Learning to Resolve Inconsistencies in Qualitative Constraint Networks

Anastasia Paparrizou[a,*], Michael Sioutis[a,*]

[a]*LIRMM UMR 5506, University of Montpellier, CNRS, France*

Abstract

In this paper, we present a reinforcement learning approach for resolving inconsistencies in qualitative constraint networks (QCNs). QCNs are typically used in constraint programming to represent and reason about intuitive spatial or temporal relations like x {*is inside of* $\vee$ *overlaps*} y. Naturally, QCNs are not immune to uncertainty, noise, or imperfect data that may be present in information, and thus, more often than not, they are hampered by inconsistencies. We propose a multi-armed bandit approach that defines a well-suited ordering of constraints for finding a maximal satisfiable subset of them. Specifically, our learning approach interacts with a solver, and after each trial a reward is returned to measure the performance of the selected action (constraint addition). The reward function is based on the reduction of the solution space of a consistent reconstruction of the input QCN. Experimental results with different bandit policies and various rewards that are obtained by our algorithm suggest that we can do better than the state of the art in terms of both effectiveness, viz., lower number of repairs obtained for an inconsistent QCN, and efficiency, viz., faster runtime.

Keywords: Spatial and Temporal Reasoning, Qualitative Constraints, Imperfect Data, Inconsistency Resolution, Maximizing Satisfiability, Reinforcement Learning, Multi-armed Bandit.

1. Introduction

Representing and reasoning about spatial or temporal information in a natural, human-like manner, is the area of study of Qualitative Spatio-Temporal Reasoning (QSTR), a rich symbolic AI framework spanning various fields, such as constraint programming, logic, and mathematics [1, 2]. As an example, within

*Corresponding author

Email addresses: anastasia.paparrizou@lirmm.fr (Anastasia Paparrizou), msioutis@lirmm.fr (Michael Sioutis)

URL: https://orcid.org/0000-0002-6440-0455 (Anastasia Paparrizou), https://msioutis.gitlab.io/ (Michael Sioutis), https://orcid.org/0000-0001-7562-2443 (Michael Sioutis)

QSTR, we can consider a relation like x $\{$*is inside of* $\vee$ *overlaps*$\}$ y, which does not involve any quantitative information and is rather intuitive. Such relations, and combinations thereof, can be modeled as a qualitative constraint network (QCN), a simplified example of which is provided in Figure 1. Quite naturally, representing spatial or temporal information even in that symbolic, intuitive way, is not exempt from the presence of inconsistencies, as, more often than not, the information is infused with uncertainty, noise, and imperfect data among other things; we tackle this issue in this paper.

Context & Motivation

We focus on the MAX-QCN problem [3], which is the problem of maximizing satisfiability in a QCN; see Figure 1. Concretely, given a QCN $\mathcal{N}$, solving the MAX-QCN problem of $\mathcal{N}$ is obtaining a deviating configuration that maximizes the number of satisfied constraints in $\mathcal{N}$. As mentioned earlier, representing spatial or temporal information in QSTR may inevitably lead to inconsistencies; this can be, for example, due to human error, noise in the data we abstract from, and/or inaccurate classifiers.

As illustration, in Business Process Management we often meet scheduling inconsistencies, due to the unavailability of resources for certain tasks [4]. In particular, automotive industry [5], as an instance of scheduling, involves assigning temporal intervals to tasks that are subject to limited resources. For example, in the context of car manufacturing, where the problem involves scheduling cars -with a high number of options combinations-along an assembly line with capacity constraints [6], an inconsistency may appear when two (or more) cars are allocated to the same assembly line in overlapping temporal intervals. To ensure safety of operations and minimisation of delayed deliveries to customers, the inconsistency must then be repaired by considering different assembly lines for the same temporal intervals, while minimizing changes so as to perturb the timetable to the least extent possible. Resolving inconsistencies in QSTR extends also to the broader context of neuro-symbolic AI architectures dealing with textual entailment or image analysis, among other tasks [7]; indeed, resolving/minimizing inconsistency (equivalently, maximizing satisfiability) can be seen as an important step of logical reasoning in the neuro-symbolic cycle, see, e.g., Figure 1 in [8], where minimizing inconsistency in a knowledge base forms the basis of logical abduction in abductive learning.

Related Work & Contribution

Solving (optimally) the MAX-QCN problem, or coming close to a solution of it, has been dealt with in various works in the literature, e.g., [3, 9, 10]. Of these works, [10] is the state of the art wrap-up of optimal encodings and (sub-)optimal heuristics for tackling this problem. Specifically, with respect to heuristics for approaching a solution of the MAX-QCN problem, [10] proposes a portfolio-style approach that combines several diverse procedures, each of which probes the constraints of a QCN in linear fashion, following its own ordering of constraints, and filters out the ones that fail the satisfiability check. Notably,

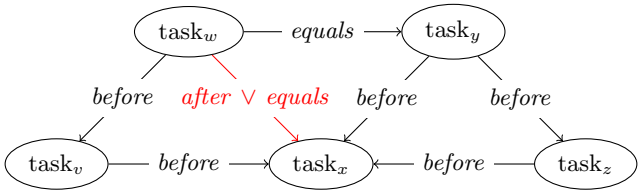


Figure 1: An illustration of the MAX-QCN problem for a qualitative constraint network (QCN) [3] and the terminology used here; the simplified QCN, which can be viewed as a temporal plan, is inconsistent, and one solution of the MAX-QCN problem, viz., an *optimal* scenario, can be obtained by placing task$_w$ {*before*} task$_x$ and hence repairing the respective constraint in the figure; note that, had we chosen to keep task$_w$ {*after* ∨ *equals*} task$_x$, we would have had to make two repairs.

each such procedure taken alone implements a simple instance of basic linear search for computing a maximal satisfiable subset of constraints, cf. [11, 12], yet taken all together, hence considering different orderings of constraints, can produce a powerful approximation of the MAX-QCN problem. To a certain extent, finding a good initial ordering of constraints in this context is similar to starting from a configuration that violates as few constraints as possible, cf. [13], in the sense that we would ideally want to have all the constraints of a maxim**um** (cardinality) subset of constraints first in our ordering (as this would give the optimal solution). As illustration, let us revisit Figure 1 and consider the case where task$_w$ {*after* ∨ *equals*} task$_x$ is the first constraint to be checked for inclusion in a (initially empty) satisfiable subset; clearly, it will be included, but it will result in a deviation from an optimal solution.

In the context of SAT/CP solving, several approaches have been proposed for the efficient determination of relaxations/explanations, referred to as Minimal Correction Subsets [14, 15, 16]. In the context of interactive configuration problems, the quality of a diagnosis algorithm is a major issue, viz., the explanations to be identified need to be relevant for the user. Therefore, the authors of [17] and [18] introduce divide-and-conquer-based approaches that identify minimal sets of faulty constraints and are suitable when preferred explanations/diagnoses are necessary. The work of [19] extends the direct diagnosis algorithm of [18] to anytime diagnosis by including a parameter that systematically limits the number of consistency checks in order to make diagnosis search more efficient. This idea of trying to learn a new better ordering of constraints by exploiting (existing) constraint ordering heuristics can be seen as similar to the works of [20, 21, 22, 23] in the context of configuration systems. The latter approaches are based on supervised learning techniques (i.e., matrix factorization) applied on historical (inconsistent) transactions in order to build constraint ordering heuristics offline. They then solve new diagnosis tasks (online) applying the learned heuristics to reorder user constraints via a direct diagnosis algorithm. However, even though both a configuration task and a QCN can be modeled as a CSP (regardless of whether it is not the most efficient representation for a QCN, see [24]), their modelings are entirely different, as they are based on entirely dif-

ferent knowledge bases. Therefore, it is not evident how these representations can leverage techniques from one another in a meaningful way when the context is different (though this could be a topic of future research). For example, contrary to the diagnosis-related approaches, in our context we have no user requirements, and/or preferred constraints over others, thus we do not see how splitting the constraint set in any way (as is typically done in those works) could be beneficial. From a technical point of view, our work differentiates from the aforementioned ones, in that we propose an entirely online learning approach versus a supervised one that learns which constraint/relation ordering heuristic is able to produce a minimal number of repairs in a given over-constrained problem, exploiting the feedback returned by the solving process.

In this paper, with respect to the previous discussion, we leverage machine learning to improve inconsistency resolution in QSTR and make the following contributions:

(i) We propose a multi-armed bandit approach [25] to synthesize different orderings of constraints, with the aim to produce a new one that will outperform each of the former ones in the number of repairs performed for an inconsistent QCN (the lower, the better); to this end, we build upon the approach of [10] and we infuse it with a reinforcement learning aspect, where every action of processing the next constraint among different orderings of constraints is rewarded based on how much it reduces the solution space.

(ii) We experimentally compare an implementation of our multi-armed bandit approach to the implementation of [10] using a standard dataset of QCNs of Interval Algebra [26], and observe several pertinent properties of our method, the most important ones being that it is both more effective (fewer number of repairs) and more efficient (faster runtime) than the one of [10]; in addition, we employ an optimal Partial MaxSAT solver to show/reaffirm that solving (optimally) the MAX-QCN problem is, generally, impractical.

Organization. The rest of the paper is organized as follows. In Section 2 we provide definitions/notations regarding QSTR and the MAX-QCN problem that are necessary for following the paper. Then, in Section 3 we detail our reinforcement learning approach for tackling the MAX-QCN problem, and in Section 4 we experimentally evaluate it against the state of the art. Finally, we conclude with some discussion for future work in Section 5.

2. Preliminaries

A binary qualitative spatial or temporal constraint language is based on a finite set B of *jointly exhaustive and pairwise disjoint* relations, called *base relations* [1] and defined over an infinite domain D (e.g., $\mathbb{R}$). The base relations of a particular qualitative constraint language can be used to represent the definite knowledge between any two of its entities with respect to the level of

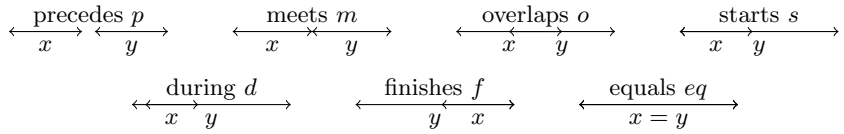


Figure 2: A representation of the 13 base relations b of IA, each one relating two potential intervals x and y as in $x\ b\ y$; the converse of b, i.e., b^{-1}, can be denoted by bi and is omitted in the figure.

granularity provided by the domain D (e.g., for $\mathsf{D} = \mathbb{R}$, we could have $\mathsf{B} = \{<, =, >\}$, with the usual semantics). The set B contains the identity relation Id, and is closed under the *converse* operation $(^{-1})$. Indefinite knowledge can be specified by a union of possible base relations, and is represented by the set containing them.

As illustration, consider the well-known qualitative temporal constraint language of Interval Algebra (IA) [26]. IA considers time intervals on the real line, and the set of base relations $\mathsf{B} = \{eq\ (= \mathsf{Id}),\ p,\ pi,\ m,\ mi,\ o,\ oi,\ s,\ si,\ d,\ di,\ f,\ fi\}$ to encode knowledge about the temporal relations between such intervals, as described in Figure 2.

Representing and reasoning about qualitative spatio-temporal information pertaining to a set of base relations B can be facilitated by a *qualitative constraint network* (QCN):

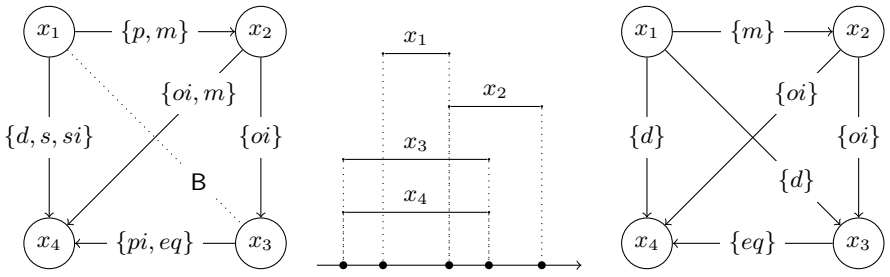
Definition 1. A *qualitative constraint network (*QCN*)* is a tuple (V, C) where:

- $V = \{v_1, \ldots, v_n\}$ is a non-empty finite set of variables (representing entities in D);
- and C is a mapping $C : V \times V \to 2^{\mathsf{B}}$ such that, $\forall v \in V$, $C(v, v) = \{\mathsf{Id}\}$, and, $\forall v, v' \in V$, $C(v, v') = (C(v', v))^{-1}$.

An example QCN of IA is shown in Figure 3a; for conciseness, converse relations or Id loops are not shown in the figure.

Definition 2. Let $\mathcal{N} = (V, C)$ be a QCN, then:

- a *solution* of $\mathcal{N}$ is a mapping $\sigma : V \to \mathsf{D}$ such that, $\forall (u, v) \in V \times V$, $\exists b \in C(u, v)$ such that $(\sigma(u), \sigma(v)) \in b$; and $\mathcal{N}$ is *satisfiable* iff it admits a solution (see Figure 3b);
- a *sub-*QCN (also known as *refinement*) $\mathcal{N}'$ of $\mathcal{N}$, denoted by $\mathcal{N}' \subseteq \mathcal{N}$, is a QCN (V, C') such that, $\forall u, v \in V$, $C'(u, v) \subseteq C(u, v)$;
- $\mathcal{N}$ is *atomic* iff, $\forall v, v' \in V$, $C(v, v')$ is a *singleton relation*, i.e., a relation $\{b\}$ with $b \in \mathsf{B}$;
- a *scenario* $\mathcal{S}$ of $\mathcal{N}$ is an atomic satisfiable sub-QCN of $\mathcal{N}$ (see Figure 3c);



(a) A satisfiable QCN $\mathcal{N}$. (b) A solution σ of $\mathcal{N}$. (c) A scenario $\mathcal{S}$ of $\mathcal{N}$.

Figure 3: Figurative examples of QCN terminology using IA.

- the *constraint graph* of $\mathcal{N}$, denoted by $\mathsf{G}(\mathcal{N})$, is the graph (V, E) where $\{u, v\} \in E$ iff $C(u, v) \neq \mathsf{B}$ and $u \neq v$;
- $\mathcal{N}$ is denoted by $\mathcal{N}_\top$ when each of its constraints is *universal*, i.e., iff, $\forall v, v' \in V$ with $v \neq v'$, $C(v, v') = \mathsf{B}$.

The MAX-QCN *problem*

The MAX-QCN problem has been introduced in the context of QSTR in [3]. Given a QCN $\mathcal{N}$ over a set of variables V, the MAX-QCN problem is the problem of finding a scenario over V that maximizes the number of satisfied constraints in $\mathcal{N}$, or, equivalently, the problem of finding a scenario over V that minimizes the number of unsatisfied constraints in $\mathcal{N}$. Such scenarios are called *optimal* scenarios of $\mathcal{N}$. Clearly, if a QCN $\mathcal{N}$ is satisfiable, any scenario of $\mathcal{N}$ is also an optimal scenario of $\mathcal{N}$. The reader is kindly asked to revisit Figure 1 in the introduction for a simplified example of the MAX-QCN problem and a solution of it. Solving the MAX-QCN problem is clearly at least as difficult as solving the satisfiability checking problem of a QCN, which is NP-hard in general for most calculi [2].

An optimal scenario, as defined above, is directly associated with a set of *repairs* for the original (unsatisfiable) QCN, i.e., a set of changes/interventions involving constraints of the original QCN so that is becomes satisfiable (indeed, as some constraints will be violated, some others need to take their place); clearly, if the original QCN is already satisfiable, then no repairs are necessary. Formally, we define the term of *repair* as follows:

Definition 3 (Repair). Given a QCN $\mathcal{N} = (V, C)$, a *repair* is defined to be an assignment $C(u, v) \leftarrow b$, where b is a base relation in the complement of $C(u, v)$, i.e., $b \in \mathsf{B} \setminus C(u, v)$, for some $u, v \in C(u, v)$ such that $C(u, v) \neq \mathsf{B}$.

Intuitively, a repair is replacing a base relation of a constraint with another base relation in the complement of that constraint. We can note that repairs will naturally occur when substituting the universal relation B for a problematic

constraint in a given QCN, i.e., a constraint that is part of a minimal correction subset of constraints in that QCN (the dual notion to a maximal satisfiable subset). For example, in Figure 1, the constraint task_w $\{\textit{after} \vee \textit{equals}\}$ task_x forms a minimal correction subset in and of itself; thus, replacing that constraint with the universal relation B will result in instatiating the constraint with a base relation in the complement of $\{\textit{after} \vee \textit{equals}\}$, which, in our example, is in fact the base relation *before*. Based on this observation, in the discussion that follows, and in our implementations, we simply focus on removing constraints from a QCN, i.e., replacing them with the universal relation B, as a means to generate the necessary repairs to make that QCN satisfiable.

3. A Multi-armed Bandit Approach

Our overall approach is presented in Algorithm 1, called POLYPUS; in what follows, we provide a description of this algorithm, focusing on its major components and analyzing its complexity.

The task of obtaining an appropriate ordering of constraints, as detailed in Section 1, can be encoded in a *multi-armed* bandit problem: given a QCN $\mathcal{N}$ and a set of arms $\alpha_1, \cdots, \alpha_k$, the bandit problem consists in assessing the performance of the arms after a sequence of trials $1, \ldots, T$. An arm α_i corresponds to a given ordering of constraints defined under a certain criterion or heuristic (details on arms follow later on), and T is the number of constraints.

More formally, a *multi-armed* bandit problem is a sequential decision process where the learner interacts with its environment. During each trial t, the algorithm selects an arm α_i with $i \in \{1, \cdots, k\}$ and receives a reward $reward_t(\alpha_i)$ for this arm. The goal is to minimize the expected regret across trials, which is defined as the expectation of the difference between the total reward obtained by the best arm and the total reward obtained by the bandit algorithm. The algorithm observes the reward for the chosen arm after each trial, but not for the other arms that could have been selected. Therefore, the minimization of regret is achieved by balancing *exploration* (acquiring new information) and *exploitation* (using acquired information). The *exploration/exploitation* ratio is defined by policy UCB1 [27] or MOSS [28], that among other policies displayed a good performance in several works related to constraint programming [29, 30, 31, 32, 33, 34].

UCB1 is the simplest policy in the Upper Confidence Bound family [27]. It has an expected logarithmic growth of regret uniformly over the number of runs (i.e., counter+1) without any prior knowledge regarding the reward distributions. The Minimax Optimal Strategy in the Stochastic case (MOSS) [28] policy is an instance of the UCB family, that compared to UCB1, it doesn't take into account only the number of trials of individual arms, but also the number of arms (4) and the number of runs. The ultimate goal is to find a strategy for mapping each trial to a probability distribution over $\{\alpha_1, \cdots, \alpha_k\}$ in order to maximize cumulative rewards.

A trial t in our case is defined as the period where an arm is pulled until a solver—here, a native qualitative constraint-based solver was chosen—returns

Algorithm 1: POLYPUS($\mathcal{N}, \mathcal{A}, e, *policy$)

input : A QCN $\mathcal{N} = (V, C)$, a set $\mathcal{A}$ of bijections $\alpha : E \to \{0, 1, \ldots, |E| - 1\}$, where $E = E(\mathsf{G}(\mathcal{N}))$ (i.e., roughly, a set of orderings of the constraints in $\mathcal{N}$), a number of epochs e with $e \in \mathbb{N}^+$, and a function pointer $*policy$ that will point to the selected policy

output : A subset $p \subseteq E(\mathsf{G}(\mathcal{N}))$ corresponding to a maximal satisfiable subset of constraints in $\mathcal{N}$

1 $P \leftarrow \emptyset$;
2 $armPulls \leftarrow map()$;
3 $meanReward \leftarrow map()$;
4 $counter \leftarrow 0$;
5 **foreach** $\alpha \in \mathcal{A}$ **do**
6 $\quad armPulls[\alpha] \leftarrow 0$;
7 $\quad meanReward[\alpha] \leftarrow 0$;
8 **for** n **from** 1 **to** e **do**
9 $\quad p \leftarrow \emptyset$;
10 $\quad \mathcal{N}' = (V, C') \leftarrow \mathcal{N}_\top$;
11 $\quad unprocessedConstraints \leftarrow E(\mathsf{G}(\mathcal{N}))$; `// each constraint initiates a trial`
12 $\quad$ **while** $unprocessedConstraints \neq \emptyset$ **do**
13 $\quad\quad policy \leftarrow \&function$; `// Policy function is selected`
14 $\quad\quad \alpha \leftarrow *policy(\alpha, meanReward, counter, \mathcal{A})$;
15 $\quad\quad c \leftarrow \min(\{c' \in \{0, 1, \ldots, |E(\mathsf{G}(\mathcal{N}))| - 1\} \mid \alpha^{-1}(c') \in unprocessedConstraints\})$;
16 $\quad\quad \{u, v\} \leftarrow \alpha^{-1}(c)$;
17 $\quad\quad C'(u, v) \leftarrow C(u, v)$; $C'(v, u) \leftarrow C(v, u)$;
18 $\quad\quad (result, reward) \leftarrow SAT(\mathcal{N}')$; `// see discussion about rewards in the text`
19 $\quad\quad$ **if** $result = true$ **then**
20 $\quad\quad\quad p \leftarrow p \cup \{\{u, v\}\}$;
21 $\quad\quad$ **else**
22 $\quad\quad\quad C'(u, v) \leftarrow \mathsf{B}$; $C'(v, u) \leftarrow \mathsf{B}$;
23 $\quad\quad meanReward[\alpha] \leftarrow \dfrac{(meanReward[\alpha] \cdot armPulls[\alpha]) + reward}{(armPulls[\alpha] + 1)}$;
24 $\quad\quad armPulls[\alpha] \leftarrow armPulls[\alpha] + 1$;
25 $\quad\quad unprocessedConstraints \leftarrow unprocessedConstraints \setminus \{\{u, v\}\}$;
26 $\quad\quad counter \leftarrow counter + 1$;
27 $\quad P \leftarrow P \cup \{p\}$;
28 **return** $p \in \arg\max_{p' \in P}(|p'|)$; `// largest maximal satisfiable subset is returned`

its output (lines 13–18 in Algorithm 1). During each trial t, a constraint is added to the network (initially empty) according to the order defined by the i arm selected at run t (line 17). If the added constraint provokes unsatisfiability, then the constraint is removed from the network (line 22) and we proceed to the next constraint. The ordering of constraints that results in the largest satisfiable subset of constraints is the best one (line 28). In the following, we analyze the

Function UCB1($armPulls$, $meanReward$, $counter$, $\mathcal{A}$)

$\alpha \leftarrow \alpha \in \arg\max_{\alpha' \in \mathcal{A}} \left(meanReward[\alpha'] + \sqrt{\frac{2 \cdot \ln(counter + 1)}{armPulls[\alpha'] + 1}} \right)$

return α

Function MOSS($armPulls$, $meanReward$, $counter$, $\mathcal{A}$)

$x \leftarrow \max\{1, \frac{counter + 1}{2 \cdot armPulls[\alpha'] + 1}\}$;

$\alpha \leftarrow \alpha \in \arg\max_{\alpha' \in \mathcal{A}} \left(meanReward[\alpha'] + \sqrt{\frac{4}{armPulls[\alpha'] + 1} \cdot \ln(x)} \right)$

return α

set of arms we deploy and we assess the performance of the chosen arm via a reward function.

Remark 1. The literature on *multi-armed* bandits contains several versions for balancing the exploitation-exploration ratio depending on the context, where additional parameters allow to insist more on exploitation or exploration. Our goal is to assess the simplicity of the learning aspect (especially in terms of complexity), thus, we use the most basic regret policies defined by UCB1 and MOSS, where no parameter tuning is required (as opposed to e-greedy [35] for example) and efficiency/convergence is manifested in other constraint problems [29, 30, 31, 32, 33, 34].

Arms

Given a QCN $\mathcal{N}$, an *arm* in Algorithm 1 is an ordering (permutation) of the (non-universal) constraints of $\mathcal{N}$, and is represented by a bijection $\alpha : E \rightarrow \{0, 1, \ldots, |E| - 1\}$, where $E = E(\mathsf{G}(\mathcal{N}))$. Thus, all the different arms form a set $\mathcal{A}$ of such bijections, which is provided as additional input to the algorithm. Each arm (i.e., ordering of constraints) is defined based on one of the following constraint ordering strategies, whose consecutive application constructs the ordering (a more detailed explanation of these strategies is provided in [36]):

- **max**: choose the constraint that contains the base relation with the most *local models* [36].[1]
- **avg**: choose the constraint with the highest average count of local models (i.e., each of its base relations contributes a count and we take the average of these counts).

[1] In sum, a *local model* of a constraint is a solution of the sub-network that solely involves the two variables of the constraint and a third neighboring variable; hence, the total number of local models of a constraint can be seen as an indicator of how much that constraint is supported (or tolerated) in the original network.

- **sum**: choose the constraint with the highest cumulative count of local models. (i.e., each of its base relations contributes a count and we take the sum of these counts).

- **weight**: choose the constraint with the largest weight, where the larger the weight, the more permissive the constraint is; see, e.g., Figure 9 in [37].[2]

- **card + weight**: choose the constraint whose smallest decomposition into sub-relations of a (maximal) tractable subset $\mathcal{S} \in 2^{\mathsf{B}}$ [38] (e.g., the ORD-Horn set for IA [39]) is the largest one; use weight as a tie-breaker (this is typical in the literature, e.g., [38]).

The above strategies have been found to be the best-performing ones in the approach of [10], called GREEDUS, in that they help to produce a lower number of repairs for a given inconsistent QCN compared to other strategies.

Remark 2. GREEDUS [10] acts as a virtual best solver at the QCN level, in the sense that it tries each arm separately, for all the constraints of a given QCN collectively, and then records the best result among the arms. On the other hand, POLYPUS aims to combine different arms for a same given QCN, in an effort to operate as a virtual best solver at the constraint level of the QCN; obtaining the actual virtual best solver in this context is infeasible, as for m constraints we would need to explore $m!$ orderings to find the best one (i.e., the best arm).

Rewards

A state-of-the-art solver for checking the satisfiability of a given QCN defined on a set of base relations B, be it SAT- or qualitative constraint-based, will typically return a locally consistent sub-QCN (refinement) of that QCN over a (maximal) tractable subset $\mathcal{S} \in 2^{\mathsf{B}}$ if the QCN is satisfiable [40, 10]; a scenario (qualitative solution) of the QCN can then be extracted in a backtrack-free manner. Such subsets exist for most well-known qualitative constraint languages [2], and in what follows we assume the use of such a solver; this is the solver that is being used in line 18 of POLYPUS.

A not straightforward and critical part in multi-armed bandits is to define an appropriate reward function that reflects, at each trial, the performance of the arm pulled, giving us a numeric feedback. Given a QCN $\mathcal{N}$, a set of arms $\mathcal{A}$, and a solver as described earlier, let $\mathcal{N}'$ be the QCN that is returned by the solver when given $\mathcal{N}$ as input; in what follows, we present some rewards that we used in our approach.

[2]Permissiveness, or, equivalently, restrictiveness, is measured for each base relation by successively composing the base relation with every possible relation, and then summing up the cardinalities of the resulting compositions; the result is then suitably scaled and a weight is assigned to each base relation, and the weight of a constraint is then given by the sum of the weights of its base relations.

0/1 values

In general, the theoretically most relevant reward is the one that optimizes directly the objective function, i.e., in our context, the one that maximizes the number of the constraints such that the resulting network is satisfiable. Indeed, the first reward that we considered in our experimental evaluation (to follow) was the one that is promoting the arm that is generating the largest maximal satisfiable set of constraints among other arms. Specifically, when we are using this reward, we are returning the value 1 as reward to the arm if the solver is producing a satisfiable instance (each time a constraint is added at line 18), and 0 otherwise. Formally, we define the reward for a pulled arm $\alpha \in \mathcal{A}$ is as follows:

$$reward(\alpha) = \begin{cases} 1, & \text{if } \mathcal{N}' \text{ is satisfiable} \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

So the arm that is producing the largest wrt to the number of constraints satisfiable network has also the biggest mean reward. However, as we will show later on, in practice this reward did not work well in our evaluation, as 0/1 values did not allow for a good discrimination among the arms.

Size of QCN

Building upon the aforementioned reward based on 0/1 values, we consider functions with finer granularity, capturing the solutions space reduction, the constraint weighting inspired from [13, 32, 41]. Let $|\mathcal{N}|$ denote the *size* of a QCN $\mathcal{N} = (V, C)$, i.e., its total number of base relations; formally, $|\mathcal{N}| = \sum_{u,v \in V} |C(u, v)|$. Then, the reward for a pulled arm $\alpha \in \mathcal{A}$ is as follows:

$$reward(\alpha) = \begin{cases} 1 - \left(\dfrac{|\mathcal{N}| - |\mathcal{N}'|}{|\mathcal{N}|} \right), & \text{if } \mathcal{N} \text{ is satisfiable} \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

As we will show in our evaluation later on, this particular reward performed well, since, as was also observed in the literature, the size of a QCN has a fairly strong positive relationship with the number of scenarios (qualittaive solutions) that the QCN contains, specifically, the Spearman's rank correlation coefficient between the two variables is ≈ 0.8 [41]. Intuitively, the more scenarios a QCN contains, the "more" satisfiable it is, and as the size of a QCN is very easy to compute (compared to counting an exponential number of scenarios), we found it to be a pertinent and useful value for our approach.

Constrainedness of QCN

Finally, we used a heuristic approach to estimate the constrainedness of a QCN. This approach is based on the weight strategy that we discussed in the section about arms. As a reminder, this strategy weighs a constraint according to its permissiveness, i.e., the larger the weight, the more permissive the constraint is; see, e.g., Figure 9 in [37].[2] For example, in the case of Interval Algebra, a weight for each of its base relations can be determined by successively

composing the base relation with every possible relation, and then summing up the cardinalities of the resulting compositions. Then, the weight for a base relation is finalized by suitably scaling the obtained result. Next, the weight of a constraint (a set of base relations, essentially) is given by the sum of the weights of its base relations, e.g., the weight of the relation $\{p, o, f\}$ is $3 + 4 + 2 = 9$ [37]. Again, the larger the weight, the more permissive the constraint is, and, equivalently, the smaller the weight of a constraint, the more restrictive the constraint is. Let $w(\mathcal{N})$ denote the *weight (w)* of a QCN $\mathcal{N} = (V, C)$, i.e., the sum of the weights of each of its constraints; formally, $w(\mathcal{N}) = \sum_{u,v \in V}$ weight of $C(u, v)$ Then, the reward for a pulled arm $\alpha \in \mathcal{A}$ is as follows:

$$reward(\alpha) = \begin{cases} 1 - \left(\dfrac{w(\mathcal{N}) - w(\mathcal{N}')}{w(\mathcal{N})} \right), & \text{if } \mathcal{N} \text{ is satisfiable} \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

Intuitively, if the arm that was pulled (that resulted in a constraint to be added) largely maintained the weight of the original QCN, it would suggest that its overall constrainedness remained roughly the same; on the other hand, if the weight of the QCN was reduced significantly, then it should have become much more restrictive.

Computational Complexity

Given a QCN $\mathcal{N}$, the total time complexity of Algorithm 1 is $\Theta(|E(\mathsf{G}(\mathcal{N}))| \cdot \beta)$, where β is the time complexity of the solver that performs the satisfiability check in line 18 of the algorithm; in the case where a qualitative constraint-based solver is used, as is the case in this paper, $\beta = O(|\mathsf{B}|^{|E(\mathsf{G}(\mathcal{N}))|})$, however, in practice, a satisfiability check can be typically performed very fast. Specifically, with regard to the total time complexity, taking into account that the number of epochs e is a small constant, it can be seen that $\Theta(|E(\mathsf{G}(\mathcal{N}))|$ calls are performed to such a satisfiability checking solver (lines 11–18), and that all other operations take constant time. Thus, a total time complexity of $\Theta(|E(\mathsf{G}(\mathcal{N}))| \cdot \beta)$ is achieved.

4. Experimentation

In this section, with respect to resolving inconsistencies in QCNs, i.e., addressing the $\mathsf{MAX\text{-}QCN}$ problem, we compare an in-house implementation of POLYPUS introduced in Section 3 (Algorithm 1) to the greedy approach from [10], called GREEDUS. Specifically, we consider the following methods:

- GREEDUS from [10];
- $\text{POLYPUS}^{\text{P}}_{\text{R}}$ (Algorithm 1), where P is one of the policies among UCB1 and MOSS and R one of the rewards among 1/0 (Equation 1), SIZE (Equation 2), and W (Equation 3);
- and RANDOM, which corresponds to a uniformly random arm selection (see lines 13–14 in Algorithm 1).

Interestingly, and in relation to Remark 2, GREEDUS can be encoded by POLYPUS if an arm is consecutively pulled for all constraints in a QCN, and then the best result among all arms (the result is defined as the number of repairs, where the best is lowest) is recorded for that QCN. We also use the RC2 MaxSAT solver [42] via the PySAT toolkit [43] to compute some ground truth, i.e., optimal number of repairs for our QCNs, to the extent possible; we use the MaxSAT encoding of [10] for this purpose, which is based on Horn theories of the used qualitative calculi (when such theories exist).

To the best of our knowledge, and as far as approaches that are tailored to QCNs are concerned, our selected baseline(s) from [10] (note that the MaxSAT encoding of [10] for QCNs is also a novelty) constitute the state-of-the-art for tackling the MAX-QCN problem.

In our evaluation campaign, we investigate the following hypotheses about the need for a more systematic learning approach than a naive greedy one.

H1 Does synthesizing different strategies for a same instance gives us a better result than choosing the best strategy (standalone) for that instance?

H2 Does this learning approach pay off in terms of runtime and constraints added to the network or does the overhead of learning outweigh its benefits?

In what follows, we show that we can give a positive answer to both of the aforementioned questions.

Note 1. All code, dataset, & logs are available at https://doi.org/10.5281/zenodo.11032284.

Dataset & Setup

We considered Interval Algebra (IA) network instances generated by the standard $\mathsf{A}(n, d, l)$ model [38], used extensively in the literature. In short, $\mathsf{A}(n, d, l)$ creates network instances of size n, average constraint graph degree d, and an average number l of base relations per constraint. We set $n = 30$ and $l = 6.5$, and we generated 100 network instances for *each* degree d between 11 and 15 with a 1-degree step; hence, 500 network instances in total. Typically, the larger the average degree d, the harder the MAX-QCN problem is to solve [3, 9] (see also the number of timeouts of RC2 in Figure 4c for the denser instances). For the experiments we used an Intel® Core™ CPU i7-12700H @ 4.70GHz, 16 GB of RAM, and the Ubuntu Linux 22.04 LTS OS, and one CPU core per network. All coding/running for our implementations was done in Python 3 and sped up with PyPy (https://www.pypy.org/).

Results & Remarks

Before going into the details of the different arms/policies and rewards used, we present in Table 1 the main findings among the better performing configurations, along with RANDOM as a point of reference. We can observe that GREEDUS

	$d = 11$	12	13	14	15	20
GREEDUS	**3.7** **1.75s**	**7.8** **3.06s**	**12.39** **4.30s**	**17.88** **5.04s**	**22.99** **6.81s**	**52.37** 10.87s
POLYPUS$^{\text{UCB1}}_{\text{W}}$ ($e = 7$)	3.83 1.91s	7.81 3.74s	12.52 4.73s	17.57 5.91s	**21.97** **5.93s**	52.03 11.67s
POLYPUS$^{\text{UCB1}}_{\text{SIZE}}$ ($e = 7$)	3.9 1.98s	7.9 **3.74s**	12.53 **4.94s**	17.56 **6.04s**	22.23 6.08s	**51.46** **12.67s**
POLYPUS$^{\text{MOSS}}_{\text{W}}$ ($e = 7$)	**4.21** **1.99s**	**8.22** 3.74s	**12.9** 4.63s	**17.5** 5.47s	22.78 5.97s	51.67 **8.56s**
POLYPUS$^{\text{MOSS}}_{\text{SIZE}}$ ($e = 7$)	4.15 1.96s	8.11 3.56s	12.56 4.75s	17.56 5.22s	22.31 5.95s	52.15 9.14s
RANDOM ($e = 7$)	3.94 2.03s	8.03 3.67s	12.76 **4.26s**	18.35 5.90s	23.27 **5.77s**	52.9 **7.65s**

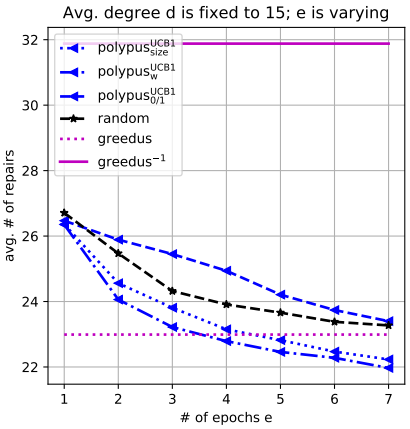
Table 1: **Synopsis**: Assessing the performance of an implementation of POLYPUS and GREEDUS [10], respectively, with Interval Algebra (IA) network instances of model A($n = 30, d, l = 6.5$) [38]; epochs e was set to 7 for POLYPUS, and is irrelevant to GREEDUS (i.e., in this case e can be considered as set to 1), RANDOM corresponds to a uniformly random arm selection (see lines 13–14 in Algorithm 1).

performs best for the less constrained instances, viz., the ones with an average degree $d \leq 13$, but quickly deteriotates to become the worst-performing approach for the more constrained instances, viz., the ones with an average degree $d > 13$. RANDOM is sometimes the fastest approach, denoted by a bold font, as it does not have to perform a selection among arms and/or calculate a reward, but it consistently performs worst with respect to the number of constraints that need to be repaired in an instance in order to recover satisfiability. Among different arm and reward configurations, all of them have their own strong points, with the exception of POLYPUS$^{\text{MOSS}}_{\text{SIZE}}$, which, though better than GREEDUS, does not outperform any other POLYPUS configuration. For this table, we have also included an extremely over-constrained case that comprises instances with an average degree of $d = 20$ to show that the identified trend above—POLYPUS is performing better for more constrained instances—is lasting. At this point, we can already give a positive answer to the hypotheses **H1** and **H2** mentioned earlier, but we will revisit this point after we delve into more detail around the different POLYPUS configurations. In what follows, we have categorized the presentation per policy, viz., UCB1 and MOSS, to avoid cluttering in figures.

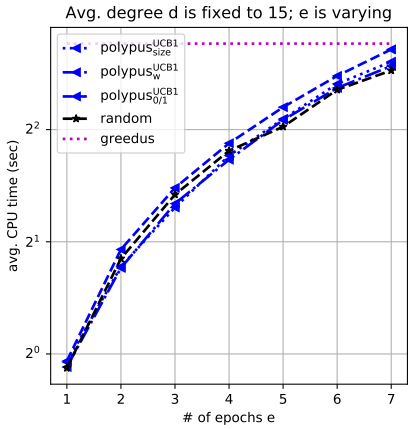
UCB1

In this section, we present results pertaining to the UCB1 policy, see Section 3, along with the three different rewards that were introduced earlier. All of the experimental results are concisely presented in Figure 4.

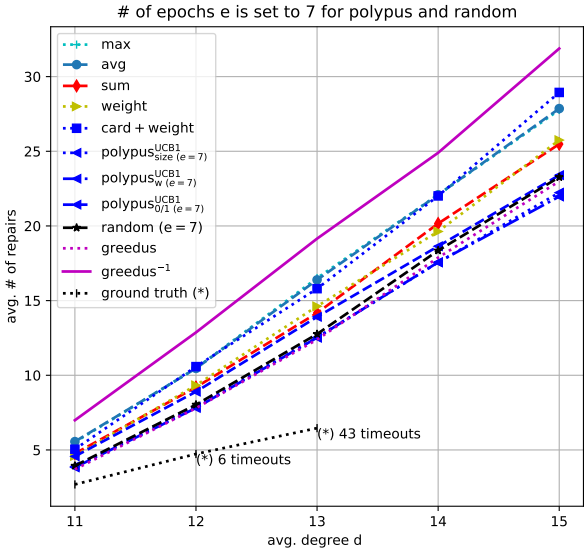
With respect to the UCB1 policy and the W reward (Equation 3), which we have found to be the better performing configuration, we can note that (i) POLY-



(a) Avg. # of repairs to # of epochs e.

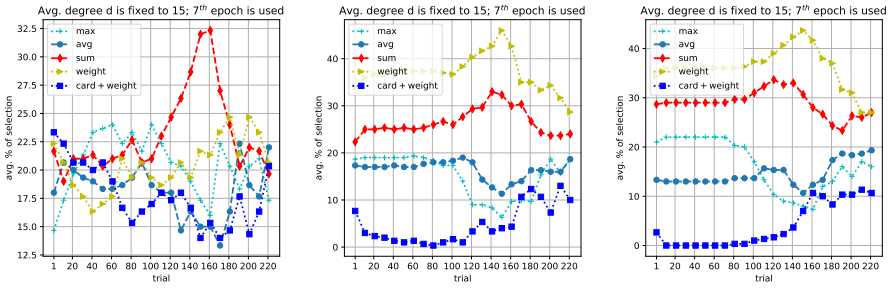


(b) Runtime to # of epochs e.



(c) Avg. # of repairs to avg. degree d.

Figure 4: **UCB1**: Assessing the performance of an implementation of POLYPUS and GREEDUS [10], respectively, with Interval Algebra (IA) network instances of model A($n = 30, d, l = 6.5$) [38]; epochs e do not apply to GREEDUS, i.e., e is always 1 (see Figures 4a and 4b), GREEDUS^{-1} is GREEDUS with the worst (instead of the best) of the 5 strategies being chosen for every instance (see also Figure 4c), RANDOM corresponds to a uniformly random arm selection (see lines 13–14 in Algorithm 1), and the ground truth was (attempted to be) computed with the RC2 MaxSAT solver offering [42] of the PySAT toolkit [43] with a 3 600s timeout for every instance (only timely outputs were used).



(a) 0/1 (Reward 1) (b) SIZE (Reward 2). (c) W (Reward 3)

Figure 5: **UCB1**: Avg. % of arm selection to trials with respect to different rewards, i.e., % of times that an arm was selected over the entire set of instances across all trials.

PUS outperfoms GREEDUS (and RANDOM) on the denser of instances, viz., for $d = 15$ (and $d = 14$), with respect to the avg. number of repairs per instance, for a number of epochs $e \geq 4$ (Figures 4a and 4c), (*ii*) POLYPUS outperfoms GREEDUS on the densest ($d = 15$) of instances with respect to runtime, for a number of epochs $e \leq 7$ (Figure 4b); (*iii*) POLYPUS learns and improves on a strategy to obtain fewer repairs in a QCN as the number of epochs e increases (Figure 4a), outperforming RANDOM (and GREEDUS); and (*iv*) the strategy that is learned by POLYPUS is versatile, ranking and combining arms (orderings of constraints) differently as trials proceed (Figure 5).[3]

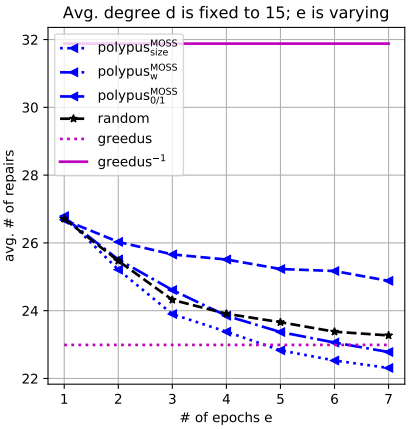
With regard to the different rewards, and, in particular, rewards 1/0 (Equation 1) and SIZE (Equation 2), which we did not detail earlier, we can note that SIZE performs very similarly to, albeit a little worse than W, and 1/0 performs quite poorly even with respect to RANDOM (this is why it was excluded in the synopsis of best results in Table 1). As we remarked in Section 3, the 0/1 values used in this reward do not allow for a good discrimination among the arms, and this becomes quite visible in Figure 5a where it is shown how erratic the choice among arms is. On the other hand, rewards SIZE and W strike a good balance between exploration and exploitation and smoothly synthesize arms/policies in search of a good selection strategy.

MOSS

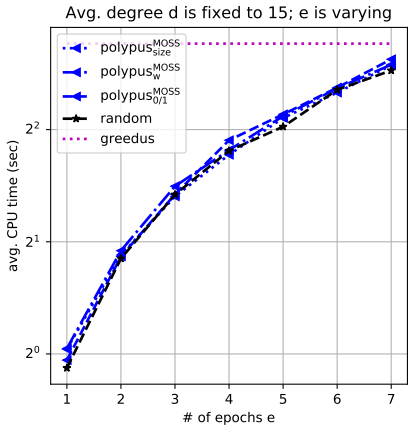
In this section, we present results pertaining to the MOSS policy, see Section 3, along with the three different rewards that were introduced earlier. All of the experimental results are concisely presented in Figure 6.

Unsurprisingly, the different configurations of POLYPUS with the MOSS policy produce results that are qualitatively similar to the ones with the UCB1 policy. However, contrary to the case of UCB1, the better-performing reward

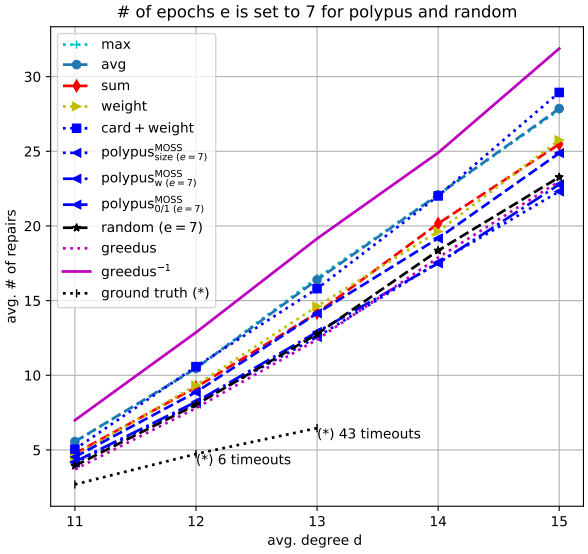
[3]For random selection each of the five arms would be selected $\approx 20\%$ of the time across trials.



(a) Avg. # of repairs to # of epochs e.

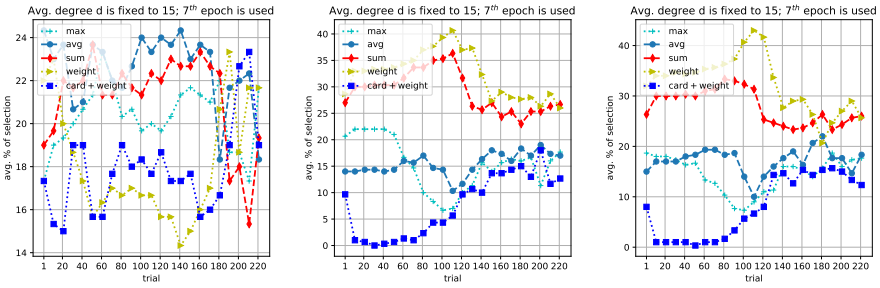


(b) Runtime to # of epochs e.



(c) Avg. # of repairs to avg. degree d.

Figure 6: **MOSS**: Assessing the performance of an implementation of POLYPUS and GREEDUS [10], respectively, with Interval Algebra (IA) network instances of model A($n = 30, d, l = 6.5$) [38]; epochs e do not apply to GREEDUS, i.e., e is always 1 (see Figures 4a and 4b), GREEDUS^{-1} is GREEDUS with the worst (instead of the best) of the 5 strategies being chosen for every instance (see also Figure 4c), RANDOM corresponds to a uniformly random arm selection (see lines 13–14 in Algorithm 1), and the ground truth was (attempted to be) computed with the RC2 MaxSAT solver offering [42] of the PySAT toolkit [43] with a 3 600s timeout for every instance (only timely outputs were used).



(a) 0/1 (Reward 1) (b) SIZE (Reward 2). (c) W (Reward 3)

Figure 7: **MOSS**: Avg. % of arm selection to trials with respect to different rewards, i.e., % of times that an arm was selected over the entire set of instances across all trials.

for POLYPUS policy powered by MOSS, with very few exceptions, is the SIZE reward (see Figure 6a in particular). Referencing Table 1, we can also note that POLYPUS ran with MOSS is often slightly faster than POLYPUS ran with UCB1, but also often slightly worse with respect to number of constraints that are repaired. Again, here the 0/1 reward is the worst-performing one, and by a much larger margin that in the case of UCB1; the arm selection, as shown in Figure 7, remains erratic for this reward.

Finally, we can note that there is a trade-off between efficiency (runtime) and effectiveness (number of repairs) for POLYPUS pertaining to the number of epochs e, as shown in Figures 4a, 6a, 4b, and 6b (and expected by algorithm design alone), and we can also *re*affirm the fact that obtaining optimal results for this type of problem is prohibitive in the general case (see the results of RC2 and also [3, 9, 10]).

Remark 3. With respect to the discussion so far, it might be tempting to question whether RANDOM could outperform POLYPUS for a high(er) number of epochs e (Figure 4a). Even though this could be true for a very high number of e (in the sense that if one randomly tries different orderings of constraints for long enough, then they will eventually get the good result), we have not found it to be the case for $e \leq 1\,000$, at which point the runtime deteriorates significantly anyway, voiding any positive outcomes of the learning approach with regard to number of repairs obtained.

5. Conclusion and Future Directions

In this paper, we presented a reinforcement learning approach for resolving inconsistencies in qualitative constraint networks (QCNs).[4] The use of such net-

[4]Preliminary results of this work appear in [44].

works is typical in constraint programming when we want to represent and reason about intuitive constraints involving spatio-temporal information. Specifically, given a QCN $\mathcal{N}$, our multi-armed bandit approach aims to define a well-suited ordering of the constraints of $\mathcal{N}$ for finding a maximal (or even maxim**um** cardinality) satisfiable subset of them. To this end, our learning approach interacts with a solver and after each trial a reward is returned to measure the performance of the selected action (constraint addition). The reward function is based on the reduction of the solution space of a consistent reconstruction of the input QCN. Our evaluation suggests that we can do better than the state of the art in terms of runtime and number of repairs. Also, our results reaffirm that the problem of resolving inconsistencies in an optimal manner is, generally, too costly, cf. [10] where smaller-sized instances were used.

This study opens up the following avenues for future research. Resolving inconsistencies in a QCN via filtering out problematic constraints and obtaining a maximal satisfiable subset of them, as we are doing here, lies at the heart of other recently introduced inconsistency-related tasks, such as decomposing a QCN into consistent components [45], or "freezing" some of its constraints to emulate a notion of paraconsistency [46] (where minimizing the number of paraconsistent constraints is critical). Thus, our work here can enhance the performance of the aforementioned approaches, and we aim to explore this in our next steps. It also goes without saying that inconsistency resolution in QCNs relates to research in inconsistency measurement in such networks [47], in the sense that different inconsistency measures can inspire different techniques of inconsistency/conflict resolution, and, moreover, being able to resolve inconsistencies (as we do here) can quantify / place a value in such measures. Further, we introduced our reinforcement learning framework in the context of inconsistency resolution, but it could also prove useful when extracting solutions from already satisfiable QCNs, as it has been done in traditional constraint programming [29, 30, 31, 32, 33]. Moreover, the learning approach on its own provides many possibilities for future work in terms of defining reward functions and utilizing other types of reinforcement learning, and making a cross-comparison analysis for uncovering more opportunities to tap into learning. Ultimately, it could be worth viewing a QCN as a CSP or SAT instance [24] and, hence, leveraging different arsenals of techniques that are particular to MaxCSP or MaxSAT respectively [48]. Finally, in this paper we considered a repair to be the act of replacing a hard constraint with another (hard) one, but it would be well-worth exploring cases where there exists a probability distribution over the constraints, as is the case in probabilistic spatiotemporal knowledge bases [49, 50].

References

[1] G. Ligozat, Qualitative Spatial and Temporal Reasoning, ISTE, Wiley, 2013.

[2] F. Dylla, J. H. Lee, T. Mossakowski, T. Schneider, A. van Delden, J. van de Ven, D. Wolter, A Survey of Qualitative Spatial and Temporal Calculi:

Algebraic and Computational Properties, ACM Comput. Surv. 50 (2017) 7:1–7:39.

[3] J. Condotta, A. Mensi, I. Nouaouri, M. Sioutis, L. B. Said, A Practical Approach for Maximizing Satisfiability in Qualitative Spatial and Temporal Constraint Networks, in: ICTAI, 2015.

[4] J. Y. Leung (Ed.), Handbook of Scheduling - Algorithms, Models, and Performance Analysis, Chapman and Hall/CRC, 2004.

[5] T. Staeblein, K. Aoki, Planning and scheduling in the automotive industry: A comparison of industrial practice at German and Japanese makers, Int. J. Prod. Econ. 162 (2015) 258–272.

[6] C. Solnon, V. D. Cung, A. Nguyen, C. Artigues, The car sequencing problem: Overview of state-of-the-art methods and industrial case-study of the ROADEF'2005 challenge problem, Eur. J. Oper. Res. 191 (2008) 912–927.

[7] J. H. Lee, M. Sioutis, K. Ahrens, M. Alirezaie, M. Kerzel, S. Wermter, Neuro-Symbolic Spatio-Temporal Reasoning, in: Compendium of Neurosymbolic Artificial Intelligence, Vol. 369, IOS Press, 2023, pp. 410–429.

[8] Z. Zhou, Abductive learning: towards bridging machine learning and logical reasoning, Sci. China Inf. Sci. 62 (2019) 76101:1–76101:3.

[9] J. Condotta, I. Nouaouri, M. Sioutis, A SAT Approach for Maximizing Satisfiability in Qualitative Spatial and Temporal Constraint Networks, in: KR, 2016.

[10] M. Sioutis, Embarrassingly Greedy Inconsistency Resolution of Qualitative Constraint Networks, in: TIME, 2023.

[11] J. Bailey, P. J. Stuckey, Discovery of Minimal Unsatisfiable Subsets of Constraints Using Hitting Set Dualization, in: PADL, 2005.

[12] J. Marques-Silva, F. Heras, M. Janota, A. Previti, A. Belov, On Computing Minimal Correction Subsets, in: IJCAI, 2013.

[13] A. Nöhrer, A. Biere, A. Egyed, Managing SAT inconsistencies with HUMUS, in: VaMoS, 2012.

[14] J. Marques-Silva, F. Heras, M. Janota, A. Previti, A. Belov, On computing minimal correction subsets, in: IJCAI, 2013.

[15] F. Bacchus, J. Davies, M. Tsimpoukelli, G. Katsirelos, Relaxation Search: A Simple Way of Managing Optional Clauses, in: AAAI, 2014.

[16] E. Grégoire, J.-M. Lagniez, B. Mazure, An Experimentally Efficient Method for (MSS, CoMSS) Partitioning, in: AAAI, 2014.

[17] U. Junker, QUICKXPLAIN: Preferred Explanations and Relaxations for Over-Constrained Problems, in: AAAI, 2004.

[18] A. Felfernig, M. Schubert, C. Zehentner, An efficient diagnosis algorithm for inconsistent constraint sets, Artif. Intell. Eng. Des. Anal. 26 (2012) 53 – 62.

[19] A. Felfernig, R. Walter, J. A. Galindo, D. Benavides, S. P. Erdeniz, M. Atas, S. Reiterer, Anytime diagnosis for reconfiguration, J. Intell. Inf. Syst. 51 (2018) 161–182.

[20] S. Polat Erdeniz, A. Felfernig, M. Atas, LearnDiag: A Direct Diagnosis Algorithm Based On Learned Heuristics, in: KI, 2018.

[21] S. P. Erdeniz, A. Felfernig, M. Atas, Learned Constraint Ordering for Consistency Based Direct Diagnosis, in: IEA/AIE, 2019.

[22] S. Polat Erdeniz, A. Felfernig, M. Atas, Applying matrix factorization to consistency-based direct diagnosis, Appl. Intell. 52 (2022) 7024–7036.

[23] M. Uta, V.-M. Le, A. Felfernig, D. Helic, Learning constraint orderings for direct diagnosis, arXiv.org e-Print archive (2024).

[24] M. Westphal, S. Wölfl, Qualitative CSP, Finite CSP, and SAT: Comparing Methods for Qualitative Constraint-based Reasoning, in: IJCAI, 2009.

[25] M. N. Katehakis, A. F. V. Jr., The Multi-Armed Bandit Problem: Decomposition and Computation, Math. Oper. Res. 12 (1987) 262–268.

[26] J. F. Allen, Maintaining Knowledge about Temporal Intervals, Commun. ACM 26 (1983) 832–843.

[27] P. Auer, N. Cesa-Bianchi, Y. Freund, R. Schapire, The Nonstochastic Multiarmed Bandit Problem, SIAM J. Comput. 32 (2002) 48–77.

[28] J.-Y. Audibert, S. Bubeck, Minimax policies for adversarial and stochastic bandits, in: COLT, 2009.

[29] M. Loth, M. Sebag, Y. Hamadi, M. Schoenauer, Bandit-Based Search for Constraint Programming, in: CP, 2013.

[30] A. Balafrej, C. Bessiere, A. Paparrizou, Multi-Armed Bandits for Adaptive Constraint Propagation, in: IJCAI, 2015.

[31] W. Xia, R. H. C. Yap, Learning Robust Search Strategies Using a Bandit-Based Approach, in: AAAI, 2018.

[32] H. Wattez, F. Koriche, C. Lecoutre, A. Paparrizou, S. Tabary, Learning Variable Ordering Heuristics with Multi-Armed Bandits and Restarts, in: ECAI, 2020.

[33] A. Paparrizou, H. Wattez, Perturbing Branching Heuristics in Constraint Solving, in: CP, 2020.

[34] F. Koriche, C. Lecoutre, A. Paparrizou, H. Wattez, Best Heuristic Identification for Constraint Satisfaction, in: IJCAI, 2022.

[35] R. Sutton, A. G. Barto, Reinforcement Learning: An Introduction, IEEE Trans. Neural Netw. Learn. Syst. 9 (1998) 1054.

[36] M. Sioutis, D. Wolter, Dynamic branching in qualitative constraint-based reasoning via counting local models, Inf. Comput. 281 (2021) 104787.

[37] P. van Beek, D. W. Manchak, The design and experimental analysis of algorithms for temporal reasoning, J. Artif. Intell. Res. 4 (1996) 1–18.

[38] J. Renz, B. Nebel, Efficient Methods for Qualitative Spatial Reasoning, J. Artif. Intell. Res. 15 (2001) 289–318.

[39] B. Nebel, H. Bürckert, Reasoning about Temporal Relations: A Maximal Tractable Subclass of Allen's Interval Algebra, J. ACM 42 (1995) 43–66.

[40] J. Renz, B. Nebel, Qualitative Spatial Reasoning Using Constraint Calculi, in: Handbook of Spatial Logics, Springer, 2007, pp. 161–215.

[41] J. Wehner, M. Sioutis, D. Wolter, On robust vs fast solving of qualitative constraints, J. Heuristics 29 (2023) 461–485.

[42] A. Ignatiev, A. Morgado, J. Marques-Silva, RC2: an Efficient MaxSAT Solver, J. Satisf. Boolean Model. Comput. 11 (2019) 53–64.

[43] A. Ignatiev, A. Morgado, J. Marques-Silva, PySAT: A Python toolkit for prototyping with SAT oracles, in: SAT, 2018.

[44] A. Paparrizou, M. Sioutis, A Reinforcement Learning Approach for Resolving Inconsistencies in Qualitative Constraint Networks, in: SUM, 2024.

[45] Y. Salhi, M. Sioutis, A Decomposition Framework for Inconsistency Handling in Qualitative Spatial and Temporal Reasoning, in: KR, 2023.

[46] Y. Salhi, M. Sioutis, A Paraconsistency Framework for Inconsistency Handling in Qualitative Spatial and Temporal Reasoning, in: ECAI, 2023.

[47] J. Condotta, B. Raddaoui, Y. Salhi, Quantifying Conflicts for Spatial and Temporal Information, in: KR, 2016.

[48] F. Bacchus, M. Järvisalo, R. Martins, Maximum satisfiabiliy, in: A. Biere, M. Heule, H. van Maaren, T. Walsh (Eds.), Handbook of Satisfiability - Second Edition, Vol. 336, IOS Press, 2021, pp. 929–991.

[49] F. Parisi, J. Grant, Knowledge Representation in Probabilistic Spatio-Temporal Knowledge Bases, J. Artif. Intell. Res. 55 (2016) 743–798.

[50] J. Grant, C. Molinaro, F. Parisi, Probabilistic spatio-temporal knowledge bases: Capacity constraints, count queries, and consistency checking, Int. J. Approx. Reason. 100 (2018) 1–28.