Journal of Artificial Intelligence Research 79 (2024) 241-271 Submitted 03/2023; published 01/2024

Query-driven Qualitative Constraint Acquisition

Mohamed-Bachir Belaid BBEL@NILU.NO
NILU, Climate and environmental research institute, Kjeller, Norway.
Department of Computer Science, OsloMet, Oslo, Norway.

Nassim Belmecheri NASSIM@SIMULA.NO
Simula Research Laboratory, Oslo, Norway.

Arnaud Gotlieb ARNAUD@SIMULA.NO
Simula Research Laboratory, Oslo, Norway.

Nadjib Lazaar LAZAAR@LIRMM.FR
LIRMM, University of Montpellier, CNRS, Montpellier, France.

Helge Spieker HELGE@SIMULA.NO
Simula Research Laboratory, Oslo, Norway.

Abstract

Many planning, scheduling or multi-dimensional packing problems involve the design of subtle logical combinations of temporal or spatial constraints. Recently, we introduced GEQCA-I, which stands for *Generic Qualitative Constraint Acquisition*, as a new active constraint acquisition method for learning qualitative constraints using *qualitative queries*. In this paper, we revise and extend GEQCA-I to GEQCA-II with a new type of query, *universal query*, for qualitative constraint acquisition, with a deeper query-driven acquisition algorithm. Our extended experimental evaluation shows the efficiency and usefulness of the concept of universal query in learning randomly-generated qualitative networks, including both temporal networks based on Allen's algebra and spatial networks based on region connection calculus. We also show the effectiveness of GEQCA-II in learning the qualitative part of real scheduling problems.

1. Introduction

Qualitative reasoning about time or space is essential for many practical Artificial Intelligence problems, such as automated planning (Belhadji & Isli, 1998), task scheduling (Barták, Salido, & Rossi, 2008), and multi-dimensional packing problems (Crainic, Perboli, & Tadei, 2012). In this context, qualitative calculus provides an algebraic framework that establishes relations between entities (or pairs of entities) through a language that is jointly exhaustive and pairwise disjoint. Examples of qualitative calculus include Point Algebra (Vilain & Kautz, 1986b) and Allen's Interval Algebra (Allen, 1983) for reasoning about temporal tasks, and Region Connection Calculus (RCC) (Randell, Cui, & Cohn, 1992) for reasoning about topological relationships between spatial regions. Constraint satisfaction techniques and constraint programming (CP) are well-suited frameworks to model and solve qualitative constraint networks.

However, in many practical situations, complex problems are not represented as constraint networks and are solved solely based on historical records and manually crafted

solutions. Furthermore, inter-relationships among entities may only be known locally and between pairs of entities, leaving the implications for other pairs of entities unknown.

To facilitate the modelling of CP problems, the concept of *constraint acquisition* (CA) was introduced for learning CP models. CA can be achieved through *passive learning* from a set of complete labelled example assignments (Bessiere, Coletta, Koriche, & O'Sullivan, 2005), or *active learning* with specific queries that aid in classifying complete assignments (Bessiere, Coletta, O'Sullivan, & Paulin, 2007).

Several state-of-the-art active CA algorithms exist, including: QuAcq (Bessiere, Coletta, Hebrard, Katsirelos, Lazaar, Narodytska, Quimper, & Walsh, 2013), an interactive query-based approach also known as *exact learning* (Bshouty, 2018), which asks complete or partial queries to reduce the set of possible satisfiable constraints from a given constraint language; MultiAcq (Arcangioli, Bessiere, & Lazaar, 2016), an extension of QuAcq that learns the maximum number of constraints violated by a given negative example; T-Quacq (Addi, Bessiere, Ezzahir, & Lazaar, 2018), which places a bound on the query-generation time to speed up CA; MQuAcq-2 (Tsouros, Stergiou, & Bessiere, 2019), which leverages the structure of learned models by focusing queries on quasi-cliques of constraints; and ClassAcq (Prestwich, Freuder, O'Sullivan, & Browne, 2021), which utilizes a Naive Bayes classifier to differentiate solutions from non-solutions and acquire constraint models in a passive manner.

Other notable approaches for learning constraint models include ModelSeeker (Beldiceanu & Simonis, 2012, 2016), which can acquire global constraints, and the general frameworks of *constraint learning* (De Raedt, Passerini, & Reso, 2018) and *constraint synthesis*, which are based on mixed linear integer programming (Pawlak & Krawiec, 2017). The issue of handling incorrect responses to queries is also a significant challenge in interactive CA (Tsouros, Stergiou, & Bessiere, 2020).

However, standard active CA algorithms face difficulties in handling qualitative constraints since managing disjunctions of general relations over variable pairs can result in an exponential increase of the constraint search space.

Moreover, although the number of possible inter-relations is limited, controlling the number of queries asked to the user or the time allocated to generate these queries is a crucial aspect of the adoption of CA techniques in practical applications (Bessiere, Koriche, Lazaar, & O'Sullivan, 2017).

Learning qualitative temporal constraints has been initiated by (Mouhoub, Marri, & Alanazi, 2018) in LQCN. LQCN follows the active learning version of Conacq by considering each qualitative constraint between time intervals as a concept to learn using *membership queries*. Then, the consistency of the network as a whole is maintained using path consistency and by considering the composition table as background knowledge. Further, LQCN was extended to the RCC8 algebra in (Mouhoub, Marri, & Alanazi, 2021) but its generalization to other algebras is not straightforward without further development and correctness proofs. In particular, understanding which fundamental property of an algebra is necessary to generalize CA to other qualitative reasoning is not easy. Also, using background knowledge beyond the only composition table can be beneficial, in cases more information about the network is available (e.g., task durations, resource limits, pre-crafted constraints). These elements are crucial to complete the constraint propagation step and further filter the queries to generate.

Recently, in (Belaid, Belmecheri, Gotlieb, Lazaar, & Spieker, 2022), we have introduced *Generic Qualitative Constraint Acquisition* coined as (GEQCA-I), which is a generic active CA algorithm for learning any kind of constraints between pairs of entities in a qualitative reasoning problem. GEQCA-I is a correct method in CA that acquires any kind of qualitative constraint network. It combines qualitative queries, time-bounded path consistency (PC), Path-Lex, a novel search heuristic and extends background knowledge propagation to learn any qualitative constraint network. Our approach differs from classical active CA algorithms in three ways. First, it handles the disjunction of qualitative constraints effectively through the usage of queries and path consistency. Second, it generalizes the concept of CA for qualitative constraint networks by building its theoretical framework on the JEPD (Joint Exhaustive and Pairwise Disjoint) property. Third, it solves CP models to reduce the number of queries using extended background knowledge.

In this paper, our revision and extension of GEQCA-I to GEQCA-II are threefold:

1. We introduce a new type of query, *universal query*, motivated by real-world scenarios and revise GEQCA-I accordingly. We provide a sound and complete query-driven CA algorithm to propagate any information through path consistency. This paper contains a full theoretical analysis of GEQCA-II;

2. We propose another constraint selection heuristics, named Path-Weighted. We experimentally evaluate which constraint selection heuristics perform better for learning qualitative networks;

3. We provide an extended experimental evaluation of GEQCA-II on temporal and spatial qualitative networks, which answers four research questions related to the comparison between GEQCA-I and II. We also address qualitative networks extracted from real-world scheduling problems.

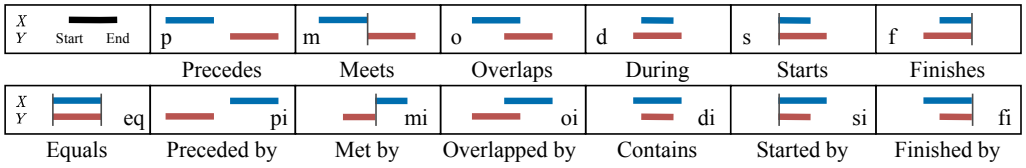
The paper is organized as follows: In the next section, we introduce the necessary background material required to understand the paper; In Section 3, we introduce the concept of *universal query* and present GEQCA-II, our generic approach to learning qualitative constraints in CA. In Section 4, we conduct experiments with our implementation of GEQCA-II to evaluate its effectiveness and performance. Finally, we conclude the paper in Section 5.

2. Background

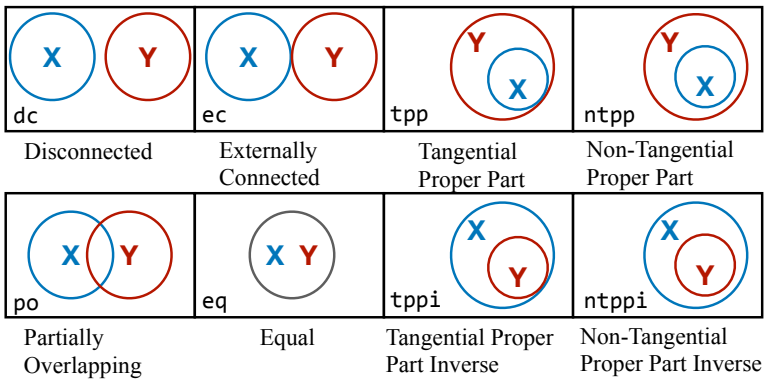
2.1 Qualitative Calculus

A Qualitative Constraint Network (QCN) is a finite set of (binary) constraints $\mathcal{C}$ expressed on X, where X is a finite set of variables representing temporal or spatial entities (points, intervals, regions, etc.). For instance, an *interval* variable X_i is a pair of endpoints (X_i^-, X_i^+) on the real line where $X_i^- < X_i^+$ holds.

A QCN is built on a language Γ, which stands for a finite set of jointly exhaustive and pairwise disjoint (JEPD) binary relations (e.g., before, after, contains, part-of, etc.) defined over an infinite domain D (e.g., a topological space for Region Connection Calculus or a real line for Allen's Interval Algebra). The 13 basic relations (resp., the 8 basic relations) in



(a)



(b)

Figure 1: (a) Overview of the 13 basic relations in Allen's Interval Algebra (Allen, 1983) as named in Krokhin et al. (2003) and (b) the 8 basic relations in Region connection calculus. These relations resemble the possible temporal/spatial qualitative queries, the oracle is asked.

Allen's Interval Algebra (resp., in RCC8) are illustrated in Figure 1.(a) (resp., Figure 1.(b)).

Formally speaking, $\Gamma = \{r_1, \ldots, r_m\}$ and $|\Gamma| = m$. Bear in mind that the relational algebra generated by Γ is finite, and it is closed under (weak) composition, converse and contains the identity. A binary constraint $C_{ij} = \{r_{k_1}, \ldots, r_{k_d}\}$, is a disjunction of d basic relations in Γ between X_i and X_j. Note that C_{ji} represents the inverse constraint (with inverse relations) of C_{ij} (i.e., $C_{ji} = C_{ij}^{-1}$). Here, C_{ij} is interpreted as $(X_i\ r_{k_1}\ X_j) \vee \ldots \vee (X_i\ r_{k_d}\ X_j)$ and $|C_{ij}| = d$. We denote by $\bot$ the empty constraint (i.e., the constraint defined by the empty set) and by $\top$ the universal constraint (i.e., the constraint defined by a set containing all basic relations of the calculus). $Size(C) = \sum_{(C_{ij} \in C | i<j)} |C_{ij}|$ denotes the total number of basic relations in C.

The QCN satisfiability problem is the problem of deciding if there exists a temporal/spatial solution (i.e., an interpretation of the variable entities satisfying all constraints) of a given QCN.

Example 1. *Figure 2 illustrates a temporal QCN with C_1 using Allen's Interval Algebra, a spatial QCN with C_2 using RCC8 and for each of the two QCNs, a possible solution. In C_1, the constraint between the two intervals X_1 and X_2 is $C_{12} = \{\mathtt{p}, \mathtt{m}, \mathtt{o}\}$. The inverse of C_{12}*

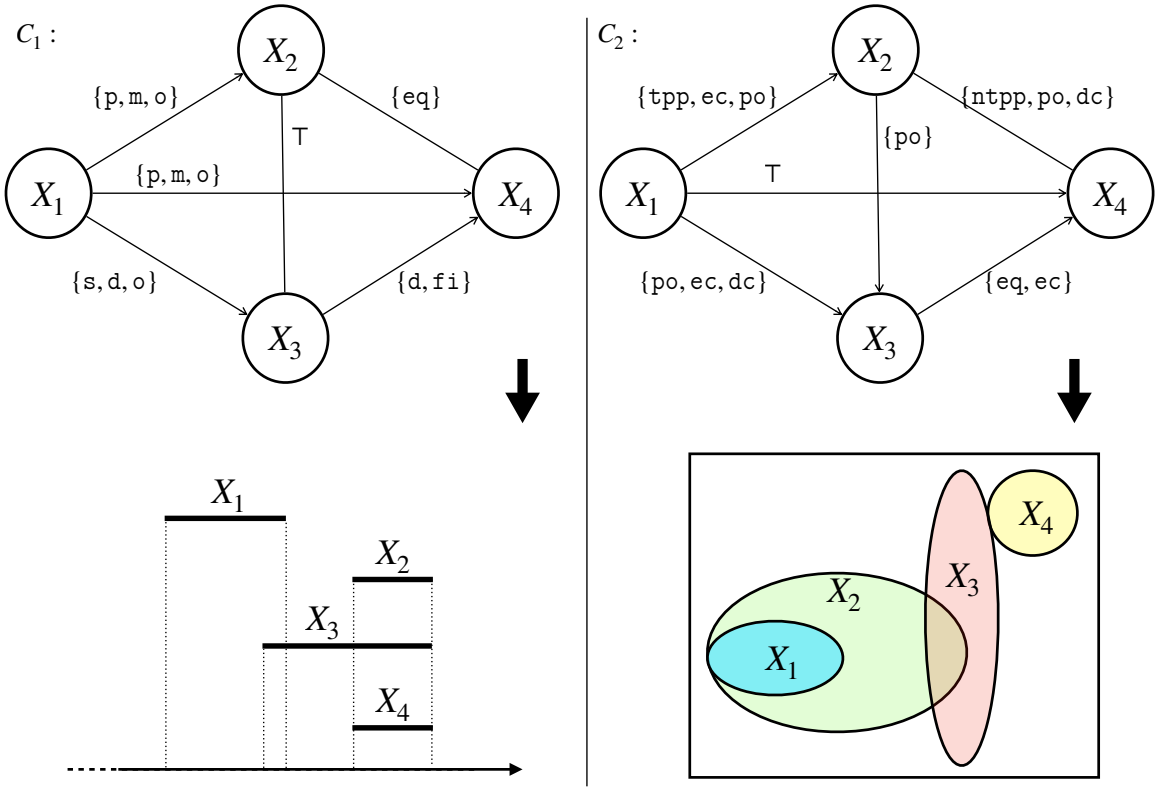


Figure 2: Illustrative examples of temporal QCN with C_1 and spatial QCN with C_2 and the corresponding possible solutions.

is $C_{21} = \{\texttt{pi}, \texttt{mi}, \texttt{oi}\}$. *The size of* C_1 *and* C_2 *is* $size(C_1) = 25$ *and* $size(C_2) = 19$, *where the universal constraint* $\top$ *is of size 13 (resp., 8) in Allen's algebra (resp., in RCC8).*

2.2 Qualitative Constraint Acquisition

A CA-Agent is a constraint acquisition algorithm (a.k.a., learner or inducer in Machine Learning) defined to acquire constraints from data. In active learning, the CA-Agent needs to share some common knowledge, materialized by the vocabulary (X, D) in our context, with an oracle to communicate through queries. An assignment e on X is rejected by a constraint network C if and only if there is at least one constraint C_{ij} which rejects $e_{ij} = (v_i, v_j)$, the projection of e on (X_i, X_j). e_{ij} violates C_{ij} iff for each $r_k \in C_{ij}$, $(v_i, v_j) \notin r_k$. If (v_i, v_j) does not violate C_{ij}, then $(v_i, v_j) \models C_{ij}$. An assignment e on X, which is accepted by C, is a solution of C. $Sol(C)$ denotes the set of solutions of C. In addition to the vocabulary (X, D), the CA-Agent owns the language Γ from which it can build constraints on specified sets of entity variables.

Given a vocabulary (X, D), a qualitative concept is a Boolean function f over $D^{|X|}$, i.e., a mapping of each complete assignment e to a value in $\{true, false\}$. A representation of a

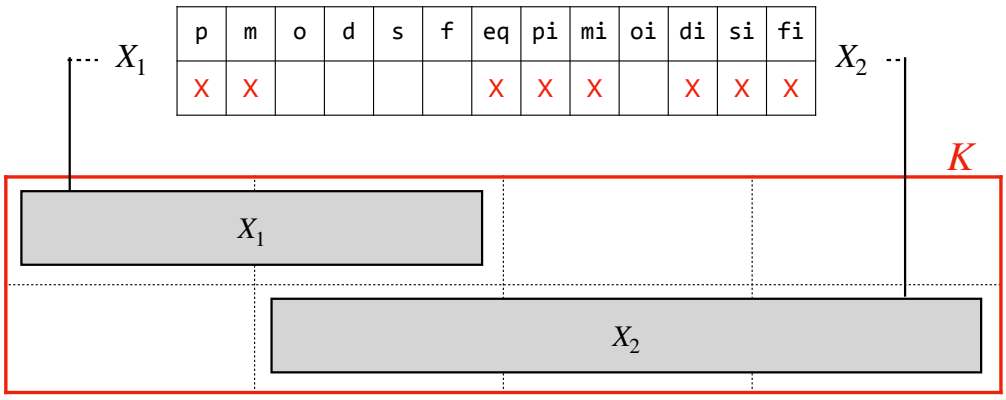


Figure 3: An example of two time intervals including background knowledge.

qualitative concept f is a QCN, C for which $f^{-1}(true) = sol(C)$, denoted by $f = sol(C)$. The oracle qualitative concept is a concept f_Q that returns $true$ for e if and only if e is a solution to the problem that the oracle has in mind. The target network, denoted as T, is a QCN representing the concept f_Q. It is worth noting that f_Q may correspond to an empty concept $f_Q = \varnothing$. In this case, any unsatisfiable QCN can serve as a candidate for representing the empty concept.

Oracles, teachers, or domain experts can serve as sources of information for learning, but they are not necessarily human-based oracles (Angluin, 1988; Valiant, 1984). The information that needs to be learned is assumed to be known by an oracle, which can be a program in some cases (Menguy, Bardin, Lazaar, & Gotlieb, 2022) or a crowd collectively answering queries (Lazaar, 2021). In many cases, this source of information is composed of multiple locally consistent pieces of knowledge that collectively form the concept to be learned, denoted as f_Q. However, this composed concept may not be entirely coherent and may not have a clear solution, resulting in an empty concept ($f_Q = \varnothing$). To represent the locally consistent part of the concept f_Q for the pair (X_i, X_j), we use the notation $f_Q[i, j]$.

We now define convergence, which is a crucial property of CA. Given a set E of examples labelled by the oracle as $true$ or $false$, we say that a network C agrees to E if C accepts all examples labelled $true$ and rejects all examples labelled $false$. The learning process has converged on the network L if **(i)** L agrees to E and **(ii)** for every other network L' agreeing to E, we have $sol(L') = sol(L)$. Hence, if the learning process has converged, we guarantee that $sol(L) = f_Q$. If there is no such L found, then we say that the learning process has collapsed. For instance, it happens when f_Q is an empty concept (i.e., $f_Q = \varnothing$).

In practical applications, it happens that the structure of the problem is known, as well as some quantitative constraints, or some models of durative tasks/actions and their preconditions/effects. In CA, that information type can be presented as background knowledge, noted $\mathcal{K}$. From $\mathcal{K}$, we can deduce an extended qualitative knowledge $\mathcal{K}_Q$ by using constraint propagation or the resolution process. Hence, the concept to learn f_Q is subsumed by $\mathcal{K}_Q$ (i.e., $f_Q \subseteq \mathcal{K}_Q$).

Example 2. *Let us consider the example presented in Figure 3 under Allen's Interval Algebra. We have two time intervals X_1 and X_2, and background knowledge $\mathcal{K}$ which states that (i) X_1 and X_2 have a duration of two and three time units each (ii) the schedule can take a maximum of four time units. From (i), we can deduce that X_1 cannot contain, be equal to, be started by, or be finished by X_2. From (ii), we can deduce that X_1 cannot precede, meet, be preceded by, or be met by X_2. What remains as basic relations between X_1 and X_2 represents the deduced qualitative knowledge:*

$$\mathcal{K}_Q : (C_{12} = \{\mathtt{o}, \mathtt{d}, \mathtt{s}, \mathtt{f}, \mathtt{oi}\})$$

Note that standard version space-based approaches such as Conacq (Bessiere et al., 2017) and QuAcq (Bessiere et al., 2013) are not suitable for qualitative reasoning, as (i) the acquisition is based on a finite domain and (ii) only languages closed under conjunction can be considered.

3. GEQCA-II: Constraint Acquisition via Qualitative Queries

We first recall the concept of the qualitative query introduced in (Belaid et al., 2022) and then, we introduce a new type of query, namely universal query. In the following, we make the hypothesis that GEQCA-II has access to a perfect oracle (human or program) that always answers queries correctly. We further discuss this hypothesis in 4.5.

Definition 1 (Qualitative Query)**.** *Given a qualitative concept f_Q on (X, D) under Γ, a qualitative query* Q-Ask$(X_i\{r\}X_j)$ *takes as input a constraint with a unique basic relation $r \in \Gamma$ on a pair of variables (X_i, X_j), and outputs "yes" if $\exists e_{ij} \in r$ that satisfies $f_Q[i, j]$ and "no" otherwise.*

Please note that the JEPD property guarantees that for any given pair of variables (X_i, X_j), each possible value pair (v_i, v_j) is contained in exactly one relation $r \in \Gamma$. As a result, we can infer that, when considering a pair of entity variables (X_i, X_j) and a basic relation $r \in \Gamma$, a qualitative query answers the question of whether $X_i \ \{r\} \ X_j$ holds or not. It is worth noting that a positive response to the qualitative query Q-Ask(X_i, r, X_j) does not necessarily imply that e_{ij} can be extended to form a solution for the qualitative concept to be learned. The existence of such a solution cannot be guaranteed. In fact, determining whether e_{ij} can be extended to a solution is an NP-complete problem (Vilain & Kautz, 1986a; Maddux, 1994).

In some practical scenarios, the oracle may provide information showing that a universal constraint exists between two entities, X_i and X_j, something that allows any basic relation to hold between them. This gives rise to a new type of query called the universal query.

Definition 2 (Universal Query)**.** *Given a qualitative concept f_Q on (X, D) under Γ, a universal query* U-Ask(X_i, X_j) *takes as input a pair of variables (X_i, X_j), and outputs "yes" if $\forall r \in \Gamma : \exists e_{ij} \in r$ that satisfies $f_Q[i, j]$ and "no" otherwise.*

Once again, by utilizing the JEPD property of Γ, a universal query can answer the question of whether all basic relations in Γ hold between a pair of entity variables (X_i, X_j), sequentially or otherwise. In other words, the universal query can determine whether all

possible basic relations between X_i and X_j are allowed. The concept of qualitative and universal queries in GEQCA-II can be explained and motivated by considering the following simple usage scenario.

Example 3. *Consider a professional architect with experience in building modern-style houses. This architect is responsible for scheduling between 50 to 100 distinct construction tasks and wishes to acquire a general constraint model that can be used for any construction project. At the beginning of the process, the architect can answer "universal queries" to determine which tasks are independent*[1] *from each other. For example, a positive response to the query* U-Ask*("carpentry", "build the walls") would validate all relations between "carpentry" and "build the walls", and there would be no need to ask qualitative queries between these two tasks. After that, "qualitative queries" can be used to detect the exact allowed relations between dependent tasks. For instance, if the architect positively answers the query* Q-Ask*("build the walls" {precedes} "put the roof"), this would validate the "precedes" relation between the task of "build the walls" and the task of "put the roof".*

3.1 Description of GEQCA-II

GEQCA-II is presented in Algorithm 1. GEQCA-II takes as input a vocabulary (X, D) of n entity variables, the Γ language of binary basic relations, a background knowledge $\mathcal{K}$ and a timed boundary parameter `cutoff`.

Initially, GEQCA-II sets the possible constraints between entities of the network L to the universal constraint $\top$ (line 4). The value of τ is then assigned to `cutoff` in order to ensure that the waiting time between two queries does not exceed the time limit of `cutoff`. Subsequently, GEQCA-II iterates over L to reduce the constraints to sets of basic relations that are equivalent to the concept being learned f_Q (lines 6-18). To accomplish this, GEQCA-II first selects a C_{ij} and, in the case of $C_{ij} = \top$, presents a universal query on (X_i, X_j) to the oracle (line 7). If the universal query is answered positively, the universal constraint between X_i and X_j is retained.

If there is no universal constraint between X_i and X_j, then at least one basic relation must be removed. To accomplish this, GEQCA-II performs two steps. First, the propagate procedure is applied to reduce C_{ij} by propagating $\mathcal{K}$ on it in τ (line 8). For example, in a temporal context, if "X_i and X_j tasks have respective duration of 1 and 2 hours", which is deduced from $\mathcal{K}$, then the propagate procedure removes *Equals*, *Contains*, *Started-by*, and *Finished-by* from C_{ij}. The value of τ is then updated by deducting the time used in $\mathcal{K}$ propagation. Second, GEQCA-II iterates over the remaining basic relations of C_{ij} (lines 10-16). Each basic relation r is checked in τ to ensure its consistency with the already learned network L and $\mathcal{K}$ (line 11). If the relation is not consistent, meaning that τ is sufficient to prove that there is no solution ($s = \varnothing$), it can be removed (line 14). Otherwise, the resolution finds a solution or τ is insufficient, and in both cases, the basic relation is presented to the oracle under a qualitative query Q-Ask(X_i, r, X_j) (line 13). Finally, the value of τ is updated by deducting the time consumed by the solve process (line 12).

If the oracle answers *no*, then GEQCA-II removes r from C_{ij} (line 14). If C_{ij} is reduced at line 8 or line 14, then GEQCA-II eliminates non-feasible relations from L by maintaining path consistency using the `PC` function, with a time limit of τ. If C_{ij} is reduced to the empty

1. Independent from user perspective, but they can be indirectly dependent.

Algorithm 1: GEQCA-II: Constraint Acquisition via Qualitative Queries.

```
1  In: vocabulary (X, D); Γ language; background knowledge K; parameter cutoff;
2  Out: a learned network L;
3  begin
4      L ← {C_ij = ⊤ : i < j};
5      τ ← cutoff;
6      foreach C_ij ∈ L do
7          if (C_ij = ⊤) ∧ (U-Ask(X_i, X_j) = yes) then continue;
8          CHANGE← propagate(K, C_ij, τ)
9          update(τ);
10         foreach r ∈ C_ij do
11             s ← solve(L ∪ K ∪ X_i{r}X_j, τ)
12             update(τ);
13             if (s = ∅) ∨ (Q-Ask(X_i{r}X_j) = no) then
14                 C_ij ← C_ij \ {r}
15                 CHANGE← true;
16                 if s ≠ ∅ then τ ← cutoff;
17         if (C_ij = ⊥) ∨ (CHANGE ∧¬PC({C_ij}, τ)) then
18             return "collapse"
19 Function PC(Q, τ):
20     while (Q ≠ ∅) ∧ (τ has not yet elapsed) do
21         pick C_ij in Q
22         foreach X_k ∈ X \ {X_i, X_j} do
23             Δ_1 ← C_ik ∩ composition(C_ij, C_jk); if Δ_1 = ⊥ then return false
24             Δ_2 ← C_ki ∩ composition(C_ki, C_ij); if Δ_2 = ⊥ then return false
25             if C_ik ≠ Δ_1 then C_ik ← Δ_1; Q ← Q ∪ {C_ik}
26             if C_kj ≠ Δ_2 then
27                 C_kj ← Δ_2; Q ← Q ∪ {C_kj}
28     if Q = ∅ then update(τ);
29     return true
```

constraint $\perp$ in line 14, or if an inconsistency is detected by PC (lines 23 and 24), it shows that the space of possible networks collapses because of an empty concept ($f_Q = \varnothing$) (line 18). In lines 19-29, we present the PC function (Mackworth, 1977), which is a time-bounded path consistency incremental propagator with a cubic complexity in the number of entity variables. PC returns *true* when a fixpoint (transitive closure) is reached by propagating qualitative knowledge, or when τ has elapsed, and $false$ when any inconsistency is detected.

3.2 Theoretical Analysis

In this section, we establish the correctness of the GEQCA-II algorithm (Algorithm 1) in learning any constraint network that represents a qualitative concept over the Γ language. We prove that GEQCA-II is sound, complete, and terminates.

Let (X, D) be a vocabulary of n entity variables, Γ be the set of basic relations, f_Q be a qualitative concept, and let $\mathcal{K}_Q \supseteq f_Q$ be knowledge deduced by a given background knowledge $\mathcal{K}$, then the following propositions and Theorem 1 hold.

Proposition 1 (Soundness)**.** *The network L returned by* GEQCA-II *is such that $f_Q \subseteq sol(L)$.*

Proof. Suppose there exists an example $e \in f_Q$ that is not a solution of L, i.e., $e \not\models L$, then there must be at least one constraint $C_{ij} \in L$ for which the projection of the solution e_{ij} does not satisfy C_{ij}, i.e., $e_{ij} \not\models C_{ij}$. This implies that there is a missing relation r^* in C_{ij} such that $e_{ij} \in r^*$. The missing relation r^* is unique due to JEPD property of the Γ language. Since the *propagate* procedure (line 8) and the consistency check (line 11) are based on a background knowledge that subsumes f_Q ($\mathcal{K}_Q \supseteq f_Q$) and the `PC` function is sound (Allen, 1983; Vilain & Kautz, 1986a), the only place where r^* can be removed from C_{ij} is at line 14 due to a negative answer on a qualitative query. Furthermore, a qualitative query on a consistent relation between a given pair of entities cannot be answered with *no*. Therefore, removing a basic relation from L cannot cause L to reject an example accepted by f_Q. □

The soundness property of GEQCA-II ensures that it generates a constraint network L that preserves the solution of the target concept f_Q ($sol(L) \subseteq f_Q$). Consequently, it avoids the removal of any basic relation that is part of a solution or could be extended to form one. Consider, for example, the set of basic relations $\mathcal{S} = \{p, o, m\}$ between X_i and X_j, representing relations participating in solutions of f_Q. The soundness property ensures the acquisition of a constraint C_{ij} that wholly encompasses the set $\mathcal{S}$ ($\mathcal{S} \subseteq C_{ij}$, as illustrated by $C_{ij} = \{p, o, m, f, s\}$). Importantly, it guarantees that no constraint C'_{ij} will be generated in a manner that $C'_{ij} \cap \mathcal{S} \neq \mathcal{S}$ (for instance, $C'_{ij} = \{o, m, s\}$).

Proposition 2 (Completeness)**.** *The network L returned by* GEQCA-II *is such that $sol(L) \subseteq f_Q$.*

Proof. Suppose there exists $e \in sol(L) \setminus f_Q$. Here, L accepts solutions rejected by the concept f_Q, which means that at least one constraint $C_{ij} \in L$ needs to be restricted further to make $e \not\models L$. To restrict C_{ij} further, we need to remove at least one relation r^* from C_{ij}. If $C_{ij} = \top$, GEQCA-II submits a universal query to the oracle at line 7. The query is negatively answered as $r^* \in C_{ij}$. If r^* is not removed using $\mathcal{K}$ propagation (lines 8 and 11) or using `PC`, it is presented to the oracle as a qualitative query. Since r^* is inconsistent on (X_i, X_j), the qualitative query on r^* can only be negatively answered. Thus, we conclude that keeping a basic relation in L cannot cause L to accept an example rejected by f_Q. □

The completeness property of GEQCA-II ensures that the resultant constraint network L does not introduce new solutions concerning the target concept f_Q (i.e., $sol(L) \subseteq f_Q$). For

example, if the basic relations between X_i and X_j, forming part of the solutions of the target concept, are denoted as $\mathcal{S} = \{p, o, m\}$, the completeness property ensures the acquisition of a constraint C_{ij} such that the basic relations in $C_{ij} \setminus \mathcal{S}$ are not part of the solutions of the target concept (e.g., $C_{ij} = \{p, o, s\}$ with s representing a globally inconsistent relation between X_i and X_j).

Proposition 3 (Termination)**.** GEQCA-II *terminates.*

Proof. The GEQCA-II algorithm iterates over all pairs of entities, which results in $\frac{n(n-1)}{2}$ iterations (line 6). For each pair of entities, if a universal query is negatively answered in line 7, GEQCA-II iterates over the language Γ (line 10). Since the size of the pair of entities and Γ is finite, and the *propagate* and *solve* procedures as well as the `PC` function are time-bounded, the algorithm is guaranteed to terminate. □

Theorem 1 (Correctness)**.** *The network L returned by* GEQCA-II *is such that $sol(L) = f_Q$.*

Proof. Correctness immediately follows from Propositions 1, 2, and 3. □

Proposition 4 (Waiting time)**.** GEQCA-II *learns a network L, or collapses, with a waiting time not exceeding* `cutoff` *time bound between two queries.*

Proof. If `cutoff` $= \infty$, the proposition holds trivially. However, if `cutoff` $< \infty$, GEQCA-II asks a universal query at each iteration of its main loop (line 6). If the query is positively answered, GEQCA-II asks a new universal query with negligible time between the queries. If a universal query is negatively answered, GEQCA-II calls the *propagate* function (line 8) and enters the inner loop in line 10 by asking a series of qualitative queries. Between two queries, *propagate*, *solve*, and `PC` are executed in a time $t < \tau$, where τ is initialized to `cutoff` (line 5) and only decreased by the *update* function after each call to the three procedures/functions. If a qualitative query is asked, τ is reset to `cutoff` at line 16. If τ is reduced to 0, GEQCA-II will ask a query at the next iteration where *propagate*, *solve*, and `PC` cannot be called, and then τ is reset to `cutoff`. Thus, the waiting time between two queries will never exceed the `cutoff` time-bound. □

Complexity of GEQCA-II in terms of queries.

We analyze the complexity of GEQCA-II in terms of number of queries. It is worth noticing that there are two types of queries involved. The first type is universal queries, which are submitted to the oracle in line 7 of Algorithm 1. The second type is qualitative queries, which are submitted to the oracle in line 13 of Algorithm 1.

Given a vocabulary (X, D) of n entity variables, a language Γ of m basic relations, a qualitative concept f_Q, qualitative knowledge $\mathcal{K}_Q \supseteq f_Q$ deduced from a given background knowledge $\mathcal{K}$, a number k of basic relations pruned using $\mathcal{K}$ and `PC` propagation, and the number p of universal queries positively answered, GEQCA-II asks $O(\frac{n(n-1)}{2})$ universal queries and exactly $\eta = (\alpha - \beta - \theta)$ qualitative queries, where:

- $\alpha = \frac{mn(n-1)}{2}$: m relations per pair of entities;

- $\beta = k$: pruned relation using $\mathcal{K}$ and PC;
- $\theta = mp$: number of relations between independent entities.

3.3 Strategies

To optimize GEQCA-II further, we can improve the constraint selection process at line 6 by using more sophisticated heuristics. By selecting constraints in different ways, we can significantly impact both $\mathcal{K}$ and PC propagation, leading to fewer queries needed to reach convergence. While several heuristics have been proposed for solving qualitative networks, such as the weighted heuristic (van Beek & Manchak, 1996), our goal is to learn these networks. To that end, we introduce two dedicated constraint selection heuristics, the Path-Weighted and Path-Lex heuristics, which are based on traversing the line graph of the QCN. The line graph is a graph derived from the QCN, where each edge of the original graph is represented as a vertex in the line graph. Two vertices in the line graph are adjacent if and only if their corresponding edges in the original graph share a common endpoint (Ore, 1987). In our context, the QCN is constructed on n entity variables X_i as vertices and $n(n-1)/2$ constraints C_{ij} as edges. In the corresponding line graph $\mathcal{L}(C)$, C_{ij} and C_{jk} are vertices that share the edge X_j. The Path-Weighted heuristic traverses the line graph in a way that minimizes the weight of the selected constraints, while the Path-Lex heuristic builds a complete path to maximize the impact of PC and the transitivity between constraints.

3.3.1 Path-Weighted Heuristic

Algorithm 2 outlines our heuristic for selecting constraints to form a path traversing the line graph of a QCN C, with the aim of maximizing the path length. Given the last selected constraint C_{ij} and the QCN C, Path-Weighted starts by updating the set of visited nodes (line 2). Next, it computes the set of entity variables X_k that have at least one non-visited constraint with either X_i or X_j (line 3). If this set is empty, it means that all constraints involving X_i or X_j have already been visited. In such a case, Path-Weighted randomly selects a new starting point for a path from the non-visited constraints (line 4). Otherwise, Path-Weighted selects the constraint linked to X_i or X_j with the minimum weight as the next constraint in the path (line 5, with MinWeight function detailed in lines 7-12). The weight of each constraint is an estimate of how much the basic relations will restrict other constraints, and it is computed as the sum of the weights of involved basic relations (van Beek & Manchak, 1996). Finally, Path-Weighted returns the selected constraint to be used in the next iteration (line 6).

3.3.2 Path-Lex Heuristic

Algorithm 3 presents our second heuristic for selecting constraints forming a path traversing the line graph of the corresponding QCN, C, while maximizing its length. Similar to Path-Weighted, Path-Lex begins by updating the set of visited nodes (line 2), and then computes the set of entity variables X_k that have a non-visited constraint with X_i or X_j (line 3). However, unlike Path-Weighted, Path-Lex selects the next constraint to visit in a lexicographic manner (line 4). Specifically, given the last visited constraint C_{ij} and

Algorithm 2: Path-Weighted constraint selection heuristic

```
1  Function Path-Weighted(C, Visited, C_ij):
2      Visited ← Visited ∪ {C_ij}
3      Y ← {X_k ∈ X | C_ik ∉ Visited ∨ C_kj ∉ Visited}
4      if Y = ∅ then Δ ← pick C_kl ∈ C \ Visited
5      else Δ ← MinWeight(Y, Visited, i, j)
6      return Δ
7  Function MinWeight(Y, Visited, i, j):
8      min ← ∞
9      foreach X_k ∈ Y do
10         if (C_ik ∉ Visited) ∧ (Weight(C_ik) < min) then Δ ← C_ik;
             min ← Weight(C_ik)
11         if (C_kj ∉ Visited) ∧ (Weight(C_kj) < min) then Δ ← C_kj;
             min ← Weight(C_kj)
12     return Δ
```

Algorithm 3: Path-Lex constraint selection heuristic

```
1  Function Path-Lex(C, Visited, C_ij):
2      Visited ← Visited ∪ {C_ij}
3      Y ← {X_k ∈ X | C_ik ∉ Visited ∨ C_kj ∉ Visited}
4      k ← lex(Y)
5      if C_ik ∉ Visited then Δ ← C_ik
6      else if C_kj ∉ Visited then Δ ← C_kj
7      else Δ ← tossUp(C_ik, C_kj)
8
9      return Δ
```

the X_k lexicographically selected at line 4, if both C_{ik} and C_{kj} are non-visited constraints, the path is extended with one of them in a random manner (line 7). The idea behind this approach is to ensure that the traversal of the line graph covers the QCN efficiently, while prioritizing constraints that are likely to have a significant impact on $\mathcal{K}$ and PC propagation.

In summary, Path-Weighted and Path-Lex are two dedicated constraint selection heuristics that aim to maximize the length of a path traversing the line graph of a QCN C while ensuring that constraints with a high impact on $\mathcal{K}$ and PC propagation are visited early on. These heuristics have been shown to significantly improve the number of queries needed to reach convergence as compared to more brute-force approaches.

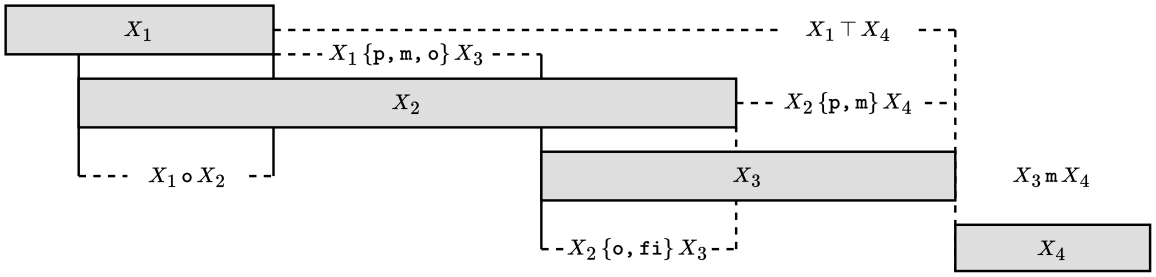


Figure 4: Visualization of the illustrative example and the relations intended by the oracle. Given duration of the time intervals are chosen arbitrarily but obey the imposed relations on the ordering.

3.4 Illustrative Example

To illustrate our approach, let an Allen's interval-based network with four time intervals $X = \{X_1, X_2, X_3, X_4\}$ and a time concept f_Q, which captures the oracle's intention. The time concept can be expressed as follows: "X_1 overlaps X_2, X_1 precedes, meets, or overlaps X_3, X_2 overlaps or finished by X_3, X_2 precedes or meets X_4, and X_3 meets X_4. For X_1 and X_4, there is no specific intention."

The oracle's intention defines an ordering of the time intervals but allows one for some flexibility in the exact placement and duration of the intervals. To illustrate the problem, Figure 4 depicts the intervals, the user's intention, and arbitrary lengths for the time intervals.

Figure 5 depicts the QCN of our example and its corresponding line graph in parts (a) and (b), respectively. Part (c) of the figure illustrates the use of Path-Weighted as a constraint selector, where w_{ij}^k represents the weight of constraint C_{ij} at iteration k of GEQCA-II. Initially, C_{24} is randomly selected, since all constraints have the same weight as the universal constraint (i.e., $\forall C_{ij} \in C : w_{ij}^1 = 34$). For the second iteration, Path-Weighted randomly selects C_{34}. Afterwards, the possible relations for C_{23} are reduced by `PC` and it gets selected in the third iteration. In the fourth iteration, constraints C_{12} and C_{13} are the remaining non-visited constraints that are connected to C_{23} (i.e., constraint at the current iteration). Constraint C_{12} is randomly selected, since it has the same weight as C_{13} (`PC` did not cause any change in their weights).

Finally, C_{14} is selected as its weight $w_{14}^5 = 3$ is the minimum among the non-visited, connected constraints, before C_{13} is selected in the sixth and last iteration. It results in the following path traversing the line graph:

$$\text{Path-Weighted}: \; C_{24} \rightarrow C_{34} \rightarrow C_{23} \rightarrow C_{12} \rightarrow C_{14} \rightarrow C_{13}$$

Figure 5.(d) illustrates the use of Path-Lex. In this heuristic, the line graph is traversed in a lexicographic order, where constraints are visited following the lexicographic order of their variables. The resulting path is as follows:

$$\text{Path-Lex}: C_{12} \rightarrow C_{23} \rightarrow C_{13} \rightarrow C_{14} \rightarrow C_{24} \rightarrow C_{34}$$

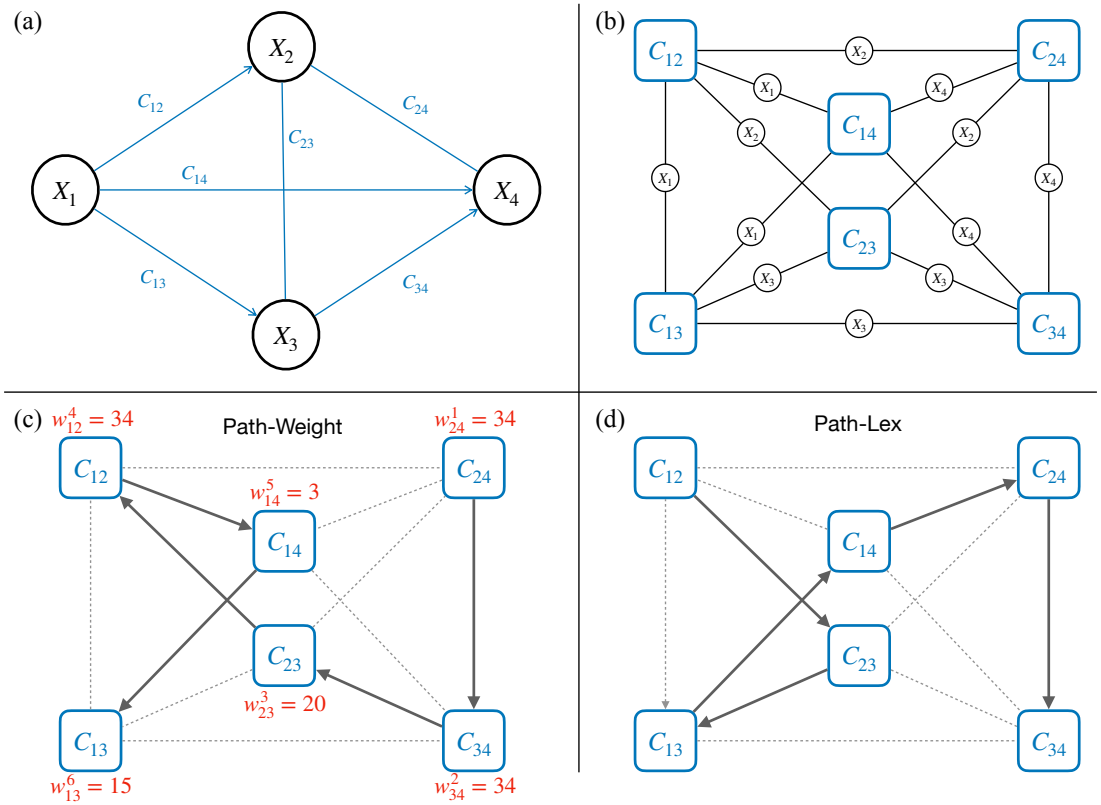


Figure 5: QCN example with four entity variables (a), its corresponding line graph (b) and illustrative examples of the use of Path-Weighted (c) and Path-Lex (d).

To illustrate the use of GEQCA-II on the example, we apply the Path-Lex heuristic. We first demonstrate the behavior of GEQCA-I (without universal queries) and then present GEQCA-II. Table 1 displays the learned relations after each iteration.

Iteration 1: The oracle is queried 13 times for all possible relations of C_{12} and restricts it to requiring that interval 1 overlaps interval 2. Since all other pairs still allow all 13 basic relations, no further reduction can be calculated at this point.

Iteration 2: The oracle is queried 13 times for the possible relations of C_{23} and restricts it to requiring that interval 2 overlaps or is finished by interval 3. Given the information already collected, the path consistency algorithm infers that for the pair of $(1, 3)$, only the relations precedes, meets, and overlaps are possible.

Iteration 3: This pair is then queried next on the remaining three relations, which are all confirmed by the oracle.

Iteration 4: The oracle is queried 13 times until it is discovered that no specific relations are intended for C_{14}.

Iteration 5: After querying on C_{24} in and confirming that interval 2 should precede or meet interval 4, the path-consistency method infers a first set of relations on pair $(3, 4)$, on which the oracle has not yet been queried. Additionally, even though the oracle has

no intention for C_{14}, we already identify through PC that, given the other answers, only a precedes relation is possible for this pair.

Final iteration 6: The oracle is queried five times to further reduce the relations to the intent that interval 3 should meet interval 4, and GEQCA-II finishes after 60 queries.

If we consider the same example, but under the availability of universal queries, the total number of queries reduces to 51. Universal queries are asked at iteration 1, 2, 4, and 5. Three of these pairs are dependent, but C_{24} is independent and the single universal query is therefore sufficient at iteration 4. At iteration 3 and 6, no universal query is asked because it is already known that some relations do exist.

Iteration	C_{12}	C_{13}	C_{14}	C_{23}	C_{24}	C_{34}
1	o	$\top$	$\top$	$\top$	$\top$	$\top$
2	o	p, m, o	$\top$	o, fi	$\top$	$\top$
3	o	p, m, o	$\top$	o, fi	$\top$	$\top$
4	o	p, m, o	$\top$	o, fi	$\top$	$\top$
5	o	p, m, o	p	o, fi	p, m	p, m, o, fi, di
6	o	p, m, o	p	o, fi	p, m	m

Table 1: Illustrative example of GEQCA with the Path-Lex heuristic. A gray background highlights the pair i, j on which the oracle is queried. Bold values show a change in the relations, either due to the queries or the following path consistency algorithm. $\top$ marks pairs with the full $|\Gamma| = 13$ basic possible relations in Allen's calculus (see Figure 1(a)).

4. Experiments

This section presents an experimental evaluation of the revised GEQCA-I, which we refer to as GEQCA-II. We compare GEQCA-II with classical GEQCA-I (Belaid et al., 2022) when learning temporal constraints based on Allen's Interval Algebra with $|\Gamma| = 13$ (see Figure 1(a)), and learning spatial networks based on RCC8 with $|\Gamma| = 8$ (see Figure 1(b)). Before reporting on the experiments, we compare different constraint selection heuristics to determine the most promising one. The evaluation aims to answer the following research questions:

- **RQ1**: Which constraint selection heuristic performs better for learning qualitative networks using GEQCA-I?
- **RQ2**: Is GEQCA-II useful for learning qualitative networks, and how does it compare with GEQCA-I and LQCN?
- **RQ3**: Is GEQCA-II capable of learning the qualitative part of real-world scheduling problems, and how does it compare with GEQCA-I?
- **RQ4**: What is the impact of a bounded time limit between two queries on the oracle effort?

Experimental Evaluation Protocol. In our experiments, we follow a protocol adapted from (Nebel, 1997; Renz & Nebel, 2001) to generate random instances for both the Allen and RCC8 calculi. We start by instantiating each pair of nodes with an initial consistent relation. To create inconsistent instances, we randomly select three nodes and replace the relation on one pair with one not in the composition of the other two pairs, as derived from the composition table. We then follow the A(n,d,s) procedure (Nebel, 1997; Renz & Nebel, 2001) and verify the consistency or inconsistency of the generated instances using the GQR reasoner (Gantner, Westphal, & Wölfl, 2008). If the instance is not consistent, we restart the process.

We vary the number of nodes `n`, the average degree `d`, which denotes how many other nodes have specified relations, the average label size `l`, which denotes the number of relations specified between nodes, and whether the instance is satisfiable or not (`SAT/UNSAT`). The maximum degree is limited by the number of nodes, and the maximum average label size is limited by the size of the calculus. The average label size of the instances describes their *density*. When there are many pairs of nodes with the universal relation, i.e., a high average label size, we refer to these instances as having a higher density because there are more alternatives in the oracle's intention. This is in comparison to instances where only a few relations are possible between pairs of nodes, which we refer to as *sparse*.

We refer to Allen instances as `Allen_n_d_l_{SAT,UNSAT}_i`, and RCC8 instances as `RCC8_n_d_l_{SAT,UNSAT}_i`, where i is the instance number. We provide the exact parameters for all experiments when they are discussed. We generated 40 instances per algebra and triplet (n, d, l), with 20 instances being SAT and 20 being UNSAT. The values of n, d, and l were chosen from the sets $\{10, 25, 50, 100\}$, $\{(n-1), (n/2)\}$, and $\{1, (|\Gamma|/2)\}$, respectively, except for $n = 100$, where only 4 instances were generated per algebra and triplet. In total, we generated 968 instances.

The algorithms GEQCA-I and GEQCA-II were implemented in Java, and the Choco solver[2] was used for the solve procedure at line 11 in Algorithm 1. The code and complete description of each instance can be found at the following public repository: `https://github.com/lirmm/ConstraintAcquisition/tree/GEQCA`.

All tests are run on an Intel core $i7$, $2.8GHz$ with RAM of $16GB$.

4.1 [RQ1]: Constraint Selection Heuristics

For our first experiment, we evaluate the effectiveness of different constraint selection heuristics. We compare four heuristics: (i) Lex - the lexicographical heuristic, (ii) Weighted - the weight-based heuristic, and our (iii) Path-Weighted and (iv) Path-Lex heuristic presented in Section 3.3. We use each heuristic with three basic settings: (1) GEQCA-I is used as described in (Belaid et al., 2022) (2) $\mathcal{K}$ is empty, and (3) the `PC` function runs until a fixpoint is reached (`cutoff` $= \infty$). As mentioned in (Belaid et al., 2022), employing the Weighted heuristic in GEQCA-I is equivalent to using LQCN (LearningQCN) as proposed in (Mouhoub et al., 2018, 2021). This equivalence enables us to compare our work with LQCN in the experiments.

Figure 6 shows the results of GEQCA-I using the different heuristics on 240 SAT and 240 UNSAT instances for Allen's Algebra and RCC8. The instances are generated with

2. https://choco-solver.org/

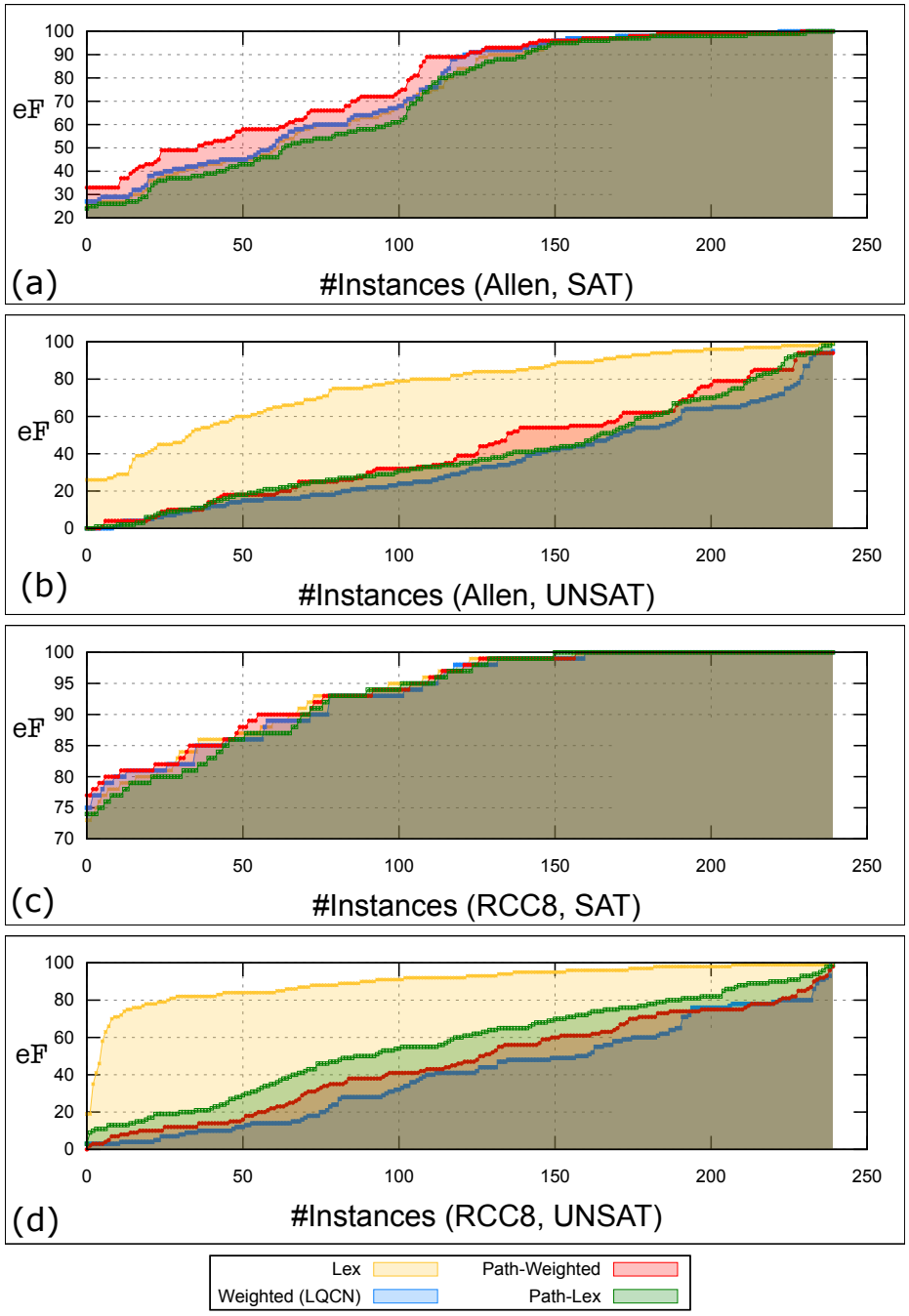


Figure 6: Comparison of GEQCA-I performance (`eF`: oracle effort) on QCN instances using different heuristics.

$n \in \{10, 25, 50\}$. For each learned instance, we report the oracle effort eF which represents the ratio of queries that need to be classified by the oracle according to $\mathtt{Q_{max}}$, the maximum number of queries (i.e., $\mathtt{Q_{max}} = \frac{mn(n-1)}{2}$, where n is the number of entities and m is the number of basic relations). The oracle effort is calculated as $\mathtt{eF}=(\mathtt{Q}^+ + \mathtt{Q}^-)/\mathtt{Q_{max}}$.

The results show the effectiveness of using the Path-Lex and Weighted heuristics for learning SAT and UNSAT instances, respectively. The Path-Lex heuristic shows better performance in learning SAT instances by enhancing the impact of PC. In contrast, the Weighted heuristic, being the most efficient in solving QCN, detects inconsistencies early in the learning phase, resulting in less effort. However, our path-based heuristics remain competitive for UNSAT instances.

On the densest instances where the PC function is not significantly involved during the learning process, we observed that Path-Lex outperforms the other heuristics on Allen's instances, with 99 instances showing more than 90% effort, compared to 111 for Lex, and 118 for both Path-Weighted and Weighted (LQCN). On RCC8 densest instances, the Weighted heuristic performed the best (LQCN), with 168 instances showing more than 90% effort, compared to 171 for Path-Lex, 175 for Lex and 185 for Path-Weighted.

Let us now take a closer look at the experiments. The first parts of Table 2 and Table 3 present the results for selected instances per configuration. For each instance and each heuristic, we report the total number of positive qualitative queries ($\mathtt{Q}^+$), the total number of negative qualitative queries ($\mathtt{Q}^-$), and the required oracle effort (eF).

The number of positive qualitative queries ($\mathtt{Q}^+$) represents the basic relations validated by the oracle in L. As PC is called at each iteration and can remove previously validated basic relations, the size of the returned L can be smaller than $\mathtt{Q}^+$ (i.e., $size(L) \leq \mathtt{Q}^+$). Our observations on SAT instances reveal that different heuristics lead the oracle to validate different sets of basic relations. For instance, the size of the learned network L for Allen_25_12_6_SAT_20 is bounded above by a range of $1,022$ to $1,502$, depending on the heuristic used. However, on UNSAT instances, using the Weighted (LQCN) heuristic leads to a lower number of validated basic relations before reaching the collapse state. For example, Weighted (LQCN) collapses on Allen_25_12_1_UNSAT_19 after validating 42 basic relations, while the other heuristics require validating between 138 and 677 basic relations.

The number of negative qualitative queries ($\mathtt{Q}^-$) gives us an indication of the impact of PC. A higher number of removed basic relations using PC results in fewer negative queries. We can observe that Path-Lex enhances the performance of PC by rapidly forming triangles in the line graph. For instance, in Allen_100_99_1_SAT_1, Path-Lex requires only $7,740$ negative queries while the other heuristics require between $8,316$ and $17,168$ negative queries.

When considering the set of 484 UNSAT instances, the average effort required by Path-Lex, Path-Weighted, Weighted (LQCN), and Lex is 48.60%, 45.09%, 38.09%, and 83.11%, respectively. In this case, Weighted (LQCN) clearly outperforms the other heuristics. However, for the set of 484 SAT instances, the average effort required by Path-Lex, Path-Weighted, Weighted (LQCN), and Lex is 82.78%, 86.43%, 84.32%, and 83.94%, respectively. Given that the main focus is on learning SAT instances, Path-Lex is the most efficient heuristic for learning QCNs. Therefore, we will use Path-Lex in the rest of the experiments.

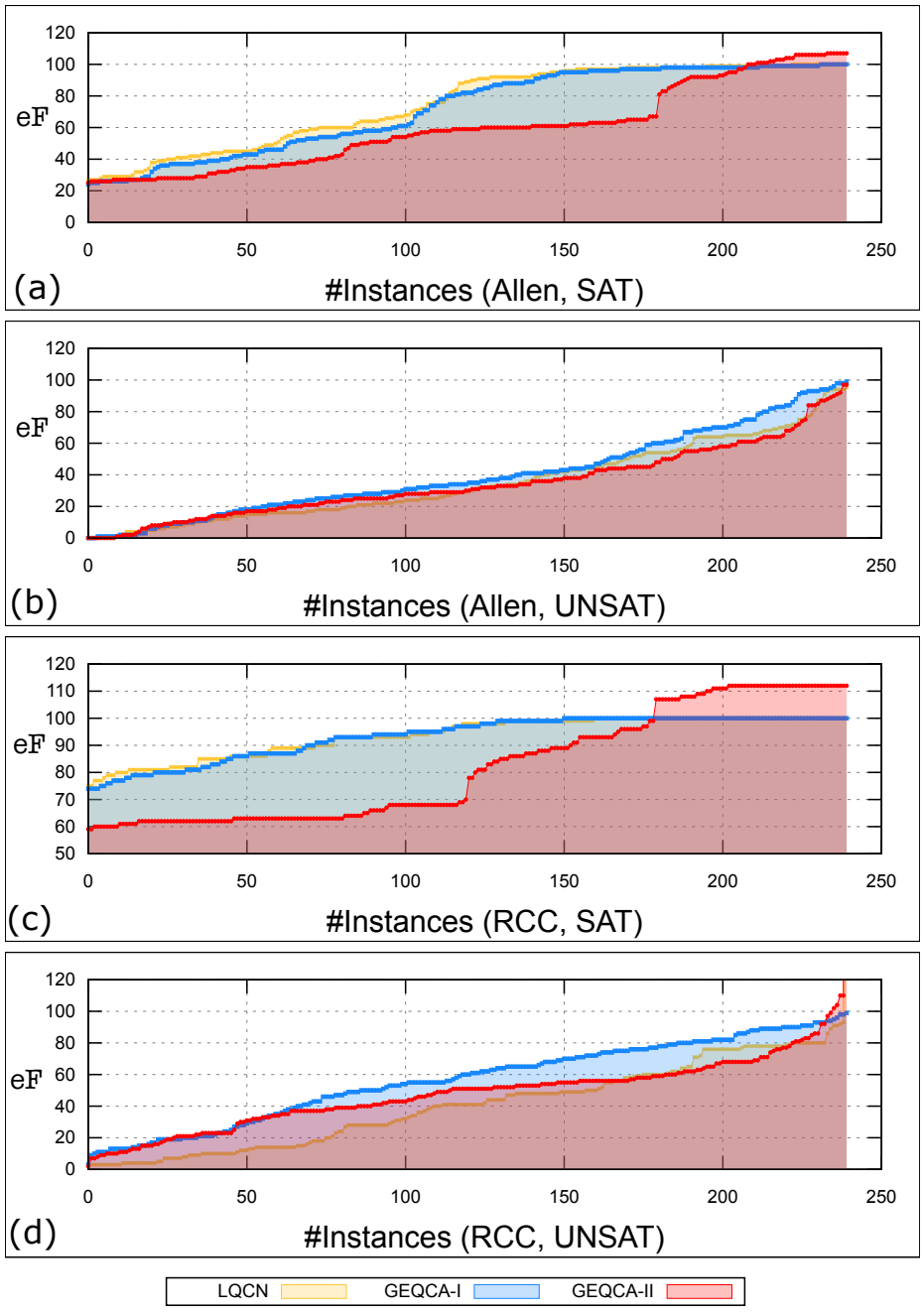


Figure 7: Comparison of Effort Required by LQCN, GEQCA-I, and GEQCA-II on QCN instances.

4.2 [RQ2]: Effectiveness of GEQCA-II Compared to LQCN and GEQCA-I

For our second experiment, we evaluate the effectiveness of GEQCA-II in comparison to GEQCA-I and LQCN. We employ GEQCA-II and GEQCA-I with three basic features: (i) selection of pairs in line 6 of Algorithm 1 is carried out using the Path-Lex heuristic; (ii) $\mathcal{K}$ is empty; and (iii) the PC function is executed until a fixpoint is reached ($\texttt{cutoff} = \infty$). The sole distinction is the utilization of universal queries (line 7 of Algorithm 1) in GEQCA-II.

Figure 7 illustrates the performance of LQCN, GEQCA-I, and GEQCA-II on 240 SAT and 240 UNSAT instances for Allen's Algebra and RCC8, with instance sizes $n \in \{10, 25, 50\}$. The results are presented in terms of the oracle effort eF, which represents the ratio of asked queries to the maximum number of possible queries. It is worth noting that the use of universal queries in GEQCA-II can lead to an effort greater than 100% for some instances. Specifically, we calculate $\texttt{eF}=(\texttt{Q}^+ + \texttt{Q}^- + \texttt{UQ}^+ + \texttt{UQ}^-)/\texttt{Q}_{\texttt{max}}$, where $\texttt{UQ}^+$ is the total number of positive universal queries and $\texttt{UQ}^-$ is the total number of negative universal queries.
Tables 2 and 3 show the performance of GEQCA-II on a selected set of instances for each configuration. Along with $\texttt{Q}^+$, $\texttt{Q}^-$, and eF, we include the total number of positive universal queries ($\texttt{UQ}^+$) and negative universal queries ($\texttt{UQ}^-$) for GEQCA-II.

On SAT instances, GEQCA-II outperforms GEQCA-I and LQCN, achieving an average effort of 71.25% compared to 83% and 84.54%, respectively. When additional universal queries are used, GEQCA-II reduces the effort needed by LQCN and GEQCA-I by approximately 5% on 75 instances, 10% on 25 instances, and more than 15% on 50 instances for Allen's Algebra (Figure 6.(a)). For instance, it reduces the effort by 40% and 38% on the instance Allen_25_24_6_SAT_20 (Table 2) compared to LQCN and GEQCA-I, respectively. Similarly, on RCC8 SAT instances (Figure 6.(c)), GEQCA-II reduces the required effort by approximately 20% on half of the instances. For example, it reduces the effort by 80% on the instance RCC8_100_50_6_SAT_3 (Table 3).

In cases where there are many universal constraints in the problem instances (i.e., dense instances), GEQCA-II outperforms both GEQCA-I and LQCN, achieving an effort greater than 90% on only 52 Allen's Algebra instances (or 88 RCC8 instances). In contrast, GEQCA-I and LQCN require an effort greater than 90% on 99 and 119 Allen's Algebra instances (or 171 and 168 RCC8 instances), respectively. However, on instances with few or no universal constraints (i.e., sparse instances), GEQCA-II is essentially equivalent to GEQCA-I, except for the additional universal queries it asks.

On UNSAT instances (Figure 7.(b) and (d)), GEQCA-II is more or less equivalent to LQCN.

In summary, the results confirm that using universal queries can decrease the effort required for instances with universal constraints. This effect is most pronounced in dense instances where there are more universal constraints between entities. However, on sparse instances where there is a few or no universal constraints, GEQCA-II performs similarly to GEQCA-I, except for the additional universal queries it asks.

Figure 8 presents a comparison of the oracle effort, `eF`, of GEQCA-II with LQCN and GEQCA-I on Allen SAT instances of size 25, for two scenarios: (a) variable label size of 1 and variable degree from 1 to 24, and (b) degree of 24 and variable label size from 1 to 13. This comparison strengthens our observations.

In Figure 8.(a), we observe that GEQCA-II significantly reduces the oracle effort compared to LQCN and GEQCA-I, especially for smaller degree values, where there are more pairs with the universal constraint. For example, for degree $\texttt{d} = 1$, GEQCA-II requires only 12% effort, while LQCN and GEQCA-I require more than 99%. This improvement is due to the use of universal queries, which prevent GEQCA-II from asking $|\Gamma|$ additional qualitative queries if a given pair of entities are independent. However, for higher degree values, GEQCA-II requires more effort. For degree $\texttt{d} = 21$, GEQCA-II requires less than 1% additional oracle effort compared to GEQCA-I (41.3% vs 40.8%) and is still better than LQCN (41.3% vs 43.7%).

In Figure 8.(b), the average degree $\texttt{d} = 24$, which means that there are no universal constraints (in average) between entities. This scenario puts GEQCA-II in the most extreme conditions where universal queries are useless. For smaller average label sizes (from 1 to 5), the effort of GEQCA-II is close to GEQCA-I (requires only 2% additional effort on average) and even outperforms LQCN (requires 10% less effort on average). For average label sizes in the middle range (from 6 to 11), GEQCA-II requires slightly more effort compared to LQCN and GEQCA-I (only 4% additional effort on average). However, for larger label size where universal constraints start to appear, GEQCA-II significantly reduces the effort, especially when label size $= |\Gamma|$ (13 in Allen's case), where GEQCA-II reduces the effort to only 7% compared to 100% using LQCN and GEQCA-I. This case highlights the usefulness of universal queries since more pairs of entities are related by the universal relation.

4.3 [RQ3]: Effectiveness of GEQCA-II vs GEQCA-I for Learning the Qualitative Part of Real-world Scheduling Problems

For our third experiment, we aim to evaluate the effectiveness of GEQCA-II and GEQCA-I in a real-world context by learning temporal constraints of the Resource Constrained Project Scheduling Problem (RCPSP) (Hartmann & Briskorn, 2010). In this experiment, we do not consider LQCN as it does not utilize the background knowledge. We use publicly available RCPSP instances[3] and consider the problem structure with task durations, resource requirements, and resource capacities as background knowledge $\mathcal{K}$, denoted by K_1. It is worth noting that K_1 can be propagated at line 8 of GEQCA-II. Additionally, some constraints, such as the cumulative global constraint and deadline constraints, may already be known by the oracle. We refer to the background knowledge that includes the cumulative and deadline constraints as K_2. The information in $K_1 \wedge K_2$ can be propagated at line 11 of GEQCA-II.

In Table 4, we present the oracle effort for 5 scheduling instances using GEQCA-II and GEQCA-I with the Path-Lex heuristic, a cutoff time of $3,600s$, and different background knowledge ($\mathcal{K}$) settings: $\varnothing$, K_1, and $K_1 \wedge K_2$. Additionally, we report the maximum waiting

3. https://github.com/MiniZinc/minizinc-benchmarks/tree/master/rcpsp

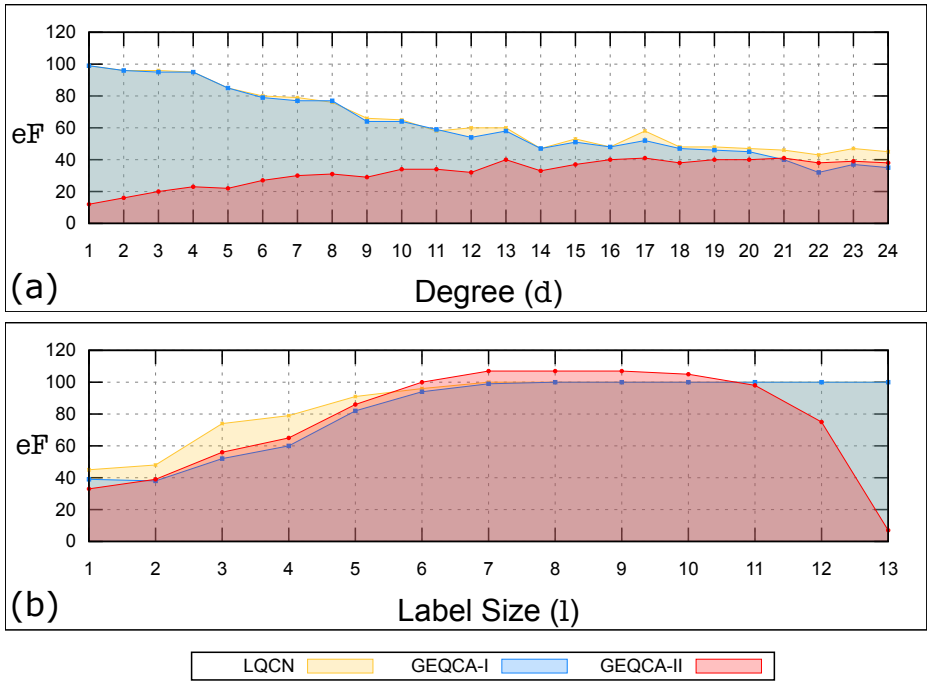


Figure 8: Comparison of Effort Required by LQCN, GEQCA-I, and GEQCA-II on Allen SAT instances of size 25 with: (i) fixed average label size of 1 and variable degree (from 1 to 24) and (ii) fixed degree of 24 and variable average label size (from 1 to 13).

Instance	GEQCA-I												GEQCA-II				
	Lex			Weighted (LQCN)			Path-Weighted			Path-Lex			Path-Lex				
	$\mathtt{Q}^+$	$\mathtt{Q}^-$	eF	$\mathtt{Q}^+$	$\mathtt{Q}^-$	eF	$\mathtt{Q}^+$	$\mathtt{Q}^-$	eF	$\mathtt{Q}^+$	$\mathtt{Q}^-$	eF	$\mathtt{Q}^+$	$\mathtt{Q}^-$	$\mathtt{UQ}^+$	$\mathtt{UQ}^-$	eF
Allen_10_9_1_SAT_19	45	308	60%	45	320	62%	45	380	72%	45	226	**46%**	45	268	0	20	57%
Allen_10_9_1_UNSAT_7	29	193	37%	11	133	24%	21	145	28%	5	49	**9%**	5	49	0	4	10%
Allen_10_9_6_SAT_14	276	304	99%	276	304	99%	276	304	99%	266	290	**95%**	276	302	0	43	106%
Allen_10_9_6_UNSAT_13	208	289	84%	15	33	**8%**	213	287	85%	15	36	9%	26	32	0	3	10%
Allen_10_5_1_SAT_17	157	232	66%	157	232	66%	273	252	89%	157	192	**59%**	32	232	13	19	**50%**
Allen_10_5_1_UNSAT_12	101	115	36%	75	91	28%	47	61	**18%**	29	93	21%	3	29	2	3	**6%**
Allen_10_5_6_SAT_18	401	172	98%	401	172	98%	422	162	99%	403	172	98%	143	168	20	21	**60%**
Allen_10_5_6_UNSAT_20	303	152	77%	102	46	25%	225	91	54%	45	25	**13%**	18	32	0	3	**9%**
Allen_25_24_1_SAT_9	300	1,120	36%	300	1,204	38%	300	1,428	44%	300	948	**32%**	300	964	0	70	34%
Allen_25_24_1_UNSAT_7	36	281	8%	17	50	1%	2	29	**0.8%**	2	29	**0.8%**	2	29	0	2	0.8%
Allen_25_24_6_SAT_7	1,711	1,877	92%	1,757	1,965	95%	1,752	1,953	95%	1,676	1,799	**89%**	1,686	1,801	0	216	95%
Allen_25_24_6_UNSAT_19	789	924	43%	792	918	43%	1,424	1,670	79%	413	479	**22%**	425	495	0	63	25%
Allen_25_12_1_SAT_16	1,072	912	50%	1,116	970	53%	1,502	1,096	66%	1,022	928	**49%**	307	930	57	74	**35%**
Allen_25_12_1_UNSAT_19	644	566	31%	42	37	**2%**	677	596	32%	138	41	4%	12	109	5	9	**3%**
Allen_25_12_6_SAT_20	2,827	1,030	98%	2,832	1,030	99%	2,833	1,028	99%	2,809	1,011	**97%**	977	1,066	138	133	**59%**
Allen_25_12_6_UNSAT_20	1,353	558	49%	729	282	26%	742	282	26%	492	276	**19%**	232	277	20	37	**14%**
Allen_50_49_1_SAT_17	1,225	3,050	27%	1,225	3,512	30%	1,225	4,934	38%	1,225	2,756	**25%**	1,225	2,966	0	232	27%
Allen_50_49_1_UNSAT_1	108	1,150	7%	157	986	7%	138	1,549	10%	16	34	**0.3%**	16	34	0	3	0.3%
Allen_50_49_6_SAT_3	6,203	6,527	80%	6,812	7,453	89%	6,829	7,420	89%	6,035	6,177	**77%**	6,112	6,243	0	584	81%
Allen_50_49_6_UNSAT_4	642	757	8%	43	56	**0.6%**	82	101	1%	92	110	1%	92	110	0	14	1%
Allen_50_25_1_SAT_15	3,353	2,708	38%	3,429	2,790	39%	4,729	3,216	50%	3,365	2,670	**37%**	1,225	2,966	0	232	27%
Allen_50_25_1_UNSAT_17	1,075	897	50%	21	61	**0.5%**	312	421	4%	88	85	1%	13	121	7	10	**0.9%**
Allen_50_25_6_SAT_3	10,999	4,203	95%	11,007	4,207	95%	11,009	4,200	95%	10,951	4,168	**95%**	4,713	4,155	477	485	**61%**
Allen_50_25_6_UNSAT_18	9,396	3,450	80%	659	240	**5%**	3,780	2,529	62%	1,452	598	13%	566	596	69	77	**8%**
Allen_100_99_1_SAT_1	4,950	8,316	20%	4,950	9,064	21%	4,950	17,168	34%	4,950	7,740	**19%**	4,950	7,992	0	636	21%
Allen_100_99_1_UNSAT_1	808	5,039	9%	110	746	**1%**	112	666	1%	813	2,453	5%	813	2,425	0	189	5%
Allen_100_50_1_SAT_1	11,946	7,428	30%	12,098	7,584	30%	15,086	10,352	39%	11,382	6,994	28%	4,639	7,196	559	582	**20%**
Allen_100_50_1_UNSAT_1	9,187	6,609	24%	6,062	3,533	15%	5,459	4,045	14%	2,850	1,973	7%	495	1,901	175	157	**4%**

Table 2: Comparison GEQCA-I using different heuristics vs GEQCA-II using the best heuristic on Allen instances. eF: Oracle effort, $\mathtt{Q}^+$: positive qualitative queries, $\mathtt{Q}^-$: negative qualitative queries, $\mathtt{UQ}^+$: positive universal queries, $\mathtt{UQ}^-$: negative universal queries.

Instance	GEQCA-I												GEQCA-II				
	Lex			Weighted (LQCN)			Path-Weighted			Path-Lex			Path-Lex				
	$\mathtt{Q}^+$	$\mathtt{Q}^-$	eF	$\mathtt{Q}^+$	$\mathtt{Q}^-$	eF	$\mathtt{Q}^+$	$\mathtt{Q}^-$	eF	$\mathtt{Q}^+$	$\mathtt{Q}^-$	eF	$\mathtt{Q}^+$	$\mathtt{Q}^-$	$\mathtt{UQ}^+$	$\mathtt{UQ}^-$	eF
RCC8_10_9_1_SAT_10	45	245	80%	45	249	81%	45	249	81%	45	223	**74%**	45	218	0	18	78%
RCC8_10_9_1_UNSAT_4	11	74	23%	7	38	**12%**	22	122	40%	8	49	16%	7	48	0	5	16%
RCC8_10_9_4_SAT_2	173	178	97%	173	178	97%	173	178	97%	173	181	98%	173	181	0	41	109%
RCC8_10_9_4_UNSAT_14	36	43	21%	9	16	**6%**	58	55	31%	13	20	9%	17	22	0	4	12%
RCC8_10_5_1_SAT_14	154	162	87%	160	163	89%	160	166	90%	154	146	**83%**	47	142	14	18	**61%**
RCC8_10_5_1_UNSAT_1	95	54	41%	76	88	45%	17	20	**10%**	35	22	16%	9	20	5	3	**10%**
RCC8_10_5_4_SAT_3	254	105	98%	254	105	99%	254	105	99%	248	97	**95%**	95	105	20	25	**68%**
RCC8_10_5_4_UNSAT_17	235	90	90%	195	82	76%	190	79	74%	73	41	**31%**	77	88	15	20	55%
RCC8_25_24_1_SAT_5	300	1,493	74%	300	1,611	79%	300	1,605	79%	300	1,478	**74%**	300	1,509	0	141	81%
RCC8_25_24_1_UNSAT_13	97	651	31%	170	834	41%	171	751	38%	71	401	**19%**	71	431	0	50	23%
RCC8_25_24_4_SAT_16	1,180	1,177	98%	1,180	1,177	98%	1,179	1,176	98%	1,174	1,153	**97%**	1,156	1,152	3	261	107%
RCC8_25_24_4_UNSAT_10	511	515	42%	49	54	**4%**	139	159	12%	132	148	11%	131	150	0	31	13%
RCC8_25_12_1_SAT_3	1,205	944	**89%**	1209	942	89%	1,206	963	90%	1,219	941	90%	371	941	105	115	**63%**
RCC8_25_12_1_UNSAT_9	651	465	46%	449	239	28%	250	235	**20%**	382	277	27%	62	277	40	38	**17%**
RCC8_25_12_4_SAT_5	1,793	599	99%	1793	599	99%	1,794	599	99%	1,792	599	99%	617	599	147	147	**63%**
RCC8_25_12_4_UNSAT_5	1338	447	74%	989	372	56%	997	372	57%	593	209	**33%**	193	209	50	49	**20%**
RCC8_50_49_1_SAT_5	1,225	6,207	75%	1,225	6,404	78%	1,225	6,454	78%	1,225	6,150	**75%**	1,225	6,189	0	579	81%
RCC8_50_49_1_UNSAT_2	700	4,561	53%	466	2,660	32%	467	2,673	32%	191	1,204	**14%**	191	1,204	0	163	15%
RCC8_50_49_4_SAT_2	4,860	4,850	99%	4,860	4,852	99%	4,860	4,845	99%	4,859	4,850	**99%**	4,828	4,849	4	1,152	110%
RCC8_50_49_4_UNSAT_7	3,789	3,767	77%	3,062	3,038	62%	3,138	3,133	64%	707	2,264	**30%**	1,703	1,723	0	419	39%
RCC8_50_25_1_SAT_7	4,990	3,824	**90%**	4,999	3,852	90%	4,997	3,848	90%	4,996	3,839	90%	1,212	3,802	471	462	**60%**
RCC8_50_25_1_UNSAT_8	887	667	15%	776	536	13%	713	489	12%	188	145	**3%**	20	145	21	20	**2%**
RCC8_50_25_4_SAT_9	7,253	2,522	99%	7,253	2,522	99%	7,250	2,523	99%	7,253	2,522	99%	2,510	2,524	593	616	**63%**
RCC8_50_25_4_UNSAT_4	3,378	1,120	45%	696	217	**9%**	704	217	9%	1,117	349	15%	381	349	92	89	**9%**
RCC8_100_99_1_SAT_3	4,950	29,003	87%	4,950	28,905	85%	4,950	28,994	85%	4,950	28,980	**85%**	4,950	28,986	0	3,248	93%
RCC8_100_99_1_UNSAT_3	3,883	21,209	63%	4,537	23,481	70%	4,549	23,619	66%	1,951	11,374	**33%**	1,951	11,374	0	1,281	36%
RCC8_100_50_1_SAT_3	21,154	16,459	95%	21,137	16,441	95%	21,118	16,439	94%	21,154	16,482	95%	722	4,218	625	590	**15%**
RCC8_100_50_1_UNSAT_3	18,368	14,486	83%	9,729	7,479	43%	10,310	7,832	45%	19,325	15,055	86%	3,788	15,052	1,943	1,997	**57%**

Table 3: Comparison GEQCA-I using different heuristics vs GEQCA-II using the best heuristic on RCC8 instances. eF: Oracle effort, $\mathtt{Q}^+$: positive qualitative queries, $\mathtt{Q}^-$: negative qualitative queries, $\mathtt{UQ}^+$: positive universal queries, $\mathtt{UQ}^-$: negative universal queries.

time between two queries, denoted as T_{max}. Each scheduling instance is labeled by the number of tasks it contains (e.g., `sch_30_1` refers to instance 1 with 30 tasks).

Table 4 provides insights into the performance of GEQCA-I and GEQCA-II on scheduling instances. Firstly, we observe that GEQCA-II significantly reduces the oracle effort on all instances when $\mathcal{K}= \varnothing$. For instance, on `sch_60_1`, GEQCA-II requires only 12% of the effort needed by GEQCA-I (99%), which is due to the presence of independent tasks in these instances. This highlights the usefulness of the universal query in establishing task independence (e.g., more than $1,600$ $\texttt{UQ}^+$ on `sch_60_1`). Secondly, using the problem structure K_1 in the propagate procedure reduces the effort of the oracle for GEQCA-I. This reduction corresponds to a 38% average decrease in the number of queries (i.e., 30,528 queries). This reduction is significant when tasks are classified as locally independent using universal queries but not globally. In such cases, K_1 drastically reduces the number of qualitative queries asked by GEQCA-I. However, this reduction has no effect on GEQCA-II, where globally inconsistent relations are removed using `PC` during the acquisition process. Finally, we observe that the oracle effort is reduced when background knowledge is used, especially with known constraints such as cumulative and deadline constraints ($\mathcal{K}= K_1 \wedge K_2$), for both GEQCA-I and GEQCA-II. For instance, on `sch_60_1`, the effort was reduced from 99% to 62% with GEQCA-I and from 12% to 10% with GEQCA-II.

Using $K_1 \wedge K_2$ in the solve procedure, in addition to using K_1 in the propagate procedure, results in a small but insignificant improvement (average effort of 56% instead of 59% using GEQCA-I and average effort of 11% instead of 14% using GEQCA-II). However, under $\mathcal{K}= K_1 \wedge K_2$, the waiting time between two queries can reach the cutoff of one hour on `sch_60_3` due to the solve procedure that spends more than an hour trying to prove the consistency of a relation with the network. Note that with $\mathcal{K}= \varnothing$, the waiting time exceeds 7 seconds with a `PC` call, while the propagate procedure can increase the waiting time to more than 9 seconds under $\mathcal{K}$=K1.

Comparing GEQCA-II to GEQCA-I, we observe that GEQCA-II spends less time between two queries than GEQCA-I does on most instances. For instance, on `sch_30_2`, T_{max} reaches 393 seconds using GEQCA-I, whereas it is only 187 seconds using GEQCA-II. This is due to the higher number of calls of the solve procedure in GEQCA-I compared to GEQCA-II. The universal queries used in GEQCA-II prevent calling the solver on independent tasks, which reduces the waiting time.

4.4 [RQ4]: Time Limit Impact on GEQCA-II

In this section, we evaluate the performance of the oracle with respect to a time limit between two queries. As presented in Table 4, the waiting time (T_{max}) can sometimes exceed the one-hour limit, which could be impractical in some scenarios. In (Lallemand & Gronier, 2012), it is suggested that a cutoff time of 2 seconds is reasonable for human users. Therefore, we conducted a new experiment with a cutoff time of 2 seconds using GEQCA-II. Table 5 displays the results of this experiment.

Table 5 presents the oracle effort on five scheduling instances using GEQCA-I with the Path-Lex heuristic, a cutoff time of 2 seconds, and $\mathcal{K}$ values of $\varnothing$, K_1, and $K_1 \wedge K_2$. Interestingly, we observe that the oracle effort remains unchanged with or without a cutoff, as shown by comparing Table 4 to Table 5. We analyzed the two learning processes

Instance	GEQCA-I						GEQCA-II					
	$\varnothing$		K_1		$K_1 \wedge K_2$		$\varnothing$		K_1		$K_1 \wedge K_2$	
	eF	T_{max}	eF	T_{max}	eF	T_{max}	eF	T_{max}	eF	T_{max}	eF	T_{max}
sch_30_1	95%	**0.79**	55%	0.91	53%	1.18	**17%**	0.86	**17%**	**0.83**	**13%**	**0.92**
sch_30_2	98%	**0.62**	52%	0.90	48%	393.08	**17%**	0.78	**17%**	**0.83**	**12%**	**186.80**
sch_60_1	99%	**4.80**	65%	8.51	62%	13.00	**12%**	5.82	**12%**	**7.16**	**10%**	**7.83**
sch_60_2	99%	7.72	64%	8.08	61%	12.55	**12%**	**4.67**	**12%**	**6.88**	**10%**	**6.95**
sch_60_3	98%	5.94	59%	9.66	57%	3,600	**12%**	**5.63**	**12%**	**7.47**	**10%**	3,600

Table 4: GEQCA-I vs GEQCA-II acting on RCPSP instances (with cutoff $= 3,600s$, T_{max} in seconds, eF: oracle effort).

Instance	GEQCA-II					
	$\varnothing$		K_1		$K_1 \wedge K_2$	
	eF	T_{max}	eF	T_{max}	eF	T_{max}
sch_30_1	17%	0.86	17%	0.83	13%	0.92
sch_30_2	17%	0.78	17%	0.83	12%	2
sch_60_1	12%	2	12%	2	10%	2
sch_60_2	12%	2	12%	2	10%	2
sch_60_3	12%	2	12%	2	10%	2

Table 5: GEQCA-II acting on RCPSP instances (with cutoff $= 2s$, T_{max} in seconds, eF: oracle effort).

and found that GEQCA-II efficiently prunes the basic relations using PC within the cutoff time. Subsequently, the PC propagation spends the remaining time proving the transitive closure of the network. Propagating the problem structure with $\mathcal{K} = K_1$ is also not a time-consuming process. We also observed that the oracle effort is the same even when using $\mathcal{K} = K_1 \wedge K_2$ on sch_60_2 and sch_60_3. This is because the solve procedure can be time-consuming without removing any relation, either because a solution is found after a significant time resolution (e.g., 187 seconds on sch_30_2), or because the time limit is reached without proving the existence of a solution (e.g., one hour on sch_60_3). These observations demonstrate that GEQCA-II is an effective CA-Agent even when subjected to a time limit on the time window between two submitted queries.

The RCPSP instances are formulated using *precedes* and *meets* relations. Additionally, we note that GEQCA-I and GEQCA-II learn twice as many *precedes* constraints compared to the ones in the benchmarks, thanks to the PC procedure which reduces a considerable number of *full* constraints to *precedes*. This leads to a faster optimal solution search using networks obtained from GEQCA-I and GEQCA-II compared to the ones in the benchmarks.

4.5 Discussion and Limits

In conclusion, our revised approach GEQCA-II shows a higher effectiveness in learning qualitative constraints as compared to the classical GEQCA-I and LQCN. This is mainly due to the new universal query that significantly reduces the number of queries raised to the oracle. The extension addresses the limitations of GEQCA-I in learning dense

instances and the qualitative part of real scheduling problems. Furthermore, both GEQCA-I and GEQCA-II offer a cutoff option to minimize the waiting time between two queries. Assigning the cutoff time to 2 seconds is a reasonable choice.

Note that GEQCA-II may not be suitable for interactive processes when a large number of queries (e.g., thousands) are required and only a single person can answer these queries. Hopefully, as said above in the paper, oracles can also be an automated system (equipped with a non-CP representation of the concept) or a crowd of users. In these cases, the oracle can only check the consistency of information with the concept being learned. For example, expert systems can easily verify the answers to queries, but they may not be able to solve complex queries. An example of an automated oracle in CA is provided in (Paulin, Bessiere, & Sallantin, 2008), where a CA-Agent and a robot interact to acquire sensorimotor behaviors. Another example is provided in (Menguy et al., 2022), where the oracle consists of a program under analysis and its postcondition, which automatically responds to the queries submitted by the CA-agent. It is also possible for the oracle to consist of a community, such as thousands of people or collaborative robots, who share the same concept and are capable of answering questions from any CA-agent (Lazaar, 2021). Furthermore, it is possible to inherit constraints from a past or obsolete model that needs updating. GEQCA-II can easily handle such cases with its incremental calculation of L.

There are several ways in which GEQCA-II can be improved:

- **Use other types of queries:** In some practical scenarios, the oracle can provide more informative answers to queries. For example, the oracle can provide all the relations that are allowed on a given pair or all pairs on which a given relation is allowed. Incorporating such query types could improve the efficiency of GEQCA-II.

- **Account for incorrect answers:** GEQCA-II assumes that the oracle always provides correct answers to queries. However, in practice, it may not be the case. To address this issue, we may consider that the oracle answers queries with incorrect answers according to a certain probability distribution. This could lead to a probabilistic version of qualitative CA, which would enable the acquisition of soft constraints enforced with probability restrictions.

- **Constraint selection heuristic:** While Path-Lex and Path-Weighted are effective in practice, they may not be suitable for all problems. Combining these heuristics with other dynamic entity-selection heuristics could be interesting. For example, using weights that are dynamically adjusted based on the current status of the acquisition process could be a promising approach.

5. Conclusion

In this paper, we presented GEQCA-II, an extension of GEQCA-I, a correct and generic active CA-Agent that can learn any qualitative network using qualitative queries, path consistency, and exploration of background knowledge. The key contribution of GEQCA-II is the incorporation of a new type of query called the universal query, which helps to reduce the number of queries required and improves the efficiency of the CA process. Our experimental results demonstrate that GEQCA-II requires only a limited number of queries

to the oracle within a reasonable amount of time, making it a practical and efficient approach. We also showed that GEQCA-II overcomes the limitations of GEQCA-I and LQCN on dense instances. Furthermore, we highlighted the generality of GEQCA-I and GEQCA-II, which can be used to learn not only temporal but also spatial networks. Overall, GEQCA-II presents a promising approach to active constraint acquisition in qualitative reasoning. Future work could explore further enhancements with new query types and real-world applications. The generality of GEQCA-I and GEQCA-II opens up avenues for their application in various domains beyond qualitative reasoning.

Acknowledgments

This work was funded by the T-LARGO and AutoCSP projects of the Norwegian Research Council, grant numbers 274786 and 324674, as well as projects from the European Union's Horizon research and innovation programme (FAIRiCUBE under grant agreement No 101059238, AI4CCAM under grant agreement No 101076911 and TAILOR under grant agreement No 952215). We would also like to express our gratitude to the reviewers for their constructive and detailed feedback on our manuscript. Their comments helped improving the quality of our work.

References

Addi, H. A., Bessiere, C., Ezzahir, R., & Lazaar, N. (2018). Time-bounded query generator for constraint acquisition. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR 2018)*, Vol. 10848 of *Lecture Notes in Computer Science*. Springer.

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

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

Arcangioli, R., Bessiere, C., & Lazaar, N. (2016). Multiple constraint aquisition. In *IJCAI: International Joint Conference on Artificial Intelligence*, pp. 698–704.

Barták, R., Salido, M., & Rossi, F. (2008). Constraint satisfaction techniques in planning and scheduling. *Journal of Intelligent Manufacturing*, *21*, 5–15.

Belaid, M.-B., Belmecheri, N., Gotlieb, A., Lazaar, N., & Spieker, H. (2022). GEQCA: Generic qualitative constraint acquisition. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 36, pp. 3690–3697.

Beldiceanu, N., & Simonis, H. (2012). A Model Seeker: Extracting global constraint models from positive examples. In *International Conference on Principles and Practice of Constraint Programming (CP 2012)*, pp. 141–157. Springer.

Beldiceanu, N., & Simonis, H. (2016). ModelSeeker: Extracting Global Constraint Models from Positive Examples. In Bessiere, C., Raedt, L. D., Kotthoff, L., Nijssen, S., O'Sullivan, B., & Pedreschi, D. (Eds.), *Data Mining and Constraint Programming*, Vol. 10101 of *Data Mining and Constraint Programming*, pp. 77–95. Springer.

Belhadji, S., & Isli, A. (1998). Temporal Constraint Satisfaction Techniques in Job Shop Scheduling Problem Solving. *Constraints*, *3*(2), 203–211.

Bessiere, C., Coletta, R., Hebrard, E., Katsirelos, G., Lazaar, N., Narodytska, N., Quimper, C.-G., & Walsh, T. (2013). Constraint acquisition via partial queries. In *Twenty-Third International Joint Conference on Artificial Intelligence*.

Bessiere, C., Coletta, R., Koriche, F., & O'Sullivan, B. (2005). A SAT-based version space algorithm for acquiring constraint satisfaction problems. In *European Conference on Machine Learning*, pp. 23–34. Springer.

Bessiere, C., Coletta, R., O'Sullivan, B., & Paulin, M. (2007). Query-driven constraint acquisition. In *IJCAI*, Vol. 7, pp. 50–55.

Bessiere, C., Koriche, F., Lazaar, N., & O'Sullivan, B. (2017). Constraint acquisition. *Artificial Intelligence*, *244*, 315–342.

Bshouty, N. H. (2018). Exact learning from an honest teacher that answers membership queries. *Theoretical Computer Science*, *733*, 4–43.

Crainic, T. G., Perboli, G., & Tadei, R. (2012). *Recent Advances in Multi-dimensional Packing Problems*, chap. 5. IntechOpen.

De Raedt, L., Passerini, A., & Reso, S. (2018). Learning constraints from examples. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, pp. 7965—7970.

Gantner, Z., Westphal, M., & Wölfl, S. (2008). GQR - A Fast Reasoner for Binary Qualitative Constraint Calculi. In *AAAI Workshop on Spatial and Temporal Reasoning*, p. 6.

Hartmann, S., & Briskorn, D. (2010). A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of operational research*, *207*(1), 1–14.

Krokhin, A., Jeavons, P., & Jonsson, P. (2003). Reasoning about temporal relations: The tractable subalgebras of allen's interval algebra. *J. ACM*, *50*(5), 591–640.

Lallemand, C., & Gronier, G. (2012). Enhancing user experience during waiting time in hci: contributions of cognitive psychology. In *Proceedings of the Designing Interactive Systems Conference*, pp. 751–760.

Lazaar, N. (2021). Parallel constraint acquisition. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI*, pp. 3860–3867. AAAI Press.

Mackworth, A. K. (1977). Consistency in networks of relations. *Artificial intelligence*, *8*(1), 99–118.

Maddux, R. D. (1994). Relation algebras for reasoning about time and space. In Nivat, M., Rattray, C., Rus, T., & Scollo, G. (Eds.), *Algebraic Methodology and Software Technology (AMAST'93)*, pp. 27–44, London. Springer London.

Menguy, G., Bardin, S., Lazaar, N., & Gotlieb, A. (2022). Automated program analysis: Revisiting precondition inference through constraint acquisition. In Raedt, L. D. (Ed.), *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, pp. 1873–1879. International Joint Conferences on Artificial Intelligence Organization.

Mouhoub, M., Marri, H. A., & Alanazi, E. (2018). Learning qualitative constraint networks. In Alechina, N., Nørvåg, K., & Penczek, W. (Eds.), *25th International Symposium on Temporal Representation and Reasoning, TIME 2018, Warsaw, Poland, October 15-17, 2018*, Vol. 120 of *LIPIcs*, pp. 19:1–19:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Mouhoub, M., Marri, H. A., & Alanazi, E. (2021). Exact learning of qualitative constraint networks from membership queries..

Nebel, B. (1997). Solving hard qualitative temporal reasoning problems: Evaluating the efficiency of using the ORD-Horn class. *Constraints*, *1*(3), 175–190.

Ore, O. (1987). *Theory of Graphs*. American Mathematical Society colloquium publications. American Mathematical Society.

Paulin, M., Bessiere, C., & Sallantin, J. (2008). Automatic design of robot behaviors through constraint network acquisition. In *2008 20th IEEE International Conference on Tools with Artificial Intelligence*, Vol. 1, pp. 275–282. IEEE.

Pawlak, T. P., & Krawiec, K. (2017). Automatic synthesis of constraints from examples using mixed integer linear programming. *European Journal of Operational Research*, *261*(3), 1141–1157.

Prestwich, S. D., Freuder, E. C., O'Sullivan, B., & Browne, D. (2021). Classifier-based constraint acquisition. *Annals of Mathematics and Artificial Intelligence*, *89*(7), 655–674.

Randell, D. A., Cui, Z., & Cohn, A. G. (1992). A spatial logic based on regions and connection.. *KR*, *92*, 165–176.

Renz, J., & Nebel, B. (2001). Efficient Methods for Qualitative Spatial Reasoning. *Journal of Artificial Intelligence Research*, *15*, 289–318.

Tsouros, D. C., Stergiou, K., & Bessiere, C. (2019). Structure-driven multiple constraint acquisition. In *International Conference on Principles and Practice of Constraint Programming*, pp. 709–725. Springer.

Tsouros, D. C., Stergiou, K., & Bessiere, C. (2020). Omissions in constraint acquisition. In Simonis, H. (Ed.), *Principles and Practice of Constraint Programming*, Vol. 12333 of *Lecture Notes in Computer Science*, pp. 935–951. Springer.

Valiant, L. G. (1984). A theory of the learnable. *Communications of the ACM*, *27*(11), 1134–1142.

van Beek, P., & Manchak, D. W. (1996). The design and experimental analysis of algorithms for temporal reasoning. *Journal of Artificial Intelligence Research*, *4*.

Vilain, M., & Kautz, H. (1986a). Constraint propagation algorithms for temporal reasoning. In *Proceedings of the Fifth AAAI National Conference on Artificial Intelligence*, AAAI'86, pp. 377–382.

Vilain, M. B., & Kautz, H. A. (1986b). Constraint propagation algorithms for temporal reasoning.. In *AAAI*, Vol. 86, pp. 377–382.