A Reinforcement Learning Approach for Resolving Inconsistencies in Qualitative Constraint Networks

Anastasia Paparrizou[0000−0002−6440−0455], Michael Sioutis[0000−0001−7562−2443], and Yoan Thomas

LIRMM UMR 5506, Université de Montpellier & CNRS, France
firstname.lastname@lirmm.fr

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. Early experimental results 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 [17,10]. As an example, within 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.

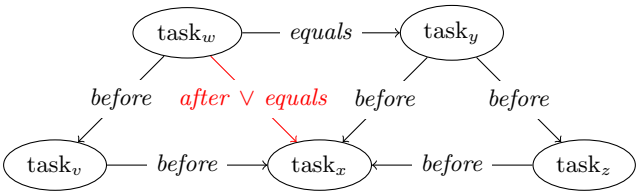


Fig. 1: An illustration of the MAX-QCN problem for a qualitative constraint network (QCN) [8] and the terminology used here; the simplified QCN, which can be viewed as a temporal plan, is inconsistent, and one solution of the MAX-QCN problem, viz., an *optimal* scenario, can be obtained by placing task$_w$ $\{before\}$ task$_x$ and hence repairing the respective constraint in the figure; note that, had we chosen to keep task$_w$ $\{after \vee equals\}$ task$_x$, we would have had to make two repairs.

Context & Motivation

We focus on the MAX-QCN problem [8], 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 scheduling inconsistencies may form due to the unavailability of resources for certain tasks [16]. In particular, timetabling, as an instance of scheduling, involves assigning temporal intervals to tasks that are subject to limited resources. For example, in the context of a railway network, an inconsistency can appear when two (or more) trains are allocated the same railway line in overlapping temporal intervals; to ensure safety of operations, the inconsistency must then be repaired by considering other railway lines, temporal intervals, and/or preferences alike, and minimizing changes so as to perturb the timetable to the least extent possible. Resolving inconsistencies in QSTR extends to the broader context of neuro-symbolic AI architectures dealing with textual entailment or image analysis, among other tasks [15]; 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 [35], 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., [8,9,28]. Of these works, [28] 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, [28] 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, each such procedure taken alone implements a simple instance of basic linear search for computing a maximal satisfiable subset of constraints, cf. [5,19], 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. [21], 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 $\{after \vee equals\}$ 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 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 [14] 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 [28] 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 [28] using a standard dataset of QCNs of Interval Algebra [1], 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 [28]; 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* [17] and defined over an infinite domain D (e.g., $\mathbb{R}$). The base relations of

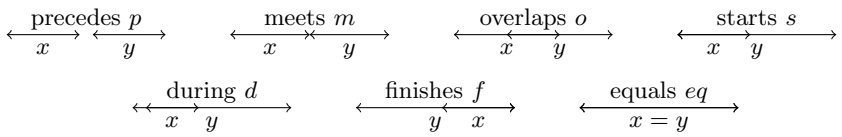


Fig. 2: A representation of the 13 base relations b of IA, each one relating two potential intervals x and y as in $x\ b\ y$; the converse of b, i.e., b^{-1}, can be denoted by bi and is omitted in the figure.

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 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) [1]. 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):

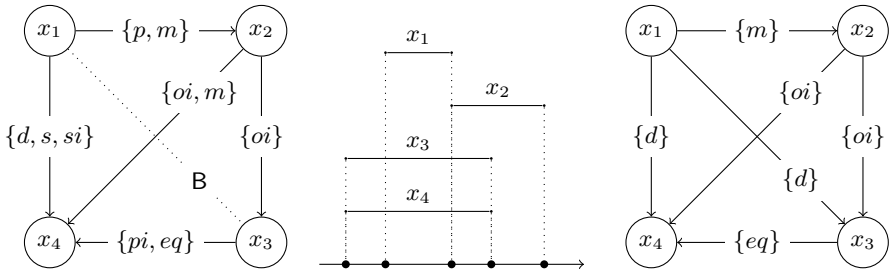
Definition 1. *A* qualitative constraint network (QCN) *is a tuple* (V, C) *where:*

- $V = \{v_1, \ldots, v_n\}$ *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);*
- *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* $v \neq v'$, $C(v, v') = \mathsf{B}$.



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

Fig. 3: Figurative examples of QCN terminology using IA.

The MAX-QCN problem

The MAX-QCN problem has been introduced in the context of QSTR in [8]. 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 [10].

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

Algorithm 1: POLYPUS($\mathcal{N}, \mathcal{A}, r$)

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 $\mathcal{N}$), and a number of epochs e with $e \in \mathbb{N}^+$

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

$P \leftarrow \emptyset$;
$armPulls \leftarrow map()$;
$meanReward \leftarrow map()$;
$counter \leftarrow 0$;
foreach $\alpha \in \mathcal{A}$ **do**
  $armPulls[\alpha] \leftarrow 0$;
  $meanReward[\alpha] \leftarrow 0$;
for n **from** 1 **to** e **do**
  $p \leftarrow \emptyset$;
  $\mathcal{N}' = (V, C') \leftarrow \mathcal{N}_\top$;
  $unprocessedConstraints \leftarrow E(\mathsf{G}(\mathcal{N}))$; `// each constraint initiates a trial`
  while $unprocessedConstraints \neq \emptyset$ **do**
    $\alpha \leftarrow \alpha \in \arg\max_{\alpha' \in \mathcal{A}} \left(meanReward[\alpha'] + \sqrt{\frac{2 \cdot \ln(counter + 1)}{armPulls[\alpha'] + 1}} \right)$; `// UCB1`
    $c \leftarrow \min(\{c' \in \{0, 1, \ldots, |E(\mathsf{G}(\mathcal{N}))| - 1\} \mid \alpha^{-1}(c') \in unprocessedConstraints\})$;
    $\{u, v\} \leftarrow \alpha^{-1}(c)$;
    $C'(u, v) \leftarrow C(u, v)$; $C'(v, u) \leftarrow C(v, u)$;
    $(result, reward) \leftarrow SAT(\mathcal{N}')$; `// see Equation 1 for` $reward$ `description`
    if $result = true$ **then**
      $p \leftarrow p \cup \{\{u, v\}\}$;
    else
      $C'(u, v) \leftarrow \mathsf{B}$; $C'(v, u) \leftarrow \mathsf{B}$;
    $meanReward[\alpha] \leftarrow \frac{(meanReward[\alpha] \cdot armPulls[\alpha]) + reward}{(armPulls[\alpha] + 1)}$;
    $armPulls[\alpha] \leftarrow armPulls[\alpha] + 1$;
    $unprocessedConstraints \leftarrow unprocessedConstraints \setminus \{\{u, v\}\}$;
    $counter \leftarrow counter + 1$;
  $P \leftarrow P \cup \{p\}$;
return $p \in \arg\max_{p' \in P}(|p'|)$; `// largest maximal satisfiable subset is returned`

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 the UCB1 policy [3] that among other policies displayed a good performance in several works related to constraint programming [18,6,34,31,22]. The ultimate

goal is to find a strategy for mapping each trial t 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 its output (lines 13–17 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 16). If the added constraint provokes unsatisfiability, then the constraint is removed from the network (line 21) 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 27). In the following, we analyze the set of arms we deploy and we assess the performance of the chosen arm via a reward function.

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 [29]):

- **max**: choose the constraint that contains the base relation with the most *local models* [29].[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).
- **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 [7].[2]
- **card + weight**: choose the constraint whose smallest decomposition into subrelations of a (maximal) tractable subset $\mathcal{S} \in 2^{\mathsf{B}}$ [24] (e.g., the ORD-Horn set for IA [20]) is the largest one; use weight as a tie-breaker (this is typical in the literature, e.g., [24]).

[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.

[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.

The above strategies have been found to be the best-performing ones in the approach of [28], called GREEDUS, in that they help to produce a lower number of repairs for a given inconsistent QCN compared to other strategies.

Remark 1. GREEDUS [28] 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).

Reward

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 [25,28]; 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 [10], and in what follows we assume the use of such a solver.

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. Further, let $|\mathcal{N}|$ denote the *size* of a QCN $\mathcal{N} = (V, C)$, i.e., its total number of base relations; formally, $|\mathcal{N}| = \sum_{u,v \in V} |C(u,v)|$. 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. In general, the theoretically most relevant reward is the one that optimises directly the objective function (i.e., to maximise the number of the constraints such that the resulting network is satisfiable). Indeed, the first reward we tried was the one that was promoting the arm that was generating the largest maximal satisfiable set of constraints among other arms. We were giving the value 1 as reward to the arm if the solver was producing a satisfiable instance (each time a constraint was added at line 17), and 0 otherwise. So the arm that was producing the largest wrt to the number of constraints satisfiable network had also the biggest mean reward. However, in practice this reward did not work well, as 0/1 values were not allowing for a good discrimination among the arms. Therefore, we tried functions with finer granularity, capturing the space reduction, the constraint weighting inspired from [21,31,32].

After conducting an empirical study on such possible measures, we define the reward for a pulled arm $\alpha \in \mathcal{A}$ as follows:

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

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 that the QCN contains, specifically, the Spearman's rank correlation coefficient between the two variables is ≈ 0.8 [32]. 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.

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 17 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 11–17), 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 MAX-QCN problem, we experimentally compare an in-house implementation of POLYPUS introduced in Section 3 (Algorithm 1) to the greedy approach from [28], called GREEDUS. Interestingly, and in relation to Remark 1, 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 [13] via the PySAT toolkit [12] to compute some ground truth, i.e., optimal number of repairs for our QCNs, to the extent possible.

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 [24], 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 [8,9] (see also the number of timeouts of RC2 in Figure 4d 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

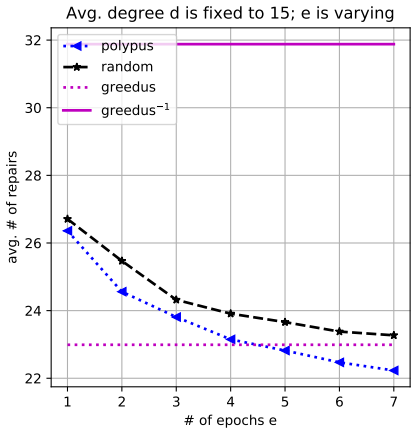
All of the experimental results are concisely presented in Figure 4. We make the following *four* observations, which positively answer hypotheses **H1** and **H2**:

(i) POLYPUS 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 $e \geq 5$ (Figures 4a and 4d);
(ii) POLYPUS outperfoms GREEDUS on the densest 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);
(iv) the strategy that is learned by POLYPUS is versatile, ranking and combining arms (orderings of constraints) differently as trials proceed (Figure 4c).[3]

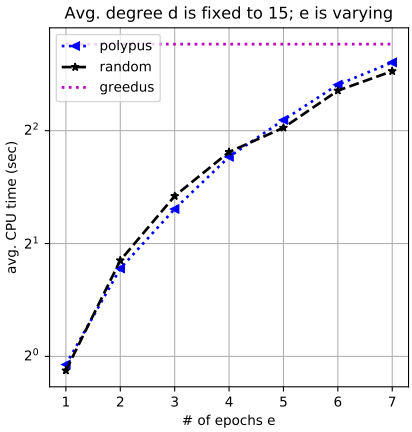
Finally, we can note that there is a trade-off between efficiency (runtime) and effectiveness (number of repairs) for POLYPUS pertaining to the number of epochs e, as shown in Figures 4a and 4b (and expected by algorithm design alone), and we can also *re*affirm the fact that obtaining optimal results for this type of problem is prohibitive in the general case (see the results of RC2 and also [8,9,28]).

Remark 2. 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.

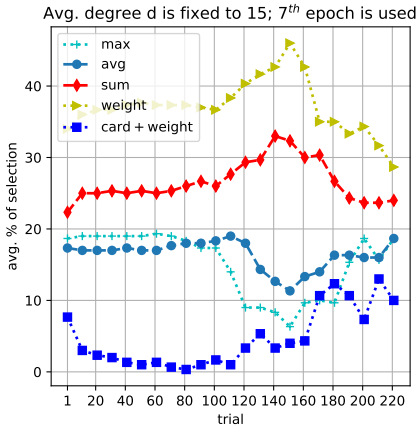
[3] For random selection each of the five arms would be selected $\approx 20\%$ of the time across trials.



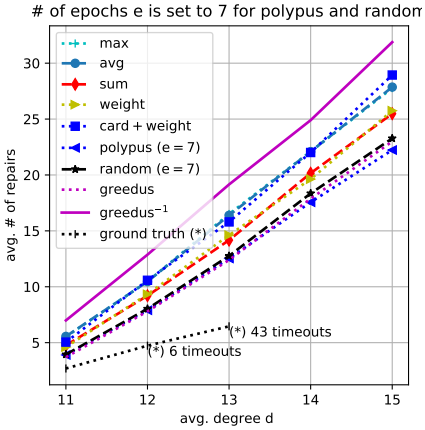
(a) Avg. # of repairs to # of epochs e.



(b) Runtime to # of epochs e.



(c) Avg. % of arm selection to trials.



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

Fig. 4: Assessing the performance of an implementation of POLYPUS and GREEDUS [28], respectively, with Interval Algebra (IA) network instances of model A($n = 30, d, l = 6.5$) [24]; 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 4d), RANDOM corresponds to a uniformly random arm selection (see line 13 in Algorithm 1), and the ground truth was (attempted to be) computed with the RC2 MaxSAT solver offering [13] of the PySAT toolkit [12] with a 3 600s timeout for every instance (only timely outputs were used).

5 Conclusion and Future Directions

In this paper, we presented a reinforcement learning approach for resolving inconsistencies in qualitative constraint networks (QCNs). The use of such networks 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 preliminary 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. [28] 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 [26], or "freezing" some of its constraints to emulate a notion of paraconsistency [27] (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. 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 [18,6,34,31,22]. Moreover, the learning approach on its own provides many possibilities for future work in terms of defining reward functions and utilizing other exploration policies (such as Epsilon-greedy [30], EXP3 [3] and MOSS [2]), 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 [33] and, hence, leveraging different arsenals of techniques that are particular to MaxCSP or MaxSAT respectively [4]. 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 [23,11].

Acknowledgments. The work was partially funded by the Agence Nationale de la Recherche (ANR) for the "Hybrid AI" project that is tied to the chair of Dr. Sioutis, and the I-SITE program of excellence of Université de Montpellier that complements the ANR funding.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Allen, J.F.: Maintaining Knowledge about Temporal Intervals. Commun. ACM **26**, 832–843 (1983)
2. Audibert, J.Y., Bubeck, S.: Minimax policies for adversarial and stochastic bandits. In: COLT. Montreal, Canada (2009)
3. Auer, P., Cesa-Bianchi, N., Freund, Y., Schapire, R.: The Nonstochastic Multi-armed Bandit Problem. SIAM J. Comput. **32**, 48–77 (2002)
4. Bacchus, F., Järvisalo, M., Martins, R.: Maximum satisfiabiliy. In: Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.) Handbook of Satisfiability - Second Edition, vol. 336, pp. 929–991. IOS Press (2021)
5. Bailey, J., Stuckey, P.J.: Discovery of Minimal Unsatisfiable Subsets of Constraints Using Hitting Set Dualization. In: PADL (2005)
6. Balafrej, A., Bessiere, C., Paparrizou, A.: Multi-Armed Bandits for Adaptive Constraint Propagation. In: IJCAI (2015)
7. van Beek, P., Manchak, D.W.: The design and experimental analysis of algorithms for temporal reasoning. J. Artif. Intell. Res. **4**, 1–18 (1996)
8. Condotta, J., Mensi, A., Nouaouri, I., Sioutis, M., Said, L.B.: A Practical Approach for Maximizing Satisfiability in Qualitative Spatial and Temporal Constraint Networks. In: ICTAI (2015)
9. Condotta, J., Nouaouri, I., Sioutis, M.: A SAT Approach for Maximizing Satisfiability in Qualitative Spatial and Temporal Constraint Networks. In: KR (2016)
10. Dylla, F., Lee, J.H., Mossakowski, T., Schneider, T., van Delden, A., van de Ven, J., Wolter, D.: A Survey of Qualitative Spatial and Temporal Calculi: Algebraic and Computational Properties. ACM Comput. Surv. **50**, 7:1–7:39 (2017)
11. Grant, J., Molinaro, C., Parisi, F.: Probabilistic spatio-temporal knowledge bases: Capacity constraints, count queries, and consistency checking. Int. J. Approx. Reason. **100**, 1–28 (2018)
12. Ignatiev, A., Morgado, A., Marques-Silva, J.: PySAT: A Python toolkit for prototyping with SAT oracles. In: SAT (2018)
13. Ignatiev, A., Morgado, A., Marques-Silva, J.: RC2: an Efficient MaxSAT Solver. J. Satisf. Boolean Model. Comput. **11**, 53–64 (2019)
14. Katehakis, M.N., Jr., A.F.V.: The Multi-Armed Bandit Problem: Decomposition and Computation. Math. Oper. Res. **12**, 262–268 (1987)
15. Lee, J.H., Sioutis, M., Ahrens, K., Alirezaie, M., Kerzel, M., Wermter, S.: Neuro-Symbolic Spatio-Temporal Reasoning. In: Compendium of Neurosymbolic Artificial Intelligence, vol. 369, pp. 410–429. IOS Press (2023)
16. Leung, J.Y. (ed.): Handbook of Scheduling - Algorithms, Models, and Performance Analysis. Chapman and Hall/CRC (2004)
17. Ligozat, G.: Qualitative Spatial and Temporal Reasoning. ISTE, Wiley (2013)
18. Loth, M., Sebag, M., Hamadi, Y., Schoenauer, M.: Bandit-Based Search for Constraint Programming. In: CP (2013)
19. Marques-Silva, J., Heras, F., Janota, M., Previti, A., Belov, A.: On Computing Minimal Correction Subsets. In: IJCAI (2013)
20. Nebel, B., Bürckert, H.: Reasoning about Temporal Relations: A Maximal Tractable Subclass of Allen's Interval Algebra. J. ACM **42**, 43–66 (1995)
21. Nöhrer, A., Biere, A., Egyed, A.: Managing SAT inconsistencies with HUMUS. In: VaMoS (2012)
22. Paparrizou, A., Wattez, H.: Perturbing Branching Heuristics in Constraint Solving. In: CP (2020)

23. Parisi, F., Grant, J.: Knowledge Representation in Probabilistic Spatio-Temporal Knowledge Bases. J. Artif. Intell. Res. **55**, 743–798 (2016)
24. Renz, J., Nebel, B.: Efficient Methods for Qualitative Spatial Reasoning. J. Artif. Intell. Res. **15**, 289–318 (2001)
25. Renz, J., Nebel, B.: Qualitative Spatial Reasoning Using Constraint Calculi. In: Handbook of Spatial Logics, pp. 161–215. Springer (2007)
26. Salhi, Y., Sioutis, M.: A Decomposition Framework for Inconsistency Handling in Qualitative Spatial and Temporal Reasoning. In: KR (2023)
27. Salhi, Y., Sioutis, M.: A Paraconsistency Framework for Inconsistency Handling in Qualitative Spatial and Temporal Reasoning. In: ECAI (2023)
28. Sioutis, M.: Embarrassingly Greedy Inconsistency Resolution of Qualitative Constraint Networks. In: TIME (2023)
29. Sioutis, M., Wolter, D.: Dynamic branching in qualitative constraint-based reasoning via counting local models. Inf. Comput. **281**, 104787 (2021)
30. Sutton, R., G. Barto, A.: Reinforcement Learning: An Introduction. IEEE Trans. Neural Netw. Learn. Syst. **9**, 1054 (1998)
31. Wattez, H., Koriche, F., Lecoutre, C., Paparrizou, A., Tabary, S.: Learning Variable Ordering Heuristics with Multi-Armed Bandits and Restarts. In: ECAI (2020)
32. Wehner, J., Sioutis, M., Wolter, D.: On robust vs fast solving of qualitative constraints. J. Heuristics **29**, 461–485 (2023)
33. Westphal, M., Wölfl, S.: Qualitative CSP, Finite CSP, and SAT: Comparing Methods for Qualitative Constraint-based Reasoning. In: IJCAI (2009)
34. Xia, W., Yap, R.H.C.: Learning Robust Search Strategies Using a Bandit-Based Approach. In: AAAI (2018)
35. Zhou, Z.: Abductive learning: towards bridging machine learning and logical reasoning. Sci. China Inf. Sci. **62**, 76101:1–76101:3 (2019)