearns. On the complexity of teaching. *Journal of Computer and System Sciences*, 50:20–31, 1995.

12 Michael J. Kearns and Umesh V. Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, Cambridge, MA, USA, 1994.

13 Sanjiang Li and Mingsheng Ying. Region connection calculus: Its models and composition table. *Artificial Intelligence*, 145(1):121–146, 2003.

14 Zhiguo Long, Michael Sioutis, and Sanjiang Li. Efficient path consistency algorithm for large qualitative constraint networks. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, pages 1202–1208. AAAI Press, 2016.

15 Roger Mohr and Thomas C Henderson. Arc and path consistency revisited. *Artificial intelligence*, 28(2):225–233, 1986.

16 Malek Mouhoub and Amrudee Sukpan. Managing temporal constraints with preferences. *Spatial Cognition & Computation*, 8(1-2):131–149, 2008.

17 Malek Mouhoub and Amrudee Sukpan. Conditional and composite temporal csps. *Applied Intelligence*, 36(1):90–107, 2012.

18 Peter Revesz. Tightened transitive closure of integer addition constraints. In *Eighth Symposium on Abstraction, Reformulation, and Approximation*, pages 136–142. AAAI, 2009.

19 Samira Sadaoui and Shubhashis Kumar Shil. A multi-attribute auction mechanism based on conditional constraints and conditional qualitative preferences. *JTAER*, 11(1):1–25, 2016.

20 Peter Van Beek and Dennis W Manchak. The design and experimental analysis of algorithms for temporal reasoning. *Journal of Artificial Intelligence Research*, 4(1):1–18, 1996.