# GEQCA: Generic Qualitative Constraint Acquisition

**Mohamed-Bachir Belaid,[1] Nassim Belmecheri,[2,3] Arnaud Gotlieb,[1] Nadjib Lazaar[3] and Helge Spieker[1]**

[1]Simula Research Laboratory, Oslo, Norway
[2]LITIO, University of Oran 1, Oran, Algeria
[3]LIRMM, University of Montpellier, CNRS, Montpellier, France
{bachir@simula.no, belmecheri.nassim@edu.univ-oran1.dz, arnaud@simula.no, lazaar@lirmm.fr, helge@simula.no}

## Abstract

Many planning, scheduling or multi-dimensional packing problems involve the design of subtle logical combinations of temporal or spatial constraints. On the one hand, the precise modelling of these constraints, which are formulated in various relation algebras, entails a number of possible logical combinations and requires expertise in constraint-based modelling. On the other hand, active constraint acquisition (CA) has been used successfully to support non-experienced users in learning conjunctive constraint networks through the generation of a sequence of queries. In this paper, we propose GEQCA, which stands for *Generic Qualitative Constraint Acquisition*, an active CA method that learns qualitative constraints via the concept of *qualitative queries*. GEQCA combines qualitative queries with time-bounded path consistency (PC) and background knowledge propagation to acquire the qualitative constraints of any scheduling or packing problem. We prove soundness, completeness and termination of GEQCA by exploiting the *jointly exhaustive and pairwise disjoint* property of qualitative calculus and we give an experimental evaluation that shows (i) the efficiency of our approach in learning temporal constraints and, (ii) the use of GEQCA on real scheduling instances.

## Introduction

Reasoning about time and space is crucial for many practical problems including automated planning (Belhadji and Isli 1998), scheduling (Barták, Salido, and Rossi 2008), or multi-dimensional packing problems (Crainic, Perboli, and Tadei 2012). In that context, qualitative calculus provide an algebraic framework that establishes relations between pairs of entities through a language that is *jointly exhaustive and pairwise disjoint*. Examples of qualitative calculus include (and are not limited to) Point Algebra (Vilain and Kautz 1986b) or Allen's interval algebra (Allen 1983) to reason about temporal tasks, Region Connection Calculus (RCC) (Randell, Cui, and Cohn 1992) to reason about topological relationships between spatial regions.

In this context, constraint satisfaction techniques and Constraint Programming (CP) are convenient frameworks to model and solve qualitative constraint networks. Nevertheless, in many practical situations, complex problems are not formulated as constraint networks and are only solved via historical records and hand-crafted solutions. Moreover, inter-relationships among entities may only be known on a local and pair-wise level, but not all implications for other pairs of entities are known. To ease the modelling of CP problems, Bessiere *et al.* introduced the framework of *constraint acquisition* (CA) to learn CP models either via *passive learning* from a set of labelled example assignments (Bessiere et al. 2005) or via *active learning* with specific queries that help to classify complete assignments (Bessiere et al. 2007). State-of-the-art active CA algorithms include: QUACQ (Bessiere et al. 2013), a query-directed learning approach also known as *exact learning* (Bshouty 2018), which interacts with the user by asking either complete or partial queries to reduce the set of possible satisfiable constraints from a given constraint language; MULTIACQ (Arcangioli, Bessiere, and Lazaar 2016), which extends QUACQ by learning a maximum number of constraints violated by a given negative example; T-QUACQ (Addi et al. 2018), which bounds the query-generation time in order to speed-up CA; MQUACQ-2 (Tsouros, Stergiou, and Bessiere 2019), which exploits the structure of learned models by focusing queries on quasi-cliques of constraints; *and* CLASSACQ (Prestwich et al. 2021), which uses a Naive Bayes classifier to discriminate solutions from non-solutions and exploiting this to passively acquire constraint models.

Other key approaches to learning constraint models include ModelSeeker (Beldiceanu and Simonis 2012, 2016), which is able to acquire global constraints or the general framework of *constraint learning* (De Raedt, Passerini, and Reso 2018) or *constraint synthesis*, which is based on mixed linear integer programming (Pawlak and Krawiec 2017). The issue of handling errors in a user's answer to a query is also at the heart of the most recent development of interactive CA (Tsouros, Stergiou, and Bessiere 2020). Standard active CA algorithms cannot easily address qualitative constraints because handling disjunctions of relations leads to a combinatorial explosion of the constraint basis size. 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 key aspect of the adoption of CA in practical applications (Bessiere et al. 2017). Learning qualitative temporal constraints has been initiated by (Mouhoub, Marri, and Alanazi 2018) in LQCN.

LQCN follows the active learning version of CONACQ by considering each qualitative constraint between two 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. Nevertheless, LQCN is limited to Allen's algebra and its generalization is not possible without a proof of correctness. Furthermore, the background knowledge considered in LQCN is limited to a composition table, while the user usually knows more information about the problem (e.g., tasks duration, resources limit, pre-crafted constraints). These elements are crucial to complete the constraint propagation step and further filter the queries to be generated.

In this paper, we introduce *Generic Qualitative Constraint Acquisition* (GEQCA), a novel generic active CA algorithm for learning any kind of qualitative constraints between each pair of entities of a specific problem. Our contribution is threefold:

1. We introduce the concept of *qualitative query* and provide a generic algorithm, GEQCA, that combines qualitative queries, time-bounded path consistency (PC), PATH a novel heuristic and extended background knowledge propagation to acquire any qualitative constraints in constraint acquisition;
2. We prove soundness, completeness and termination of GEQCA by exploiting the "jointly exhaustive and pairwise disjoint" (JEPD) property of qualitative calculus;
3. We provide an implementation of GEQCA with strategies and we experimentally evaluate the benefit of GEQCA on randomly generated temporal instances and on real scheduling instances.

To the best of our knowledge, this is the first time a generic and correct method is introduced within CA to handle qualitative networks. Furthermore, this is the first time in CA a CP model is solved to reduce the number of queries using the extended background knowledge.

## Background

A (binary) *constraint network* is a set $C$ of constraints on the vocabulary $(X, D)$, where $X$ is a set of $n$ variables and $D$ a specified finite domain. In our context, $X$ represents a set of $n$ entities (points, intervals, regions, etc.) to place in a finite temporal/spatial space $D$. For instance, for intervals the variable $X_i$ is a pair of endpoints $(X_i^-, X_i^+)$ in a specified finite domain $D$, where $X_i^- < X_i^+$ holds. The constraint network $C$ is built on $\Gamma$ language. $\Gamma$ represents a finite set of time/space relational operators (e.g., *before*, *after*, *contains*, *part-of*, etc.), where $\Gamma = \{r_1, \ldots, r_m\}$ and $|\Gamma| = m$. The atomic relations in $\Gamma$ are (i) non-empty and pairwise disjoint, and (ii) the union of the $m$ relations forms the universal one. Then, $\Gamma$ is *a jointly exhaustive and pairwise disjoint* language (JEPD property). Bear in mind that the relational algebra generated by $\Gamma$ is finite and it is closed under composition, converse and contains the identity. A binary constraint $C_{ij} \in C$, denoted by $C_{ij} = ((X_i, X_j), \{r_{k_1}, \ldots, r_{k_d}\})$, is a disjunction of $d$ atomic relations in $\Gamma$. 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* and by $\top$ the *universal* constraint.

In constraint acquisition (CA) the learner and the user share a common knowledge to communicate altogether, which is materialized by the vocabulary $(X, D)$. We denote by $size(C) = \sum_{C_{ij} \in C} |C_{ij}|$ the total number of atomic relations in $C$. An assignment $e \in D^X$, *is rejected by* a constraint network $C$ iff we have at least one constraint $C_{ij}$, rejecting $e_{ij}$, the projection of $e$ on $(X_i, X_j)$. If $e_{ij}$ does not violate $C_{ij}$, then $e_{ij} \models C_{ij}$. An assignment $e$ on $X$ that is accepted by $C$ is a *solution* of $C$. We write $sol(C)$ for the set of solutions of $C$. In addition to the vocabulary, the learner owns the language from which it can build constraints on specified sets of entity variables. Bear in mind that standard version space-based approaches like CONACQ and QUACQ are not suitable for use in qualitative reasoning, where only languages closed by conjunction on quantitative constraints are considered. For that, such systems need to build a constraint basis from $\Gamma$ on the vocabulary $(X, D)$, which leads to a basis of size $(\frac{2^m n(n-1)}{2})$.

Given a vocabulary $(X, D)$, a *qualitative concept* is a Boolean function $f$ over $D^X$, that is, a map that assigns to each assignment $e$ a value in $\{0, 1\}$. A *representation* of a concept $f$ is a constraint network $C$ for which $f^{-1}(1) = sol(C)$, denoted $f = sol(C)$. A *user qualitative concept* is a concept $f_Q$ that returns 1 for $e$ if and only if $e$ is a solution of the problem that the user has in mind.

We now define *convergence*, which is the CA problem we are interested in. Given a set $E$ of examples labelled by the user as *yes* or *no*, we say that a network $C$ agrees with $E$ if $C$ accepts all examples labelled *yes* in $E$ and does not accept those labelled *no*. The learning process has *converged* on the network $L$ if **(i)** $L$ agrees with $E$ and **(ii)** for every other network $L'$ agreeing with $E$, we have $sol(L') = sol(L)$. It is thus guaranteed that $sol(L) = f_Q$. If there does not exist any $L$ such that $L$ agrees with $E$, we say that we have *collapsed*. This happens when $f_Q$ is an infeasible concept.

In practical applications, it is often the case that we already know particular parts of the problem. This set of known information (problem structure, quantitative constraints, a model of durative tasks/actions and their preconditions/effects, etc.) represents a background knowledge, noted $\mathcal{K}$. From $\mathcal{K}$ we can deduce a qualitative knowledge $\mathcal{K}_Q$ by using the propagation and/or resolution processes. Hence, $\mathcal{K}_Q$ is subsumed by the concept to learn $f_Q$ (i.e., $\mathcal{K}_Q \subseteq f_Q$).

## GEQCA: Constraint Acquisition via Qualitative Queries

We propose GEQCA, a generic constraint acquisition algorithm that can learn qualitative constraints. For this, we introduce the concept of a *qualitative query*.

**Definition 1** (Qualitative Query)**.** *Given a pair of entity variables $(X_i, X_j)$ and a basic relation $r \in \Gamma$, a qualitative query* Q-ASK$(X_i, r, X_j)$ *is answered yes by the user if and only if $X_i$ can be placed under $r$ wrt $X_j$.*

A classified qualitative query is called a positive or negative *example* depending on whether Q-ASK$(X_i, r, X_j)$

is *yes* or *no*. It is important to observe that "Q-ASK$(X_i, r, X_j)$=*yes* " does not mean that $(X_i, r, X_j)$ extends to a solution of the qualitative concept to learn, which would put an NP-complete problem on the shoulders of the user (Vilain and Kautz 1986a; Maddux 1994). The rationale behind GEQCA is to learn constraints between entities relying on the JEPD property of $\Gamma$.

## Description of GEQCA

GEQCA (see Algorithm 1) takes as input a vocabulary $(X, D)$ of $n$ entity variables, the $\Gamma$ language of binary atomic relations, a background knowledge $\mathcal{K}$ and a timed boundary parameter `cutoff`. First, GEQCA initializes the possible constraints between entities of the network $L$ to the universal constraint $\top$ (line 4). $\tau$ is set to `cutoff` to ensure that, between two asked queries, the waiting time does not exceed the given time boundary (`cutoff`). Afterwards, GEQCA loops on $L$ to reduce the constraints to sets of atomic relations equivalent to the user's concept $f_Q$ (lines 6-17). Once $C_{ij}$ selected, GEQCA uses a *propagate* procedure, in $\tau$, to reduce $C_{ij}$ by propagating $\mathcal{K}$ on it (line 7). For instance, in a temporal context, if *"$X_i$ and $X_j$ tasks have respectively, duration of* 1 *and* 2 *hours"*, which is deduced from $\mathcal{K}$, the *propagate* procedure will remove *Equals*, *Contains*, *Started-by* and *Finished-by* from $X_i$ to $X_j$. Then, $\tau$ is updated by deducing the amount of time used in $\mathcal{K}$ propagation. Once completed, GEQCA iterates on the remaining atomic relations of $C_{ij}$ (lines 9-15). Each atomic relation $r$ is first checked, in $\tau$, if it is consistent with the provided $\mathcal{K}$ (line 10). If the relation is not consistent,[1] we can remove it at line 13. Otherwise, the resolution finds a solution or $\tau$ is not sufficient, and in these two cases the basic relation is presented to the user under a qualitative query Q-ASK$(X_i, r, X_j)$ in line 12. Then, $\tau$ is updated by deducing the amount of time used in the *solve* process (line 11). If the user answers *no*, we remove $r$ from $C_{ij}$ (line 13). If $C_{ij}$ is reduced at line 7 or at line 13, GEQCA eliminates non-feasible relations from $L$ by maintaining path consistency using the `PC` function and in a time not exceeding $\tau$. If $C_{ij}$ is reduced to the empty constraint $\perp$ in line 13 or if an inconsistency is reported by `PC` (lines 23 and 24), this means that the space of possible networks collapses because of an infeasible concept ($f_T \equiv \perp$) (line 17).

In lines 18-28, we present the `PC` function (Mackworth 1977). The `PC` function is a time-bounded path consistency propagator of a cubic complexity in number of entity variables. `PC` returns $true$ when a fixpoint (transitive closure) is reached by propagating qualitative knowledge, or when $\tau$ is elapsed, and $false$ if any inconsistency is detected.

## Theoretical Analysis

We first show that an acquisition using GEQCA (Algorithm 1) is a correct algorithm to learn any constraint network representing a qualitative concept over $\Gamma$ language with a waiting time between two queries not exceeding a given time boundary. For the following propositions and Theorem 1, we have a vocabulary $(X, D)$ of $n$ entity variables, the set $\Gamma$, a qualitative concept $f_Q$ and a knowledge $\mathcal{K}_Q \subseteq f_Q$ subsumed by a given background knowledge $\mathcal{K}$.

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

*Proof.* Suppose there exists $e \in f_Q \setminus sol(L)$. Here, $e \not\models L$, which means that there exists at least one constraint $C_{ij} \in L$ with the projection of the solution $e_{ij} \not\models C_{ij}$. At this point, we conclude that a relation $r^*$ missing in a given $C_{ij}$ to have $e_{ij} \models C_{ij}$. That is, $!\exists r^* \notin C_{ij} : e_{ij} \models X_i\{r^*\}X_j$ (a unique relation because of the JEPD property of $\Gamma$ language). Knowing that $propagate$ procedure (line 7) and consistency check (line 10) are based on a $\mathcal{K}$ subsumed by $f_Q$ ($\mathcal{K}_Q \subseteq f_Q$) and that the `PC` function is sound (Allen 1983; Vilain and Kautz 1986a), the unique place where $r^*$ can be removed from $C_{ij}$ is at line 13 because of a negative answer on a qualitative query. Knowing that a qualitative query on a consistent relation between a given pair of entities cannot be answered by *no*, we deduce that removing an atomic relation from a given constraint in $L$ cannot reject an example accepted by $f_Q$. □

Note that the soundness property shows that GEQCA returns a constraint network $L$ not containing false negatives (only consistent constraints). That is, there exists a constraint network $C_Q$ representing $f_Q$, where $L \subseteq C_Q$.

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

*Proof.* Suppose there exists $e \in sol(L) \setminus f_Q$. Here, $L$ is accepting solutions rejected by the concept $f_Q$. It means that we need to restrict at least one constraint $C_{ij} \in L$ to have $e \not\models L$. Restricting $C_{ij}$ means removing at least one relation $r^*$ from $C_{ij}$. If the relation $r^*$ is not removed using $\mathcal{K}$ propagation (line 7 and 10) or using `PC`, it is presented to the user under a qualitative query. As $r^*$ is inconsistent on $(X_i, X_j)$, the qualitative query on $r^*$ can only be answered by *no*. We deduce that keeping an atomic relation in a given constraint in $L$ cannot accept an example rejected by $f_Q$. □

The completeness property shows that GEQCA returns a constraint network $L$ not missing a true positive (a consistent constraint). That is, there exists a constraint network $C_Q$ representing $f_Q$, where $C_Q \subseteq L$.

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

*Proof.* GEQCA iterates on all pairs of entities (i.e., $\frac{n(n-1)}{2}$ iterations, line 6), and for each pair of entities, GEQCA iterates on the language $\Gamma$ line 9. Since the pair of entities and $\Gamma$ have finite size, $propagate$ and $solve$ are time-bounded procedures, and the `PC` function is time-bounded and/or cubic, we have termination. □

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

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

[1] if $\tau$ is sufficient to prove that there is no solution: $s = \varnothing$.

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

*Proof.* If `cutoff` $= \infty$, it is trivial. Suppose now that `cutoff` $< \infty$, at each iteration of GEQCA main loop (line 6), $propagate$, $solve$ and `PC` are executed in a time $t < \tau$. $\tau$ is initialized to `cutoff` value (line 5) and only decreased by the $update$ function after each call of the three procedures/functions. If a qualitative query is asked, we reset $\tau$ to `cutoff` at line 15. Now, if $\tau$ is reduced to 0, GEQCA will ask a query at the next iteration where $propagate$, $solve$, and `PC` are not called; then we reset $\tau$ to `cutoff`. Thus, the waiting time between two queries will never exceed the given time bound `cutoff`. □

**Complexity of GEQCA in terms of queries.** We analyse the complexity of GEQCA in terms of the number of qualitative queries it can submit to the user. The queries are proposed to the user in line 12 of Algorithm 1.

Given a vocabulary $(X, D)$ of $n$ entity variables, the language $\Gamma$ of $m$ atomic relations, a qualitative concept $f_Q$, a qualitative knowledge $\mathcal{K}_Q \subseteq f_Q$ subsumed by a given background knowledge $\mathcal{K}$, and a number of atomic relations pruned using $\mathcal{K}$ and `PC` propagation noted $k$. GEQCA asks $O(d)$ positive queries and $O(\frac{mn(n-1)}{2} - (d+k))$ negative queries to prove convergence on a constraint network $L$ of $size(L) = d$, or to collapse. The convergence is obtained in GEQCA once any possible atomic relation between two entities is visited. An atomic relation can be visited by a qualitative query or by a remove action (using *propagate*, *solve* or `PC`). Note that we have $\frac{n(n-1)}{2}$ constraints of $m$ atomic relations. If *propagate*, *solve* and `PC` remove $k$ relations, the total number of queries is $O(\frac{mn(n-1)}{2} - k)$. An atomic relation in $L$ after convergence is a relation that is not removed with *propagate*, *solve* or `PC`, and validated with a positive qualitative query. That is, the number of positive queries is equal to $size(L)$. It follows that the number of negative queries required for convergence is bounded above by $O(\frac{mn(n-1)}{2} - (d+k))$.

## Strategies

GEQCA can be improved by making the constraint selection at line 6 less brute-force. That is, selecting constraints in different ways can have a significant impact on $\mathcal{K}$ and `PC` propagation, thus leading to great improvements in the number of asked queries until convergence. We introduce a dedicated constraint selection heuristic based on traversal of a complete graph. PATH heuristic first selects a random constraint $C_{ij}$ in $L$ and then selects a connected one ($C_{ki}$ or $C_{jk}$), otherwise it again selects a random one. As the initial constraint network $L$ is a complete graph of universal constraints as edges, we visit all constraints (edges) of $L$ by forming a maximum number of paths. The rationale behind is to maximize the impact of PC and the transitivity between constraints with a constraint selector building paths.

**Algorithm 1:** GEQCA: Constraint Acquisition via Qualitative Queries.

1 **In**: vocabulary $(X, D)$; $\Gamma$ language; background knowledge $\mathcal{K}$; parameter `cutoff`;
2 **Out**: a learned network $L$;
3 **begin**
4 $L \leftarrow \{C_{ij} = \top : i < j\}$;
5 $\tau \leftarrow$ `cutoff`;
6 **foreach** $C_{ij} \in L$ **do**
7 CHANGE$\leftarrow propagate(\mathcal{K}, C_{ij}, \tau)$
8 $update(\tau)$;
9 **foreach** $r \in C_{ij}$ **do**
10 $s \leftarrow solve(L \cup \mathcal{K} \cup X_i\{r\}X_j, \tau)$
11 $update(\tau)$;
12 **if** $(s = \varnothing) \vee$ (Q-ASK$(X_i, r, X_j) = no$) **then**
13 $C_{ij} \leftarrow C_{ij} \setminus \{r\}$
14 CHANGE$\leftarrow$ **true**;
15 **if** $s \neq \varnothing$ **then** $\tau \leftarrow$ `cutoff`;
16 **if** $(C_{ij} = \bot) \vee$ (CHANGE $\wedge \neg PC(L, \tau)$ **then**
17 **return** *"collapse"*
18 **Function** `PC`$(L, \tau)$**:**
19 $Q \leftarrow L$
20 **while** $(Q \neq \varnothing) \wedge$ ($\tau$ *is not yet up*) **do**
21 pick $C_{ij}$ in $Q$
22 **foreach** $X_k \in X \setminus \{X_i, X_j\}$ **do**
23 $\Delta_1 \leftarrow C_{ik} \cap$ `composition`$(C_{ij}, C_{jk})$;
**if** $\Delta_1 = \bot$ **then return false**
24 $\Delta_2 \leftarrow C_{ki} \cap$ `composition`$(C_{ki}, C_{ij})$;
**if** $\Delta_2 = \bot$ **then return false**
25 **if** $C_{ik} \neq \Delta_1$ **then** $C_{ik} \leftarrow \Delta_1$;
$Q \leftarrow Q \cup \{C_{ik}\}$
26 **if** $C_{kj} \neq \Delta_2$ **then** $C_{kj} \leftarrow \Delta_2$;
$Q \leftarrow Q \cup \{C_{kj}\}$
27 **if** $Q = \varnothing$ **then** $update(\tau)$;
28 **return true**

# Experiments

In this section, we experimentally evaluate GEQCA, with time intervals as entities, on learning temporal constraints based on the Allen's interval algebra, where $|\Gamma| = 13$ (see Figure 1). The goal is 1) to compare our approach with LQCN (Mouhoub, Marri, and Alanazi 2018) 2) to evaluate our contributions for learning temporal networks and real-world scheduling problems. The basic version of GEQCA under time intervals as variables is similar to LQCN except for the used strategy of pairs selection. LQCN uses a weight-based heuristic from (van Beek and Manchak 1996). Unlike LQCN, GEQCA uses the novel PATH heuristic as presented in the previous section and GEQCA is a correct and generic CA algorithm that can learn any qualitative network. More precisely, our evaluation aims to answer the following research questions:

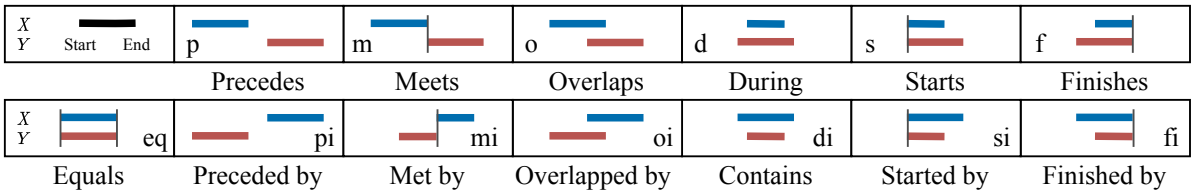



Figure 1: Overview of the 13 atomic relations in Allen's Interval Algebra (Allen 1983) as named in (Krokhin, Jeavons, and Jonsson 2003). These relations resemble the possible qualitative queries, the user is asked in GEQCA.

- **RQ1**: *Is* GEQCA *useful for learning temporal networks and how does it compare with* LQCN*?*
- **RQ2**: *Is* GEQCA *capable of learning the qualitative part of real-world scheduling problems?*

**Experimental Evaluation Protocol.** For the first experiment, we generated 12 random problem instances representing feasible temporal concepts (i.e., $f_Q \not\equiv \bot$). We consider $n$, the number of Time Intervals (TI) (10, 25, 50 and 100) and we randomly sample both a subset of pairs of TI and their atomic relations. Then, we maintain path consistency on the network and check its satisfiability. The density of an instance corresponds to the percentage of atomic relations present in each temporal constraint. For example, Ins_50_48 denotes the instance with 50 TI and a density of 48%, i.e. 6 of 13 relations are possible for each pair on average. GEQCA is implemented in Java and using Choco solver[2] for the *solve* procedure at line 10 in Algorithm 1. The code and full descriptions of each instance are publicly available at https://github.com/lirmm/ConstraintAcquisition/tree/GEQCA. All tests are run on an Intel core $i7$, $2.8GHz$ with RAM of $16GB$.

**[RQ1]: Effectiveness of GEQCA** For our first experiment, we use GEQCA with three basic features: 1) pairs in line 6 of Algorithm 1 are selected using the PATH heuristic; 2) $\mathcal{K}$ is empty; and 3) the PC function is run until a fixpoint is reached (cutoff $= \infty$).

Table 1 reports on the performance of GEQCA as compared to LQCN. In Table 1, we report the total number of positive queries $\mathtt{Q}^+$ for each instance. Note that $\mathtt{Q}^+$ are mandatory queries that validate the consistency of atomic relations of $L$ (i.e., $size(L) = \mathtt{Q}^+$). Hence, $\mathtt{Q}^+$ is similar for both approaches. On the other hand, the number of negative queries, $\mathtt{Q}^-$, depends on the strength of PC propagation. For each approach, on each instance, we report the number of negative queries, $\mathtt{Q}^-$, and the user effort eF, which represents the ratio of queries that need to be classified by the user according to $\mathtt{Q}_{\mathtt{max}}$, the maximum number of queries (i.e., $\mathtt{Q}_{\mathtt{max}} = \frac{13n(n-1)}{2}$ and eF=$(\mathtt{Q}^+ + \mathtt{Q}^-)/\mathtt{Q}_{\mathtt{max}}$).

For learning qualitative constraints (using LQCN or GEQCA), we observe a positive correlation between $\mathtt{Q}^+$ and eF. This is especially true when $\mathtt{Q}^+$ are queries that must be submitted to the user to validate the learned network. However, we observe a negative correlation between $\mathtt{Q}^-$ and eF. This is explained by the fact that the atomic relations removed using PC (less user effort) can only be classified as negatives if submitted to the user. For instance, using GEQCA, the user effort on Ins_25_8 is 17% with 8% effort on positives and 9% on negatives. On the other hand, the user effort on Ins_25_58 is 78% with 59% effort on positives and 19% on negatives.

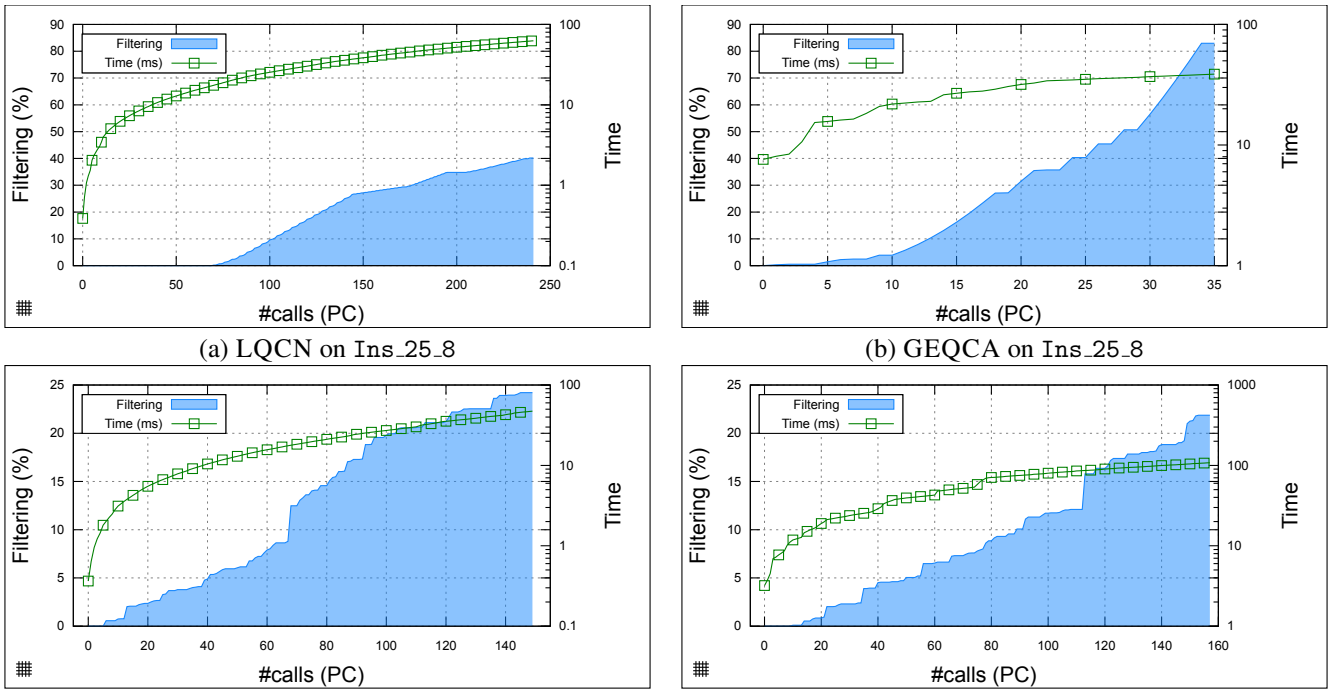
Comparing LQCN to GEQCA, our initial observation is that GEQCA improves on 1) sparse instances (density not exceeding 10%) and 2) large instances (100 TI), which both represent 7 over 12 instances. On these instances, the saved user effort using GEQCA ranges in between 17% and 40%, which amounts for 76,517 less queries than LQCN. On the remaining instances (5 out of 12), LQCN is the winner with a saved user effort in between 3% and 11%, which represents 793 queries less than GEQCA. But our main observation is on instances of 100 TI, where GEQCA outperforms LQCN with a constant saved effort of 35%, which amounts for 22,520 queries less than LQCN per instance. Our analysis shows that the PATH heuristic combined with PC has a great impact on large instances. Even though the instances have been randomly generated, these results are promising for using GEQCA on both sparse real-world instances (few solutions) and large ones (i.e., more than 100 TI).

To strengthen our observations, Figure 2 shows the cumulative filtering (percentage of removed atomic relations) and the cumulative CPU time of PC calls using LQCN and GEQCA, and acting on two instances (Ins_25_8 and Ins_25_58).

For sparse Ins_25_8 (Figure 2.(a) and (b)), we observe that GEQCA, with its heuristic, removes more than 80% of atomic relations using 35 PC calls, where LQCN, with its weight-based heuristic, is only able to remove 40% using 240 PC calls. Another observation is on CPU time (in seconds) needed by both approaches. GEQCA removes the 80% relations in a total time not exceeding 40sec., whereas 40% relations of LQCN are removed in 62sec. On sparse instances and within a given amount of time, GEQCA is able to save twice the user effort with 7 times fewer calls of PC, as compared to LQCN.

For dense Ins_25_58 (Figure 2.(c) and (d)), we observe a similar behavior for both approaches with a slight advantage for LQCN in terms of number of calls (16 calls less), in terms of filtering (2% more) and in terms of cumulative CPU time (50 seconds less).

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




(a) LQCN on Ins_25_8

(b) GEQCA on Ins_25_8

(c) LQCN on Ins_25_58

(d) GEQCA on Ins_25_58

Figure 2: LQCN vs GEQCA: removed relations and time wrt PC calls.

| Instance | $\mathtt{Q}^+ = size(L)$ | LQCN | | GEQCA | |
|---|---|---|---|---|---|
| | | $\mathtt{Q}^-$ | eF | $\mathtt{Q}^-$ | eF |
| Ins_10_8 | 45 | 228 | 47% | **112** | **26%** |
| Ins_10_25 | 145 | **177** | **55%** | 201 | 59% |
| Ins_10_46 | 268 | 189 | 78% | **106** | **63%** |
| Ins_25_8 | 300 | 2,034 | 60% | **364** | **17%** |
| Ins_25_28 | 1,100 | **695** | **46%** | 748 | 47% |
| Ins_25_58 | 2,253 | **703** | **76%** | 794 | 78% |
| Ins_50_8 | 1,225 | 7,308 | 53% | **748** | **12%** |
| Ins_50_48 | 7,680 | **1,500** | **58%** | 1,763 | 59% |
| Ins_50_52 | 8,316 | **1,689** | **63%** | 2,051 | 65% |
| Ins_100_8 | 4,950 | 23,754 | 44% | **1,634** | **10%** |
| Ins_100_32 | 20,395 | 24,276 | 69% | **1,653** | **34%** |
| Ins_100_37 | 23,942 | 24,974 | 76% | **1,629** | **40%** |

Table 1: User effort, eF, of LQCN vs GEQCA.

## [RQ2]: Using GEQCA for Learning the Qualitative Part of Real-world Scheduling Problems

For our second experiment, we evaluate GEQCA in a real-world context by learning (temporal) constraints of the Resource Constrained Project Scheduling Problem (RCPSP) (Hartmann and Briskorn 2010). We use the publicly available RCPSP instances[3] and we consider the structure of the problem with tasks duration, resource requirements, and resource capacities as background knowledge $\mathcal{K}$, noted $K_1$. Note that $K_1$ can be propagated at line 7 of GEQCA. Also, some constraints are already known by the user, like the *cumulative* global constraint and deadline constraint. We call $K_2$ the background knowledge that includes the *cumulative* and deadline constraint. Here, the information in $K_1 \wedge K_2$ can be propagated at line 10 of GEQCA.

In Table 2, we report the user effort on 5 scheduling instances using GEQCA with PATH heuristic, cutoff $= 3,600s$ and $\mathcal{K} \in \{\varnothing, K_1, K_1 \wedge K_2\}$. We also report $\mathrm{T}_{max}$, the maximum waiting time between two queries. A scheduling instance is characterized by the number of tasks (e.g., sch_30_1 denotes instance 1 with 30 TI).

The first observation from Table 2 is that feeding GEQCA with the structure of the problem $K_1$, used in the *propagate* procedure, reduces substantially the effort of the user (38% of reduction on average corresponding to 30,528 queries). The second observation is that the effort is also reduced when background knowledge is fed and especially with known constraints like *cumulative* and deadline constraints. Using $K_1 \wedge K_2$ in the *solve* procedure of GEQCA (in addition to using $K_1$ in the *propagate* procedure) brings a small but not very significant improvement (reduction of 41% instead of 38% in average). In addition, in terms of CPU time, the waiting time between two queries can reach the cutoff of one hour under $\mathcal{K} = K_1 \wedge K_2$. This is explained by the *solve* procedure, which can spend more than one hour trying to prove that a relation is consistent with the network. Note that with $\mathcal{K} = \varnothing$, the waiting time exceeds 7sec. with a PC call. The *propagate* procedure can also increase the waiting time to more than 9sec. under $\mathcal{K} = K_1$. According to (Lallemand and Gronier 2012), a cutoff of 2sec. corresponds

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

| Instance | $\mathcal{K}$ | | | | | |
|---|---|---|---|---|---|---|
| | $\varnothing$ | | $K_1$ | | $K_1 \wedge K_2$ | |
| | eF | $T_{max}$ | eF | $T_{max}$ | eF | $T_{max}$ |
| sch_30_1 | 95% | 0.79 | 55% | 0.91 | 53% | 1.18 |
| sch_30_2 | 98% | 0.62 | 52% | 0.90 | 48% | 393.08 |
| sch_60_1 | 99% | 4.80 | 65% | 8.51 | 62% | 13.00 |
| sch_60_2 | 99% | 7.72 | 64% | 8.08 | 61% | 12.55 |
| sch_60_3 | 98% | 5.94 | 59% | 9.66 | 57% | 3,600 |

Table 2: User effort eF using GEQCA acting on RCPSP instances (with cutoff $= 3,600s$, $T_{max}$ in seconds).

| Instance | $\mathcal{K}$ | | | | | |
|---|---|---|---|---|---|---|
| | $\varnothing$ | | $K_1$ | | $K_1 \wedge K_2$ | |
| | eF | $T_{max}$ | eF | $T_{max}$ | eF | $T_{max}$ |
| sch_30_1 | 95% | 0.79 | 55% | 0.91 | 53% | 1.18 |
| sch_30_2 | 98% | 0.62 | 52% | 0.90 | 49% | 2 |
| sch_60_1 | 99% | 2 | 65% | 2 | 62% | 2 |
| sch_60_2 | 99% | 2 | 64% | 2 | 61% | 2 |
| sch_60_3 | 98% | 2 | 59% | 2 | 58% | 2 |

Table 3: User effort eF using GEQCA acting on RCPSP instances (with cutoff $= 2s$, $T_{max}$ in seconds).

to an acceptable waiting time for a human user. Hence, we conduct a new experiment by setting the cutoff time to 2sec. Table 3 reports on the obtained results.

Like in Table 2, we report the user effort on 5 scheduling instances using GEQCA with PATH heuristic, cutoff $= 2s$ and $\mathcal{K} \in \{\varnothing, K_1, K_1 \wedge K_2\}$. Comparing to Table 2, the user effort is exactly the same with $\mathcal{K} = \varnothing$ but with a waiting time not exceeding the cutoff time. This means that PC removes inconsistent basic relations in less than 2sec. in the first iteration. The same observation is drawn With $\mathcal{K} = K_1$, where propagating the structure of the problem on the learned network is realized in less than 2sec. Using $\mathcal{K} = K_1 \wedge K_2$, we observe a very low decline on two instances (sch_30_2 and sch_60_2). On sch_30_2 for example, GEQCA requires an effort of only $48\%$ but with a waiting time exceeding 6 minutes, whereas using a cutoff of 2sec. the effort is $49\%$.

The RCPSP instances are expressed using *precedes* and *meets* relations. As a last point, we observe that GEQCA learns two times more of *precedes* constraints than the ones present in the benchmarks. Thanks to the PC procedure that reduces an important number of *full* constraints to *precedes*. As a direct consequence, we observe that finding the optimal solution is faster using the learned network.

## Discussion & Limits

To wrap up, the results show the effectiveness of our approach, GEQCA, in learning qualitative constraints. The number of queries asked to the user, stays in low boundaries on sparse and large instances, as compared to LQCN. Indeed, in GEQCA, the PC propagation altogether with heuristic PATH drastically reduces the number of queries on these instances, while it does not bring significant improvements on dense instances, as compared to LQCN. Also, GEQCA provides a cutoff option to minimize the waiting time between two queries and setting this cutoff time to 2sec. is a respectful choice.

It is worth noticing that a large number of queries (e.g., thousands of queries) prevents using GEQCA in an interactive process with a (single) human user in the loop. A concept to learn can also be known by a non-CP representation of the problem. For instance, some expert systems can easily check and respond queries, even though they are incapable of solving problems.[4] Also, the concept to learn can be known by a human community (e.g., thousands of users) sharing the same concept and able to classify a few queries within a crowdsourcing/parallel context (Lazaar 2021). In addition, one can inherit constraints from a past/obsolete model that needs to be updated. GEQCA can easily be adapted to handle such cases with its incremental calculation of $L$.

However, GEQCA can be improved by:

- **Adaptive cutoff selection:** Finding the best trade-of for setting the cutoff value is a real dilemma. A small value yields less waiting time between queries but compromises the capabilities of the propagation (using *propagate*, *solve* or PC) to filter enough. An approach could be to dynamically adapt the cutoff value with the estimated capabilities of the propagation.
- **Filter positive queries:** GEQCA does not provide a mechanism to avoid asking positive queries. Hence, on dense problems, the propagation does not help reducing the effort (most queries are positive). Considering ways to infer/avoid some queries may improve GEQCA.
- **Constraint selection heuristic:** Despite its effectiveness in practice, PATH heuristic is not suitable for all types of problems. Combining PATH with other dynamic entity-selection heuristics can be further investigated.

## Conclusion

In this paper, we propose GEQCA, a new generic and correct active CA algorithm, that learns any qualitative network via qualitative queries. GEQCA exploits the *JEPD* property of qualitative calculus to prove convergence in a polynomial number of queries and path consistency over qualitative constraints to minimize the number of queries. Our results show that GEQCA asks a limited number of queries to the user in a reasonable amount of time, making this approach suitable for practical applications. In addition, GEQCA not only generalizes but also outperforms the existing approach LQCN by using a dedicated constraint selection heuristic. Note also that, unlike LQCN, GEQCA can exploit extended background knowledge to reduce the number of queries. Our future work involves two directions. First, GEQCA can be further refined via smarter heuristics for constraint selection. For example, selecting first the next query to ask on the size of remaining atomic relations between each pair of entities may speed up the acquisition. Second, use GEQCA to learn complex qualitative networks which combine both spatial and temporal constraints.

[4] Some expert systems become *Rube Goldberg machines* after having been updated too many times.

## Acknowledgments

This work has received funding from the Norwegian Research Council T-LARGO project under grant agreement No 274786, the European Union's Horizon 2020 research and innovation programme under grant agreement No 952215, and University of Montpellier I-Site MUSE under CARUM2020/2021 project.

## References

Addi, H. A.; Bessiere, C.; Ezzahir, R.; and Lazaar, N. 2018. Time-bounded query generator for constraint acquisition. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR 2018)*, volume 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.

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

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

Beldiceanu, N.; and 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)*, 141–157. Springer.

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

Belhadji, S.; and 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.; and Walsh, T. 2013. Constraint acquisition via partial queries. In *Twenty-Third International Joint Conference on Artificial Intelligence*.

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

Bessiere, C.; Coletta, R.; O'Sullivan, B.; and Paulin, M. 2007. Query-Driven Constraint Acquisition. In *IJCAI*, volume 7, 50–55.

Bessiere, C.; Koriche, F.; Lazaar, N.; and O'Sullivan, B. 2017. Constraint acquisition. *Artificial Intelligence*, 244: 315–342.

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

Crainic, T. G.; Perboli, G.; and Tadei, R. 2012. *Recent Advances in Multi-dimensional Packing Problems*. ISBN 978-953-51-0480-3.

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

Hartmann, S.; and 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.; and Jonsson, P. 2003. Reasoning about Temporal Relations: The Tractable Subalgebras of Allen's Interval Algebra. *J. ACM*, 50(5): 591–640.

Lallemand, C.; and Gronier, G. 2012. Enhancing User eXperience during waiting time in HCI: contributions of cognitive psychology. In *Proceedings of the Designing Interactive Systems Conference*, 751–760.

Lazaar, N. 2021. Parallel Constraint Acquisition. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI*, 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.; and Scollo, G., eds., *Algebraic Methodology and Software Technology (AMAST'93)*, 27–44. London: Springer London. ISBN 978-1-4471-3227-1.

Mouhoub, M.; Marri, H. A.; and Alanazi, E. 2018. Learning Qualitative Constraint Networks. In Alechina, N.; Nørvåg, K.; and Penczek, W., eds., *25th International Symposium on Temporal Representation and Reasoning, TIME 2018, Warsaw, Poland, October 15-17, 2018*, volume 120 of *LIPIcs*, 19:1–19:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Pawlak, T. P.; and 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.; and Browne, D. 2021. Classifier-based constraint acquisition. *Annals of Mathematics and Artificial Intelligence*, 1573–7470.

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

Tsouros, D. C.; Stergiou, K.; and Bessiere, C. 2019. Structure-Driven Multiple Constraint Acquisition. In *International Conference on Principles and Practice of Constraint Programming*, 709–725. Springer.

Tsouros, D. C.; Stergiou, K.; and Bessiere, C. 2020. Omissions in Constraint Acquisition. In Simonis, H., ed., *Principles and Practice of Constraint Programming*, volume 12333 of *Lecture Notes in Computer Science*, 935–951. Springer.

van Beek, P.; and Manchak, D. W. 1996. The Design and Experimental Analysis of Algorithms for Temporal Reasoning. *Journal of Artificial Intelligence Research*, 4.

Vilain, M.; and Kautz, H. 1986a. Constraint Propagation Algorithms for Temporal Reasoning. In *Proceedings of the Fifth AAAI National Conference on Artificial Intelligence*, AAAI'86, 377–382.

Vilain, M. B.; and Kautz, H. A. 1986b. Constraint propagation algorithms for temporal reasoning. In *Aaai*, volume 86, 377–382.