Efficient constraint learning for stream reasoning

Mourad Hassani, Amel Bouzeghoub

Efficient Constraint Learning For Stream Reasoning

Mourad Hassani
Samovar, Télécom SudParis
Institut Polytechnique de Paris
Palaiseau, France
Mourad.Hassani@telecom-sudparis.eu

Amel Bouzeghoub
Samovar, Télécom SudParis
Institut Polytechnique de Paris
Palaiseau, France
Amel.Bouzeghoub@telecom-sudparis.eu

***Abstract*—Nowadays, large amounts of data are generated by a wide range of data streams. This data contains valuable knowledge to support decision-making processes in a variety of application domains, such as smart cities, social media analysis, and the Internet of Things. To address the constantly changing data, stream reasoning has emerged as a crucial method for performing logical reasoning on streams of data. This paper focuses on the limitations of existing stream reasoning systems based on Answer Set Programming (ASP). It proposes a novel solution based on Deep Reinforcement Learning (DRL) to handle continuous data streams efficiently. Existing ASP-based stream reasoning systems face challenges in managing their internal state and effectively handling data streams over extended periods. To overcome these limitations, we propose an approach that integrates cache management techniques with DRL and ideas from heuristics developed for the Conflict-Driven Constraint Learning (CDCL) algorithm. DRL is chosen for its ability to outperform traditional Reinforcement Learning (RL) algorithms in managing complex tasks and generalizing learned policies to unseen situations. The research contributions include a DRL-based framework that efficiently manages learned constraints in ASP-based stream reasoning engines. The proposed approach is compared to recent work in the literature, and the results demonstrate its benefits in terms of average cache utilization. The findings showcase the potential of the proposed DRL-based solution for improving the performance and widespread implementation of stream reasoning with ASP engines. By effectively handling the dynamic and continuous nature of data streams.**

***Index Terms*—Data Streams, Stream Reasoning, Reinforcement Learning, ASP, Constraint Learning**

I. Introduction

The explosion of data worldwide poses new challenges and opportunities for research, business, and society in general. According to the International Data Corporation (IDC), the amount of global data is set to grow from 33 zettabytes (Zb) in 2018 to 175 Zb in 2025. This data is generated by technologies such as autonomous vehicles, IoT devices, augmented reality, and 5G communications.

The Web and the Internet of Things are dynamic environments where data streams are a valuable source of knowledge for many use cases, such as social media analytics, smart cities, safety, and autonomous vehicle control. Stream reasoning is an approach that studies how to perform logical reasoning on constantly changing data. Since this data has a limited validity period and needs to be processed quickly to produce relevant results, reasoning about the data must be done in near-real time.

Stream reasoning can be used to extract implicit knowledge from data flows. For example, it can be used to detect anomalies in a data stream and provide clear explanations to quickly understand the situation. Answer Set Programming (ASP) is a declarative programming paradigm. It is a powerful rule-based language for knowledge representation and reasoning. The Logic-based framework for Analytic Reasoning over Streams (LARS) extends ASP to specify complex decision problems on streaming data. Several specific stream reasoners supporting LARS, e.g., Ticker [11] or Laser [24], have recently been proposed. The limitations of these modern stream reasoning systems in terms of expressivity of the LARS formalism they support pushed some researchers to focus on stream reasoning systems based on ASP. The effectiveness of ASP solvers, which utilize Conflict-Driven Constraint Learning (CDCL), has been demonstrated in modeling and solving a wide range of problems in stream reasoning. However, a significant drawback arises from their algorithmic limitations when operating continuously over extended periods. These solvers lack an efficient mechanism to manage their internal state while sequentially handling instances from an input stream. Their current design mainly focuses on finding one or more solutions for individual problem instances during each execution, making them unsuitable for continuous, real-time applications. In the context of CDCL algorithm, the constraint learning module plays a crucial role. It aims to derive a concise constraint that explains the cause of a conflict when encountered during the search process, which helps in pruning the search space. The effectiveness of CDCL implementations heavily relies on how learned constraints are managed, as the size of the constraints database (DB) grows exponentially with each conflict.

To mitigate this combinatorial explosion, various strategies have been proposed to maintain a reasonable-sized learned constraints DB [1] [2] [3] [4]. These strategies involve selective constraint filtering, activity heuristics to delete irrelevant constraints, quantifying the quality of constraints based on certain criteria, and dynamic freezing and activation of learned constraints. Researchers have also explored the use of Reinforcement Learning (RL) to generate a constraint removal policy [5] aimed at improving the execution time of the CDCL algorithm. This approach has proved to be promising in achieving better performance.

However, these approaches are primarily designed for iso-

lated problems and may not be suitable for scenarios that involve continuous data streams, such as stream reasoning. Stream reasoning requires adapting and enhancing the established approaches to handle streaming data effectively while maintaining accuracy and effectiveness. In this context, the authors in [6] propose an RL-based solution specifically tailored for stream reasoners using ASP. Their objective is to identify valuable constraints that enhance the reasoning process for a data stream. They introduce an efficient data caching mechanism for storing computed constraints in the context of stream reasoning.

Positive aspects of the proposed approach include intelligent management of learned constraints for stream reasoning, leading to improved solver and ASP-based stream reasoning engine performances. However, there are also concerns related to the RL agent's dependence on the learning rate set by the user, which could affect the performance. Another issue is the need to store a maximum number of constraints in the RL agent's cache, which can become a limitation if the number of generated constraints per instance exceeds the cache capacity. Addressing these limitations is essential for practical application and broader adoption of the proposed RL-based solution for stream reasoning with ASP engines.

This research paper aims to address the existing limitations in stream reasoning based on ASP and enhance its efficiency and effectiveness by adopting a novel solution based on Deep Reinforcement Learning (DRL). The choice of DRL stems from its ability to outperform traditional RL algorithms in handling complex tasks and its ability to generalize learned policies to previously unseen situations through representation learning capabilities. To accommodate the relatively large discrete action space required by our approach, we have opted for the *Wolpertinger* architecture [14]. This decision choice ensures that the agent's performance remains unaffected even as it encounters increasingly complex problems. This paper presents a significant advancement in the domain of stream reasoning with ASP, showing great potential for enhancing performance and widespread implementation across diverse practical applications. The study introduces the following contributions:

- A novel approach that adapts the CDCL algorithm to cope with data streams. It combines methodologies derived from cache management with DRL and heuristics employed in the CDCL algorithm.
- A DRL framework called CLOSER, short for Constraint Learning fOr StrEam Reasoning, to effectively manage learned constraints in stream reasoning engines. This framework seamlessly integrates with ASP solvers, making it compatible with ASP-based stream reasoning systems.
- An experiment based on a Gym environment that compares our proposed approach with recent work in the literature on the average cache utilization revealed the benefits of our approach

The rest of the paper is organized as follows. Section II presents the preliminaries, and Section III discusses the literature review. Then, Sections IV and V present the system model and the problem formulation, respectively. In section VI, we present the proposed approach. The experiments and the results are detailed in Section VII. Finally, Section VIII concludes and gives some perspectives.

II. Preliminaries

LARS is a logic-based framework for stream reasoning that extends ASP with generic window operators and additional controls to specify complex decision problems on streaming data [8]. LARS enables reasoning about continuous data streams in real-time, and provides constructs for representing temporal data, including time intervals and duration. LARS uses *window functions* to access parts of the data stream. It also offers *temporal modalities* : the *at* operator $@_t$, the *somewhere* operator $\Diamond$ and the *everywhere* operator $\Box$. We focus in this work on stream reasoning based on translating Plain LARS programs into ASP programs. This technique was extensively studied in [10] and [11].

An ASP program is a set of rules of the form:

$$a \leftarrow l_1, ..., l_n \tag{1}$$

where a is an atom and $l_1, ..., l_n$ are literals for $n \geq 0$. An atom is an expression of the form $p(t_1, ..., t_k)$, where p is a predicate symbol and $t_1, ..., t_k$ are terms, i.e., either a variable or a constant (a function could be represented using predicates). A literal l is either an atom a_i (positive) or its negation $not\ a_i$ (negative), where not is negation as failure; the complement (opposite) of l is denoted by $\bar{l}$, and we let $\overline{L} = \{\bar{l} | l \in L\}$. An atom, a literal, or a rule is ground, if no variables appear in it. The grounding of a program Π is the set Π^G of all ground rules constructible from rules $r \in \Pi$ by substituting each variable in r with some constant appearing in Π.

Given a rule r of the form (1), the set $H(r) = \{a\}$ is the head and the set $B(r) = B^+(r) \cup B^-(r) = \{l_1, ..., l_n\}$ is the body of r, where $B^+(r)$ and $B^-(r)$ contain the positive and negative body literals, respectively. A rule r is a fact if $B(r) = \emptyset$ and a constraint if $H(r) = \emptyset$.

The semantics of an ASP program Π is given for its ground instantiation Π^G. Let A be the set of all ground literals occurring in Π^G. An interpretation is a set $I \subseteq A \cup \overline{A}$ of literals that is consistent, i.e., $I \cap \overline{I} = \emptyset$; each literal $l \in I$ is true, each literal $l \in \overline{I}$ is false, and any other literal is undefined. An interpretation I is total, if $A \subseteq I \cup \overline{I}$. An interpretation I satisfies a rule $r \in \Pi^G$, if $H(r) \subseteq I$ whenever $B(r) \subseteq I$. A model of Π^G is a total interpretation I satisfying each $r \in \Pi^G$. Moreover, I is stable (an answer set), if I is a $\subseteq$-minimal model of the reduct $\{H(r) \leftarrow B^+(r) | r \in \Pi^G, B^-(r) \cap I = \emptyset\}$ [12]. Any answer set of Π^G is also an answer set of Π.

Modern implementations of ASP solvers, including Wasp [19], are based on the CDCL algorithm [13]. Wasp computes an answer set for a given propositional program P using the

Algorithm 1 [18]. The process begins with an empty interpretation I as input. The Propagate function extends I with deduced literals (line 1) while keeping track of the reasons for each deduction by constructing an implication graph representation. This function is similar to unit propagation used in SAT solvers [21] but also leverages the characteristics of ASP to perform other inferences (e.g., utilizing the knowledge that each answer set is a minimal model). Propagate returns false if an inconsistency or conflict is detected and true otherwise. If Propagate returns true and I is a total interpretation (line 2), CheckModel is invoked (line 3) to verify if I constitutes an answer set. If the stability check succeeds, I is returned; otherwise, it is further analyzed by the AnalyzeConflictAndLearnConstraints procedure. Alternatively, if there are undefined literals in I, a heuristic criterion is employed to choose one (denoted as l). The computation continues with a recursive call to ComputeAnswerSet on $I \cup \{l\}$ (lines 8-9). If the recursive call returns an answer set, the computation terminates by returning it (line 11). Otherwise, the algorithm backtracks and unwinds choices until the consistency of I is restored, while propagating the consequences of learned constraints derived from conflict analysis. Conflicts detected during propagation are analyzed by the AnalyzeConflictAndLearnConstraints procedure (line 17).

This procedure is typically complemented with heuristic techniques that control the number of generated constraints (which can be exponential) and potentially restart the computation to explore different branches of the search tree.

Algorithm 1 Compute Answer Set

Input: Interpretation I **for program** P
Output: Answer set or $INCOHERENT$

```
1:  while Propagate(I) do
2:      if I is total then
3:          if CheckModel(I) then
4:              return I;
5:              break;
6:          end if
7:      end if
8:      l ← ChooseUndefinedLiteral();
9:      l' ← ComputeAnswerSet(I ∪ {l})
10:     if l' ≠ INCOHERENT then
11:         return I';
12:     end if
13:     if there are violated constraints then
14:         return INCOHERENT;
15:     end if
16: end while
17: AnalyzeConflictAndLearnConstraints(I);
18: return INCOHERENT;
```

III. Related Work

In current implementations of the CDCL algorithm, the constraint learning module is recognized as one of the most important components. This importance stems from the role that constraints play in the CDCL algorithm. Constraint learning is initiated when the current branch of the search tree leads to a conflict, and its aim is to derive a constraint that succinctly expresses the causes of the conflict. This learned constraint is then used to prune the search space. In practice, the effectiveness of CDCL implementations depends heavily on the strategy used to manage the learned constraints DB. Indeed, as each conflict adds a new constraint to the learned constraints DB, its size grows exponentially. To limit the impact of this combinatorial explosion, several strategies have been proposed. The aim of these strategies is to maintain a reasonable-sized learned constraints DB by eliminating constraints deemed irrelevant for subsequent searches.

In the literature, two main approaches are currently available to delete learned constraints:

- Static approaches, where a numerical value is assigned to the learned constraint during the conflict processing. This value represents the constraint's activity score and is used to weigh each constraint according to its relevance to the search process.
- Dynamic approaches where the activity values associated with the learned constraints change throughout the CDCL algorithm execution. Constraints deemed irrelevant (inactive) are eliminated from the DB.

In order to reduce the number of constraints to be stored during the execution of the CDCL algorithm, many approaches were developed in the field of satisfiability testing (SAT) [21]. In what follows, we present a non-exhaustive list of the main approaches based on a review of the literature we conducted. One of the first works to explore clause deletion strategies is GRASP [1]. It uses a constraint filtering strategy with a selective approach for controlling the size of the constraint DB, where constraints resulting from conflicts below a certain size threshold are added to the DB, while larger constraints are retained as long as they remain unit constraints. In [2], the authors suggest an aggressive constraint suppression method utilizing an activity heuristic, where constraints with low activity or limited contribution to recent conflict analysis are deemed irrelevant, and the constraint limit is progressively expanded after each restart. In [3], the authors use the number of different levels of Literals Blocks Distance (LBD) [1] involved in a given learned constraint to quantify the quality of the learned constraints. Constraints with a smaller LBD are considered more relevant. In [4], the authors proposed an approach based on dynamic freezing and activation of learned constraints. At a given search state, using a relevant selection function based on progress saving, it activates the most promising learned constraints while freezing irrelevant ones.

These works embody extensive human experience and rigorous experimentation, resulting in the development and successful application of highly effective heuristics. These achieve-

[1]The term "Literal Block Distance" (LBD) is used to denote the number of decision levels in a learned constraint. This value serves as an indicator used to assess the significance of a learned constraint.

ments represent significant progress in the field. Nevertheless, an intriguing question arises: Could alternative heuristics hold the promise of achieving even higher levels of performance? As a consequence, the pursuit of improved heuristics remains an ongoing challenge.

In an effort to address this challenge, the authors in [5] present an approach based on RL that generates a heuristic specifically designed to handle constraint deletions within the CDCL algorithm. The aim of this work is to use machine learning to automatically learn this heuristic. The authors employ RL techniques to develop a constraint removal policy aimed at improving the execution time of the CDCL algorithm. The problem is formulated as an episodic RL task, where an agent interacts with an environment over discrete and finite time steps. During each time step, which corresponds to a garbage collection event, the agent receives observations from the environment, takes actions, and accumulates rewards. To make decisions about which constraints to retain, the agent uses a policy gradient algorithm to directly optimize a stochastic policy. At each time step, the agent determines which constraints, out of a total of N, should be preserved or discarded. This is achieved by generating an integer LBD threshold as an action. Any constraint with an LBD value greater than this threshold is selected for deletion.

The previously presented studies has made significant contributions to the field. However, these initial approaches were mainly designed for isolated problems and are not well-suited for scenarios that demand the processing of continuous data streams, especially in the context of stream reasoning where modern stream reasoners like TICKER [11] or the distributed reasoner [10] utilize ASP solvers to generate answer streams for newly incoming data. This approach, referred to as RESTART [6], involves creating a fresh instance of an ASP solver each time reasoning is invoked, thereby utilizing learned constraints for a single reasoning cycle. The existing methods fall short when it comes to efficiently addressing stream-based problems. Hence, the primary challenge in this area is to adapt and enhance these established approaches to effectively handle streaming data while maintaining accuracy and effectiveness.

Overcoming this challenge is crucial as it lays the groundwork for advancing stream reasoning applications and unlocking its full potential across various domains and industries. In their research paper, the authors of [6] suggest an innovative approach to address this problem. They propose an RL-based solution that is specifically tailored for stream reasoners using ASP. The objective is to identify the most valuable constraints that can enhance the overall reasoning process for a data stream. They propose an efficient data caching mechanism for storing the computed constraints in the context of stream reasoning. Additionally, the article presents an extension to the WASP [19] ASP solver, allowing for data exchange with an external cache. The effectiveness of the learned constraints is influenced by the learning rate. When the learning rate is high, the system assumes strong interdependence among the data in the stream, enabling previously cached data to assist in solving subsequent problems. Conversely, with a low learning rate, the system becomes more skeptical and adjusts its estimates gradually. In this approach, the constraint learning problem is transformed into a multi-armed bandit problem with multiple plays [7], which can be formulated as follows: given a set $N = \{n_1, ..., n_k\}$ of random variables of unknown mean $\Theta = \{\theta_i = E[n_i] | n_i \in N\}$ which are i.i.d. in time, at each instant t, a set $N_t \subseteq N$ is selected according to the weights W_{t-1} associated with the variables in N. The selected variables N_t are observed at t, and a reward vector R_t is determined for them, allowing to calculate a new set of weights W_t that better approximates Θ.

The authors consider the following reward function:

$$R_t = a \cdot [1 - 2 \cdot LBD_t(c) + ua_t(c) - uf_t(c) - 0.25 \cdot nf_t(c)] \tag{2}$$

Where $LBD_t(c)$ is the value of the LBD heuristic calculated for a generated constraint c, a is a coefficient chosen in relation to the number of decision levels of a basic program, $uf_t(c) = 1$ if c has been frozen, i.e. it was not initially supplied to the solution algorithm, but rediscovered during its execution, $ua_t(c) = 1$ if a constraint was supplied to and used by the solution algorithm and $nf_t(c) = 1$ if the constraint was frozen and not rediscovered. The coefficient nf allows the algorithm to penalize and subsequently remove constraints that have been frozen for a long time.

In the previous work, several positive aspects and advancements are evident. Notably, it introduces intelligent management of learned constraints in the context of stream reasoning, a significant update to traditional solutions. This innovation proves to enhance the performance of modern solvers and ASP-based stream reasoning engines by efficiently reusing data from previous instances. By leveraging this learned knowledge, subsequent instances are solved more effectively, leading to improved overall performance. This work represents a notable step forward in the field of stream reasoning with ASP engines.

However, certain shortcomings must also be considered. One notable concern arises from the dependence of the RL agent's behavior on the learning rate set by the user. Inappropriate settings of the learning rate could potentially lead to suboptimal performance and hinder the agent's ability to achieve desirable results. Another critical issue lies in the need for the RL agent to store a maximum number of constraints in its cache. This requirement could become a significant limitation, particularly if the solver generates an excessive number of constraints that surpass the cache's capacity. When the cache cannot accommodate all generated constraints, the agent's action values will not be properly updated, which can negatively impact the overall efficiency and accuracy of the system.

Addressing the limitations mentioned above is of utmost importance to ensure the practical application and widespread acceptance of the proposed solution. To achieve this objective, our research paper makes several significant contributions. Firstly, we propose a novel approach that modifies

the CDCL algorithm to handle data streams by incorporating techniques from cache management, DRL, and heuristics from the CDCL algorithm. Secondly, we introduce CLOSER (Constraint Learning fOr StrEam Reasoning), a DRL framework to effectively manage learned constraints in stream reasoning engines, seamlessly integrating it with ASP solvers, thereby ensuring compatibility with all ASP-based stream reasoning systems. Lastly, we conduct an experiment comparing our proposed approach to recent work in the field concerning average cache utilization. The results demonstrate the clear advantages and benefits of their approach over existing methodologies.

IV. Closer Architecture

CLOSER involves two separate systems, namely the WASP reasoner and the DRL agent, which work collaboratively to optimize the management of learned constraints. The communication between both systems is facilitated thanks to the use of sockets, allowing seamless data exchange. The key components and interactions are presented in Fig.1.

The WASP system is responsible for executing Algorithm1 which involves constraint learning. Whenever WASP learns a new constraint during the learning process, it sends this information to the DRL agent. Each time the solver deletes a constraint from the set of constraints being considered, it sends a message to the DRL agent to inform about this event. The DRL agent takes up the responsibility of managing the cache where constraints are stored. To facilitate efficient communication and tracking, the communication between the two systems utilizes the unique IDs generated for constraints when they were initially generated. These IDs serve as reference points for identifying specific constraints during data exchange.

Before executing Algorithm1 the solver retrieves all the constraints stored in the cache. This step ensures that the solver has access to the most up-to-date and relevant set of constraints for constraint learning and problem-solving. After executing Algorithm1 the solver generates a reward that reflects the quality of the constraints stored in the cache. This reward is then sent back to the DRL agent, providing valuable feedback to the agent about the effectiveness of the constraint set and the overall problem-solving process.

V. Problem Formulation

This section addresses the constraint caching problem within the framework of stream reasoning using ASP. Our primary objective is to optimize cache performance by maximizing the number of cached constraints that are accessed by the ASP system while solving the instance.

The performance of the cache is computed using the formula:

$$P = \Sigma_{n=1}^{M} 1_n \tag{3}$$

Where :

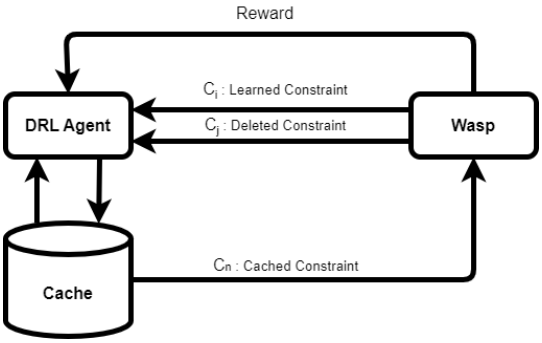

Fig. 1. Architecture of CLOSER

$$1_n = \begin{cases} 1, \text{ if the cached constraint } C_n \\ \quad \text{is accessed by the solver,} \\ 0, \text{ otherwise} \end{cases} \tag{4}$$

The value we are trying to maximize can be expressed using the maximization problem :

$$\text{Problem: } \underset{\phi}{Maximize}\ P \tag{5}$$

$$\text{Subject to } \Sigma_{n=1}\phi_n \leq M \tag{6}$$

where ϕ is the vector that records the states of all constraints (describing whether they are cached or not), and each element ϕ_n in the vector is an indicator to show if the constraint is cached :

$$\phi_n = \begin{cases} 1, \text{ if the constraint } C_n \text{ is cached,} \\ 0, \text{ otherwise} \end{cases} \tag{7}$$

We opt for formula (3) as it gives precedence to the cached constraints, rather than the conventional cache hit rate formula that focuses on requests. This choice is based on the appropriateness of formula (3) for our specific use case.

VI. The proposed Approach

In this section, we present our DRL-based approach to constraint caching for stream reasoning. Our approach for managing learned constraints is based on the *Wolpertinger* architecture [2] [14], shown in Algorithm 2 which is an extension of the actor-critic framework [20]. Within this approach, we find an efficient action-generating actor and a critic to enhance the actor's decisions. The architecture was selected due to its remarkable capability in handling large discrete action spaces, which is particularly relevant for our research, as we encounter a significantly large number of potential actions. Specifically, our DRL agent generates the LBD value, representing constraints it deems most effective in addressing forthcoming instances in the data stream. The LBD value

[2]Our implementation of the wolpertigner architecture is based on the github repository at the address: https://github.com/ChangyWen/wolpertinger_ddpg

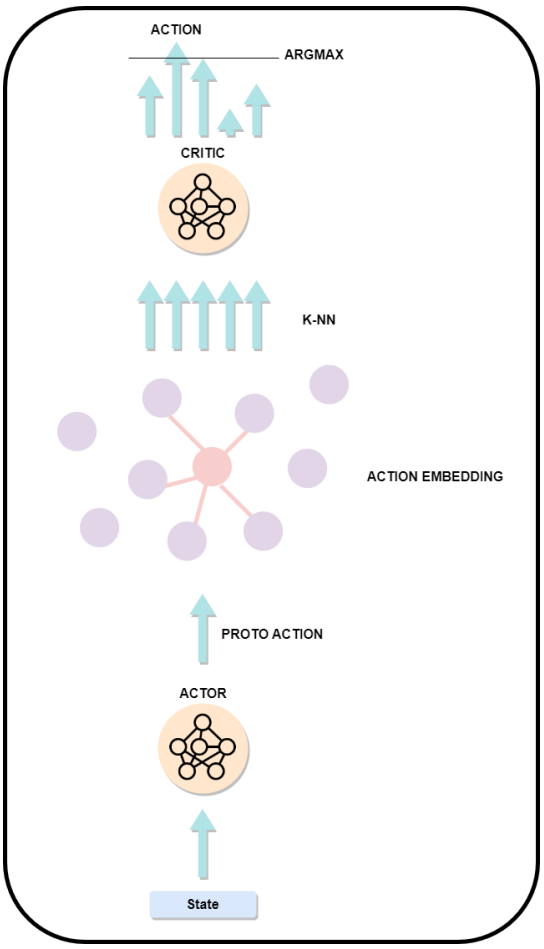

Fig. 2. The DRL Agent Architecture [14]

Algorithm 2 Actor-Critic Algorithm for Constraint Caching

Initialize actor network $\mu(s|\theta^{\mu})$ and critic network $Q(s,a|\theta^{Q})$ with random weights θ^{μ} and θ^{Q}.
Initialize target networks: $\theta^{\mu'} \leftarrow \theta^{\mu}$ and $\theta^{Q'} \leftarrow \theta^{Q}$
Initialize empty replay buffer D
Initialize features space F
for $t = 1$ **to** T **do**
    Receive Observation s_t
    Receive proto action from actor network : $\hat{a}_t = \mu(s_t|\theta^{\mu})$
    Receive k closest actions $A_k = g_k(\hat{a}_t)$
    Receive best action : $a_t = argmax_{a_j \in A_K} Q(s_t, a_j|\theta^{Q})$
    Execute action a_t and observe reward r_t and new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in D
    Sample randomly a mini batch of b transitions (s_i, a_i, r_i, s_{i+1}) from the replay buffer D
    Calculate target for every i in b : $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update the critic by minimizing the MSE Error : $L = \frac{1}{b}\Sigma_i(y_i - Q(s_i, a_i|\theta^{Q}))^2$
    Update the actor based on the gardient : $\nabla_{\theta^{\mu}} J \approx \frac{1}{b}\Sigma_i \nabla_{\theta^{\mu}} \mu(s_i|\theta^{\mu}) \nabla_a Q(s_i, a|\theta^{Q})|_{a=\mu(s_i)}$
    Update the target networks using a soft update based on $\tau \ll 1$: $\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}$ and $\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'}$
    Update the cache state
    Update features space F
    Update cache hit rate
end for

ranges from 2 to the size of the largest constraint acquired by the solver. Given that the size of learned constraints is beyond our control, the sheer magnitude of possible actions becomes even more pronounced.

The DRL agent, as shown in Fig. 2, consists of three main parts: actor network, K-Nearest Neighbors (KNN) and critic network. The actor network takes as input statistics about the LBD values of cached constraints, and outputs a proto action $\hat{a}$. The KNN receives the proto action $\hat{a}$ and calculates the L2 distance between every valid action (that is part of our action space) and then returns the K-nearest possible actions. The critic network then receives the K actions and refines the actor network on the basis of the Q value. The Deep Deterministic Policy Gradient (DDPG) [23] is applied to update both critic and actor networks.

Actor: The actor network is defined as a function parameterized by θ^{μ}, It takes as input the state $s \in \mathbb{R}^S$ and outputs a proto action $\hat{a} \in \mathbb{R}^a$.

K-nearest neighbors: The KNN is used to avoid having poor decision making due to the reduction of the huge action space to one action. To achieve that, the KNN expands the proto action $\hat{a}$ to a set of valid actions in our action space :

$$A_k = g_k(\hat{a}_t)$$

$$g_k = \underset{a \in A}{\text{k}}argmin\, |a - \hat{a}| \tag{8}$$

Critic: The critic refines the choice of action by selecting the highest-scoring action according to the Q-values :

$$\pi_{\theta}(s) = \underset{s \in \mathbb{R}^S, \hat{a} \in A_k}{argmax} Q(s, a|\theta^{Q}) \tag{9}$$

The actor policy is updated using the gradient:

$$\nabla_{\theta^{\mu}} J \approx \frac{1}{b}\Sigma_i \nabla_{\theta^{\mu}} \mu(s_i|\theta^{\mu}) \nabla_a Q(s_i, a|\theta^{Q})|_{a=\mu(s_i)} \tag{10}$$

The parameters of the actor and critic networks are updated using the following soft updates :

$$\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'} \tag{11}$$

$$\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'} \tag{12}$$

At the beginning of each epoch t, the agent observes the current state s_t. Subsequently, the actor network generates a proto-actor based on the current policy (with some exploration noise), which is then passed to the KNN module. The expanded action set is evaluated by the critic network to estimate the potential value of each action. For updating the actor and critic networks, a minibatch of size b is randomly sampled from the memory D (replay buffer). This memory allows the networks to learn from past experiences and refine their performance. The complete process is outlined in Algorithm 2, providing a step-by-step description of CLOSER.

VII. Evaluation

A. Dataset

To evaluate our approach, we chose to work on the N-Queens Completion problem [22]. We chose this problem because it is a well-known problem in the community and it was used as a benchmark in many ASP competitions. In addition, it can only be solved using stream reasoning based on a translation of LARS program into ASP program. The N-Queens Completion problem is a combinatorial problem based on the game of chess that tries to answer thiss question: "How can N queens, with k queens already on an $N \times N$ chessboard in non-attacking positions, be arranged in such a way that none of the N queens can attack each other?". It is an interesting benchmark for stream reasoning systems as it allows for adjusting its difficulty using the parameter n. The dataset employed in our evaluation was originally introduced in [6]. In this dataset creation process, the authors began with an initial satisfiable QC instance. They generated a set of $0.4 \times N$ queens positioned them on the chessboard in a manner that ensured they couldn't attack each other. Subsequently, the instance underwent various transformations, such as rotations and configurations with non-attacking queens. Ultimately, the dataset was composed of 256 QC instances for each of the test cases $N \in \{14, 18, 22, 26, 30\}$. In our research study, we conducted evaluations using two separate streams of data. The training stream was generated exclusively from the first 150 QC instances, while the testing stream was generated from the remaining 106 QC instances.

B. Results

The neural network architecture employed in our experimental tests consists of two components: the actor neural network and the critic neural network. The actor neural network comprises two hidden layers with 256 and 128 neurons. Similarly, the critic neural network also consists of two hidden layers, with 256 and 128 neurons, enabling them to approximate the state-action value function accurately. To ensure effective learning and long-term rewards, we set the discount factor to 0.99, which encourages the agent to consider future rewards when making decisions. Additionally, the number of neighbors in the k-nearest neighbors (KNN) algorithm was set to 10 [3].

[3]Implementation and dataset used in evaluation can be found at : https://anonymous.4open.science/r/CLOSER-B5F5/README.md

TABLE I
AVERAGE NUMBER OF CACHED CONSTRAINTS USED BY THE SOLVER

Methods	Datasets				
	Enc-14	Enc-18	Enc-22	Enc-26	Enc-30
RESTART	1.0997	1.3227	0.3787	0.241	0.0555
CLOSER	**1.1417**	**10.7907**	**9.8203**	**8.4603**	**7.5563**

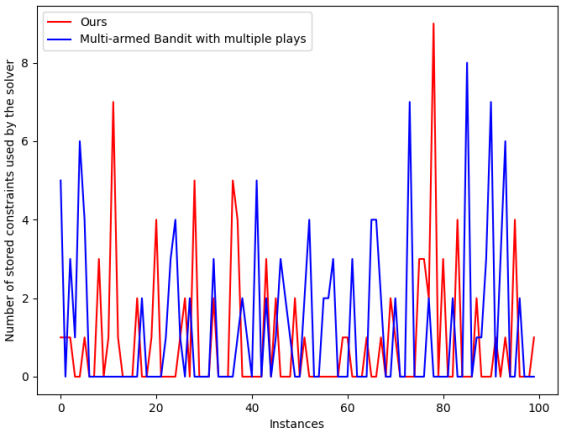

Fig. 3. Plot of cache performance for Enc-14

In this section, we discuss the evaluations conducted in our research and compare the obtained results with those of RESTART [6]. The primary focus of our evaluation is to measure the number of constraints stored in the cache, which are subsequently used by the solver. The size of the cache is set to 100 for both approaches. The average number of constraints stored in the cache, for both approaches is presented in Table I. The CLOSER's performances are far superior to those of the RESTART in all test cases. Our results consistently show competitive cache utilization compared to RESTART. However, we observed some degradation in performance for complex problems. In Fig. 3, we depict the plots representing the outcomes of the simplest test case: Enc-14. The graph illustrates that both approaches yield comparable results, and it is evident that the count of cached constraints utilized by the solver fluctuates for each approach. In Fig. 4, we showcase the outcomes achieved for the most challenging test scenario, Enc-30, with a fixed value of $N = 30$. The results indicate that CLOSER outperforms RESTART, yielding superior results across almost all instances within the data stream. More precisely, in our research methodology, the solver effectively employs over 20 constraints from the cache of size 100 for certain instances. On the contrary, the competing approach struggles to handle the same situations, as it only utilizes fewer than 3 constraints out of the total 100. This strikingly underscores the remarkable adaptability of our proposed method in effectively addressing problems of diverse complexities, . signifying its potential to provide robust solutions in constraint-solving scenarios.

In summary, our approach demonstrates remarkable results in terms of average cache memory usage, compared with

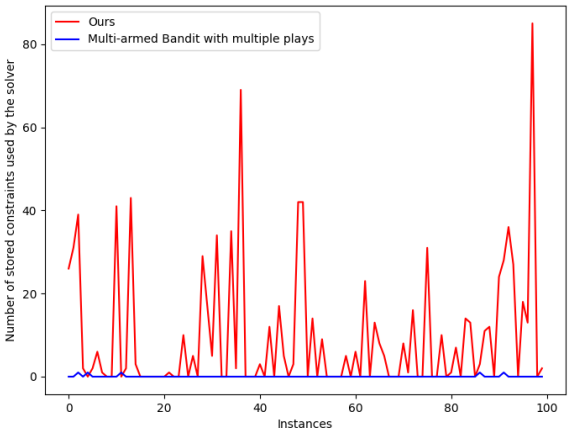

Fig. 4. Plot of cache performance for Enc-30

RESTART. Nevertheless, the observed variability in cache usage across different trials emphasizes the need for further investigation and fine-tuning. We are confident that our research provides a solid foundation for potential enhancements in ASP solver efficiency, contributing to the advancement of constraint-based problem-solving techniques.

VIII. Conclusion

We proposed in this paper CLOSER, a novel approach to address the challenge of managing constraints in stream reasoning engines. By adapting the CDCL algorithm to handle data streams and integrating it in a DRL architecture and by using cache management methodologies, we have developed an efficient framework for constraint-solving scenarios in ASP-based stream reasoning systems. The experiment conducted to compare our proposed approach with recent work in the literature on average cache utilization clearly demonstrated the advantages of our approach. The results indicate that our solution outperforms the existing method, particularly when dealing with diverse problem complexities. This showcases the potential of our approach to offer robust solutions in various constraint-solving scenarios. While we have achieved remarkable results in terms of average cache utilization, it is important to acknowledge the observed variability in cache usage across different trials. This variability emphasizes the need for further investigation and fine-tuning to optimize the performance of our approach. The presented work is the beginning of an ongoing journey to improve and refine our approach. There is still much room for further exploration.

References

[1] Silva, JP Marques, and Karem A. Sakallah. "GRASP-a new search algorithm for satisfiability." Proceedings of International Conference on Computer Aided Design. IEEE, 1996.

[2] Eén, Niklas, and Niklas Sörensson. "An extensible SAT-solver." International conference on theory and applications of satisfiability testing. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003.

[3] Audemard, Gilles, and Laurent Simon. "Predicting learnt clauses quality in modern SAT solvers." Twenty-first international joint conference on artificial intelligence. 2009.

[4] Audemard, Gilles, et al. "On freezing and reactivating learnt clauses." Theory and Applications of Satisfiability Testing-SAT 2011: 14th International Conference, SAT 2011, Ann Arbor, MI, USA, June 19-22, 2011. Proceedings 14. Springer Berlin Heidelberg, 2011.

[5] Vaezipoor, Pashootan, et al. "Learning clause deletion heuristics with reinforcement learning." 5th Conference on Artificial Intelligence and Theorem Proving. 2020.

[6] Dodaro, Carmine, Thomas Eiter, Paul Ogris, and Konstantin Schekotihin. "Managing caching strategies for stream reasoning with reinforcement learning." Theory and Practice of Logic Programming 20, no. 5 (2020): 625-640.

[7] Anantharam, Venkatachalam, Pravin Varaiya, and Jean Walrand. "Asymptotically efficient allocation rules for the multiarmed bandit problem with multiple plays-part i: Iid rewards." IEEE Transactions on Automatic Control 32, no. 11 (1987): 968-976.

[8] Beck, Harald, Minh Dao-Tran, and Thomas Eiter. "LARS: A logic-based framework for analytic reasoning over streams." Artificial Intelligence 261 (2018): 16-70.

[9] Della Valle, Emanuele, Stefano Ceri, Davide Francesco Barbieri, Daniele Braga, and Alessandro Campi. "A first step towards stream reasoning." In Future Internet–FIS 2008: First Future Internet Symposium, FIS 2008 Vienna, Austria, September 29-30, 2008 Revised Selected Papers 1, pp. 72-81. Springer Berlin Heidelberg, 2009.

[10] Eiter, Thomas, Paul Ogris, and Konstantin Schekotihin. "A distributed approach to LARS stream reasoning (system paper)." Theory and Practice of Logic Programming 19, no. 5-6 (2019): 974-989.

[11] Beck, Harald, Thomas Eiter, and Christian Folie. "Ticker: A system for incremental ASP-based stream reasoning." Theory and Practice of Logic Programming 17, no. 5-6 (2017): 744-763.

[12] Gelfond, Michael, and Vladimir Lifschitz. "The stable model semantics for logic programming." In ICLP/SLP, vol. 88, pp. 1070-1080. 1988.

[13] Kaufmann, Benjamin, Nicola Leone, Simona Perri, and Torsten Schaub. "Grounding and solving in answer set programming." AI magazine 37, no. 3 (2016): 25-32.

[14] Dulac-Arnold, Gabriel, Richard Evans, Hado van Hasselt, Peter Sunehag, Timothy Lillicrap, Jonathan Hunt, Timothy Mann, Theophane Weber, Thomas Degris, and Ben Coppin. "Deep reinforcement learning in large discrete action spaces." arXiv preprint arXiv:1512.07679 (2015).

[15] Gomes, Carla P., Bart Selman, and Henry Kautz. "Boosting combinatorial search through randomization." AAAI/IAAI 98 (1998): 431-437.

[16] Huang, Jinbo. "The Effect of Restarts on the Efficiency of Clause Learning." In IJCAI, vol. 7, pp. 2318-2323. 2007.

[17] Audemard, Gilles, and Laurent Simon. "On the glucose SAT solver." International Journal on Artificial Intelligence Tools 27, no. 01 (2018): 1840001.

[18] Alviano, Mario, Carmine Dodaro, Nicola Leone, and Francesco Ricca. "Advances in WASP." In Logic Programming and Nonmonotonic Reasoning: 13th International Conference, LPNMR 2015, Lexington, KY, USA, September 27-30, 2015. Proceedings 13, pp. 40-54. Springer International Publishing, 2015.

[19] Alviano, Mario, Carmine Dodaro, Wolfgang Faber, Nicola Leone, and Francesco Ricca. "WASP: A native ASP solver based on constraint learning." In Logic Programming and Nonmonotonic Reasoning: 12th International Conference, LPNMR 2013, Corunna, Spain, September 15-19, 2013. Proceedings 12, pp. 54-66. Springer Berlin Heidelberg, 2013.

[20] Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. MIT press, 2018.

[21] Silva, JP Marques, and Karem A. Sakallah. "Conflict analysis in search algorithms for satisfiability." In Proceedings Eighth IEEE International Conference on Tools with Artificial Intelligence, pp. 467-469. IEEE, 1996.

[22] Gent, Ian P., Christopher Jefferson, and Peter Nightingale. "Complexity of n-queens completion." Journal of Artificial Intelligence Research 59 (2017): 815-848.

[23] Lillicrap, Timothy P., Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. "Continuous control with deep reinforcement learning." arXiv preprint arXiv:1509.02971 (2015).

[24] Bazoobandi, Hamid R and Beck, Harald and Urbani, Jacopo. "Expressive stream reasoning with laser" International Semantic Web Conference, pp.87-103. Springer, 2017.