Stealthy Targeted Data Poisoning Attack on Knowledge Graphs

Prithu Banerjee † Lingyang Chu ‡ Yong Zhang ‡ Laks V.S. Lakshmanan † Lanjun Wang ‡
University of British Columbia, {prithu,laks}@cs.ubc.ca †
Huawei Canada Technologies Co. Ltd., {lingyang.chu1,lanjun.wang,yong.zhang3}@huawei.com ‡

***Abstract*—A host of different KG embedding techniques have emerged recently and have been empirically shown to be very effective in accurately predicting missing facts in a KG, thus improving its coverage and quality. Unfortunately, embedding techniques can fall prey to adversarial data poisoning attack. In this form of attack, facts may be added to or deleted from a KG, called performing perturbations, that results in the manipulation of the plausibility of target facts in a KG. While recent works confirm this intuition, the attacks considered there ignore the risk of exposure. Intuitively, an attack is of limited value if it is highly likely to be caught, i.e., exposed. To address this, we introduce a notion of the exposure risk and propose a novel problem of attacking a KG by means of perturbations where the goal is to maximize the manipulation of the target fact's plausibility while keeping the risk of exposure under a given budget. We design a deep reinforcement learning-based framework, called RATA, that learns to use low-risk perturbations without compromising on the performance, i.e., manipulation of target fact plausibility. We test the performance of RATA against recently proposed strategies for KG attacks, on two different benchmark datasets and on different kinds of target facts. Our experiments show that RATA achieves state-of-the-art performance even while using a fraction of the risk.**

I. Introduction

An integral step in growing and maintaining a large KG is to employ a *knowledge completion model* which when trained on a set of *true* positive and *true* negative facts, can predict other triples that are highly likely to be true and therefore can be used to augment the KG. These training facts are often obtained by means such as crowd sourcing [1], [2], automated text mining [3], [4], etc.

Unfortunately, as has been demonstrated recently [5], [6], knowledge graphs are susceptible to *data poisoning attacks*, whereby an attacker can maliciously add facts, whose inclusion in a KG may have the detrimental effect of manipulating the plausibility score of other facts: facts with a low score (and thus unlikely to be true) can have their score bumped up and facts with a high score (thus highly likely true) can have their scored lowered. A high quality KG is the cornerstone for downstream applications that leverage the KG. Clearly, such attacks can erode the quality of the KG and may have a negative impact on downstream applications.

Facts that are added to the training set by a malicious attacker, for manipulating the scores of facts and thus the quality of a KG, are called *perturbations*. In this paper, we are specifically interested in *targeted data poisoning attack* [5]. In this attack, a specific target fact is chosen, and then perturbations are added to the KG to *manipulate* the score of the target fact to a low (or high) plausibility score. In this way, an attacker can turn a true fact into a fake fact (or vice versa), and she can further exploit this loophole to launch serious attacks on downstream applications, such as hiding accurate recommendations from recommendation systems, and concealing valid answers in information searching systems and question answering systems.

Existing targeted data poisoning attacks [6], [5] *ignore the risk of getting exposed during the attack.* Being aware of and mitigating the risk of exposure is an important aspect of a good quality attack, and it has been considered for data poisoning attacks in other domains [7], [8]. However, the previous works on knowledge graph attacks ignore exposure risk completely. In contrast, in this paper we aim to find a *stealthy attack* that is hard to detect and hence represents a much greater threat to the robustness of knowledge graphs.

Is it possible to conduct a stealthy targeted data poisoning attack, while keeping the exposure risk low? In this paper, we answer this affirmatively. While we formalize the notion of risk in Section II, intuitively speaking, perturbations that add facts with a higher plausibility score into a knowledge graph are less risky than those that add facts with a lower plausibility score, as determined by the completion model. By exploiting the high non-linearity of the knowledge graph structure, we find that it is possible to achieve a good attack performance while selecting low risk perturbations.

To the best of our knowledge, we are the first to study risk aware attacks on knowledge graphs. In Section II, we formally introduce the novel problem of targeted KG data poisoning attack under an exposure risk budget, as a constrained optimization problem.

We then show that finding a sequence of perturbations can be naturally modeled as Markov Decision Process (MDP). We propose a solution framework based on reinforcement learning (RL), leveraging the fact that RL algorithms using policies such as Q-learning can efficiently solve MDPs [9]. Also to preserve the well-known non-linearity of the KG, we use a deep non-linear neural network to learn the Q functions. Lastly, due to the high number of entities and relations in a KG, a naive approach of using an embedding of the entire collection of facts as a state and a single perturbation as an action, quickly becomes practically infeasible. To address this bottleneck, we use a hierarchical Q-learning approach that decomposes a single action into a three-step decision making process. Once trained, our model, Risk called Aware

Targeted Attacker (RATA for short), can perform effective attacks even under exposure risk constraints, unlike the risk agnostic baselines.

To summarize, the major contributions of our paper are as follows: (1) We introduce the novel problem of finding targeted knowledge graph attack under an exposure risk constraint (Section II). (2) We develop a principled framework for attacks, based on hierarchical deep Q-learning, by making use of the natural match between reinforcement learning and MDP. Our attack approach, called RATA, succeeds in exploiting the non-linear structure of the KG efficiently (Section III). (3) Our experiments on large real world benchmark KG datasets, demonstrate that RATA achieves good attack performance, while staying within a given exposure risk budget (Section IV).

II. Problem Formulation

In this section, we first define the notations that we use in the paper, and then formulate our problem.

A knowledge graph G is a set of facts F represented as an unweighted directed graph, where each vertex represents an entity, and a directed edge connecting two vertices represents the relationship between the two entities. We refer to a triple $f = (h, r, t)$, where h and t are the head and tail entity, respectively, and r is the relationship from h to t, as a *fact*. Two triples $f_1 = (h_1, r_1, t_1)$ and $f_2 = (h_2, r_2, t_2)$ are equal iff they are equal component-wise. Clearly, there is a one-to-one correspondence between a knowledge graph G and the set of facts F that it represents. We thus, use these notions interchangeably.

Denote by $\mathcal{E}$ the universe of all entities, by $\mathcal{R}$ the universe of all relations, and by $P = \mathcal{E} \times \mathcal{R} \times \mathcal{E}$ the set of all possible triples. Clearly, every fact in F is a triple in P, so we have $F \subseteq P$.

There is some unknown subset $F' : F \subset F' \subset P$ of all "true" facts, not all of which are present in the knowledge graph. A *completion model* is trained on all facts in F and then the trained model is used to identify whether a missing fact $f \in P \setminus F$ could be in F'. Based on this, a decision is made as to whether f should be added to the KG, and the KG could grow [10]. Specifically, the completion model first computes the embeddings of all the entities and relationships in F, and then uses these embeddings to predict the *plausibility score* of every triple $f \in P$. A higher (lower) plausibility score means a triple is more (resp., less) likely to be true. For triples in $P \setminus F$, only the ones with high plausibility scores are regarded as facts and added into F. We denote by $s(\cdot) : P \to \mathbb{R}$ the *plausibility score function* that maps a triple $f \in P$ to its plausibility score $s(f)$. An accurate score value $s(f)$ is critical for accurately identifying whether a missing fact $f \in P \setminus F$ is true and should be included in F.

The plausibility score function is uniquely determined by a completion model trained on F, and we denote it by $s_F(\cdot)$. If F is deliberately perturbed by adding some triples carefully, the scoring function can be altered to manipulate the plausibility score of facts in $P \setminus F$. This could prevent a missing fact from being added into F, and attackers can exploit this loophole to intentionally hide valuable facts from being added into a knowledge graph [11]. Such a manipulation of the plausibility score of a given target fact is called a *targeted data poisoning attack* [5], which usually consists of a sequence of perturbations. Here, a *perturbation* is the operation of adding a selected triple $d \in P \setminus F$ into F. Two perturbations are said to be the same if they are adding the same triple $d \in P \setminus F$ into F. Since each triple $d \in P \setminus F$ uniquely corresponds to a perturbation, a *sequence of perturbations*, denoted by D, is uniquely determined by a partial permutation of the triples in $P \setminus F$.

Data poisoning attack papers on normal graph have shown that removing training samples is very easy to detect as an attack [8], [12]. Thus we do not consider perturbations corresponding to removing facts from F.

The target is chosen to be one of the facts which is not yet part of the existing set of facts F. Denote by $f^* \in P \setminus F$, a target fact to attack, by D, a set of perturbations where $f^* \notin D$, by $F \cup D$, the perturbed set of facts generated by perturbing F with D. Let $s_{F \cup D}(\cdot)$ be the manipulated plausibility score function determined by the same completion model, but trained on the perturbed training data $F \cup D$. Without loss of generality and following previous work [5], we only focus on reducing the plausibility score of f^* after adding D as perturbation, given by $s_F(f^*) - s_{F \cup D}(f^*)$.

The goal of our work is to conduct a *stealthy targeted data poisoning attack* that significantly lowers the plausibility score of a target fact while keeping the exposure risk low. Depending on the triple $d \in P$ chosen to be added into F, different perturbations have different risks of being detected. We model the risk of perturbations by an *exposure risk function* $\rho(\cdot) : P \to \mathbb{R}$ that maps a triple $d \in P \setminus F$ to the risk score $\rho(d)$ of adding d into F. A higher value of $\rho(d)$ means the perturbation d is easier to detect, i.e., the attacker will be exposed more easily. For a sequence of perturbations denoted by D, we extend the risk function to evaluate the exposure risk of perturbing F with D, by $\rho(D)$.

Next, we formally state our Risk Constrained Targeted Data Poisoning (RC-TDP) problem.

Problem 1 (Risk Constrained Targeted Data Poisoning Problem)**.** *Given a knowledge graph in the form of set of facts F, a score function $s(\cdot)$ given by a completion model, a target fact $f^* \in P \setminus F$, an exposure risk function $\rho(\cdot)$, and a risk budget ϱ, the problem of risk constrained targeted data poisoning is to find a sequence of perturbations D, where $f^* \notin D$, such that D maximally manipulates the plausibility score of f^*, while keeping the exposure risk bounded, i.e., $\rho(D) \leq \varrho$. More precisely, the problem is to find:*

$$\begin{aligned} &\underset{D \subseteq P \setminus F \wedge f^* \notin D}{\arg\max} \; s_F(f^*) - s_{F \cup D}(f^*) \\ &s.t. \quad \rho(D) \leq \varrho \end{aligned} \tag{1}$$

III. Knowledge Graph Attack

A general KG attack environment under a risk budget constraint is presented in Section III-A. specific adjustments

we make to adapt the environment for our problem is in Section III-B and our training algorithm is in III-C.

A. General Attack environment

State. $\mathcal{S}$ is the set of all states. At a given timestep l, the corresponding state $g_l \in \mathcal{S}$ corresponds to the set of facts in KG at time l, denoted by F_l, and the remaining budget available, ϱ_l. When $l = 1$, $\varrho_l = \varrho$ and $F_1 = F$, the initial set of facts contained in the original KG, prior to any attack. At every timestep l, a new fact is f_l added to F_l, changing F_l to $F_{l+1} = F_l \cup \{f_l\}$. Meanwhile, the remaining risk budget is also updated as follows: $\varrho_{l+1} = \varrho_l \ominus \rho(f_l)$, where $\ominus$ is an operation that updates the left-over risk budget, taking into account the risk of the newly added fact f_l. In the most general case, ϱ_{l+1} is incrementally computed and needs to be revised based on $\rho(F \cup F_{l-1} \cup \{f_l\})$. However if the risk function is simply the sum of the risks of individual perturbations, then $\ominus$ is just subtraction, i.e., $\varrho_{l+1} = \varrho_l - \rho(f_l)$. Finally, we model a state g_l as the set of embeddings of the facts in F_l along-with the available budget ϱ_l at that state.

Action. An action at time l, denoted by a_l for $1 \leq l < T$, is to add a fact $f_l \in P \setminus F_l$ s.t. $\rho(f_l) \leq \varrho \ominus \varrho_l$. Since a fact is a triple of head entity, tail entity and their relationship, an action a_l involves choosing a head entity, a tail entity, and a relation. Thus the search complexity is $O(\mathcal{E} \times \mathcal{R} \times \mathcal{E})$, which is prohibitively high. Section III-B uses a decomposition strategy to reduce it to $O(\mathcal{E} + \mathcal{R} + \mathcal{E})$.

$(g_1, a_1^{(h)}, a_1^{(r)}, a_1^{(t)}, g_2, ..., g_{T-1}, a_{T-1}^{(h)}, a_{T-1}^{(r)}, a_{T-1}^{(t)}, g_T)$, is a path where g_T is terminal state. Sampling a path is also called running an episode.

Termination of an episode. Episode's termination condition is governed by the budget ϱ. An MDP path terminates when the remaining budget ϱ_l at any state l is not enough to select any further action. Further we enforce that our MDP is a finite horizon MDP, hence if the budget does not exhaust after m steps, we terminate the process. m is set as a hyper-parameter which denotes the maximum possible length of a path.

Reward. First, our agent should reduce the plausibility of the target fact as much as possible. In addition, we also want to discourage the agent to choose perturbations that are highly risky. One way to do that is to encourage the agent to choose longer episodes, because for a given budget, in a longer episode, each perturbation has to have a lower risk than in an episode of a shorter length. Also notice that the manipulation score produced by the final set of perturbations at the end of an episode is critical to our objective. The intermediate drop in the score is irrelevant. At any intermediate step, even if the manipulation score is low, we should not discard the exploration because by adding more perturbations the final drop could increase. To achieve this, we set a reward function which is fully realized only after adding all the perturbations. Reward r_l for any $l \leq T-1$ is 0, the reward is non-zero for the last action a_{T-1}. For $l = T$, one factor of the reward is based on the manipulation in the plausibility score of the target fact, i.e., $s_F(f^*) - s_{F_T}(f^*)$, where F and F_T are the sets of facts in states g_1 and g_T respectively. The higher the final manipulation, more the reward. The other factor is based on the length of the episode, i.e., $1 - \frac{1}{T} + \epsilon$, where $\epsilon < 1$ is a hyperparameter. For a higher value of T, the reward will be higher. Combining the two, the final reward is: $r_T = ((1 - \frac{1}{T}) + \epsilon) \cdot (s_F(f^*) - s_{F_T}(f^*))$.

Lastly we set the discount factor $\gamma = 1$. Hence the final reward is propagated back to every intermediate step. Notice that since our MDP has a finite horizon, it is feasible to propagate the reward information back in this way.

B. Optimization policy under the hierarchical decomposition

For discrete optimization problems with a finite horizon, Q-learning to learn the paths for MDPs is shown to outperform other policy optimization methods like Advantage Actor Critic (AAC) [9]. Thus, we propose our optimization model using Q-learning.

The expected reward for taking an action a_l at a given state g_l, is given by the Q-function $Q(s_l, a_l)$. The function values are stored in the form of a table called Q-table. During training, the agent aims to accurately learn the entries of the Q-tables. The Q-tables thus store the expected reward for a given (state, action) pair which is a combination of the transition probability of states and the deterministic rewards in states.

To compute the optimal expected reward for an action a_l at a given state g_l, Q-learning directly fits the Bellman optimality equation Thus the optimal action at any given state g_l can then be chosen by the simple greedy policy:

$a^* = \pi(g_l) = \arg\max_{a_l} Q^*(g_l, a_l)$.

In our case, we set the granularity of an action a_l to choosing one fact. This action is decomposed into three parts $a_l^{(h)}, a_l^{(r)}, a_l^{(t)}$, i.e., choosing the head entity, relation, and tail entity. It is hard to directly design the function $Q(g_l, a_l^{(h)}, a_l^{(r)}, a_l^{(t)})$ and apply one policy to select each of the three individual actions efficiently. Hence we learn three different Q functions for each of them as described below.

Three level decomposition. To address the above mentioned problem, we integrate three deep Q networks (DQN), $Q = \{Q^{(h)}, Q^{(r)}, Q^{(t)}\}$, to model Q values for each of the decomposed actions. The first function $Q^{(h)}$ is responsible for guiding the policy to select the head entity $a_l^{(h)}$. Depending on the selected $a_l^{(h)}$, the second DQN $Q^{(r)}$ learns a policy to select relation $a_l^{(r)}$. Finally given head entity and relation, the third DQN $Q^{(t)}$ learns the policy to select the tail entity $a_l^{(t)}$.

We now discuss our greedy policy on parametrized Q function. Let i be one of $\{h, r, t\}$. Then $\theta^{(i)} = \{W_1^{(i)}, W_2^{(i)}\}$ denote weights to be trained for the i-th DQN of the corresponding Q function. the greedy policies for selecting the corresponding actions are then given by the following equations

$$a_l^{(h)} = \pi(g_l) = \arg\max_{h \in \mathcal{E}} Q^{(h)}(g_l, h; \theta^{(h)}) \tag{2}$$

$$a_l^{(r)} = \pi(g_l, a_l^{(h)}) = \arg\max_{r \in \mathcal{R}} Q^{(r)}(g_l, a_l^{(h)}, r; \theta^{(r)}) \tag{3}$$

$$a_l^{(t)} = \pi(g_l, a_l^{(h)}, a_l^{(r)}) = \arg\max_{t \in \mathcal{E}} Q^{(t)}(g_l, a_l^{(h)}, a_l^{(r)}, t; \theta^{(t)}) \tag{4}$$

Recall that for a given target fact, our goal is to find a sequence of perturbations D. After decomposition, the output

sequence is basically $0 \leq l < T, (a_l^{(h)}, a_l^{(r)}, a_l^{(t)})$. In what follows, we first describe the general training method we use to learn the parameters of the Q functions. Once the parameters are learnt, they are used to output D for a test target fact.

C. Training algorithm

In this section, we describe our training algorithm in detail. The goal of training in the Q-learning framework is to learn the Q-function, i.e., essentially learning the entries in the Q-tables. Since our Q-functions are deep neural networks parameterized by their corresponding weights, our training objective is to learn these weights using exploration of the RL agent. We assume the score function $s(\cdot)$ and risk function $\rho(\cdot)$ are accessible to the attacker during the training, via query oracles. The peudocode of our training algorithm for a given target fact is shown in Algorithm 1.

Algorithm 1 $RATA\_TRAIN(F, f^*, s, \rho, \varrho)$

1: $e \leftarrow$ Number of episodes
2: $m \leftarrow$ Maximum length of an episode
3: Initialize Q functions with random parameters
4: **while** $episode \leq e$ **do**
5: $\quad l \leftarrow 1$; $F_l = F$; $\varrho_l = \varrho$
6: $\quad$ **while** $l \leq m$ **do**
7: $\quad\quad$ Select $a_l^{(h)}, a_l^{(r)}$ and $a_l^{(t)}$ using Eq 2, 3 and 4
8: $\quad\quad$ $k_l \leftarrow \rho(a_l^{(h)}, a_l^{(r)}, a_l^{(t)})$
9: $\quad\quad$ $\varrho_l = \varrho_l \ominus k_l$
10: $\quad\quad$ **if** $\varrho_l < 0$ **then**
11: $\quad\quad\quad$ break
12: $\quad\quad$ **end if**
13: $\quad\quad$ $F_l \leftarrow F \cup \{(a_l^{(h)}, a_l^{(r)}, a_l^{(t)})\}$
14: $\quad$ **end while**
15: $\quad$ Compute reward $r = ((1 - \frac{1}{l}) + \epsilon) \cdot (s_F(f^*) - s_{F_l}(f^*))$
16: $\quad$ Update the transition probabilities and the weight parameters of Q using a replay buffer
17: **end while**
18: Return the learnt parameters of Q

Similar to other finite horizon RLs, it first sets two hyperparameters, namely, the number of episodes e and the maximum length of each episode m. After they are initialized, the Q-function parameters are randomly initialized (see Lines 1-3). Then in the beginning of each episode, the available budget of that episode is set as the budget constraint provided as input (Line 5). Each episode is run either till the budget is exhausted (Line 10) or if budget permits, till the maximum length of an episode set by the hyperparameter e (Line 4).

At every step of the episode, the best triple $(a_l^{(h)}, a_l^{(r)}, a_l^{(t)})$ is selected based on the current parameters of Q. The risk of the triple is "subtracted" from the available budget for the episode. Finally the triple is added a perturbation if the budget permits (Lines 6–13).

At the end of an episode, two key updates related to Q table are performed. First, based on the states that are visited in the episode, the transition probabilities are updated. The probability of visiting g_{l+1} from g_l is updated as the proportion of the times g_{l+1} is visited out of those that g_l is visited.

Secondly, the final reward is observed and accordingly the parameters of the Q function are updated. The goal of this update is to learn the set of parameters that yields the best reward. However, the key challenge is to keep the training stable while avoiding over-fitting. Thus, updating these parameters for every episode tends to make the learning unstable. To address this, we use a technique called *memory replay buffer*, frequently used in RL.

IV. Experiments

RATA is evaluated against the baselines on two benchmark datasets. In this section we describe the experimental set up and analyze the performance of the compared algorithms.

Baselines. There is no previous work that can perform the KG data poisoning attack problem under a budget constraint, however there are two works that study the attack without considering a exposure risk constraint: DirectAdd [5] and CRIAGE [6]. CRIAGE uses an *inverter* for optimizing the search for the perturbations. The inverter works only for multiplicative KG completion models such as DistMult [13]. Hence we compare CRIAGE only against DistMult. Zhang et al. [5] proposed an attack algorithm that works for additive models as well.

KG completion models. As stated, for the experiments involving CRAIGE we use a mulitplicative completion model DistMult [13]. For the experiments of DirectAdd, besides DistMult, we choose another two state-of-the-art completion models – TransE [14] and TransR [10]. We use OpenKE implementation of all the completion models [15].

Knowledge graphs. We test our algorithms on two most commonly used KG benchmark datasets, namely, FB15k and WN18. FB15k is mined from Freebase, a collaborative knowledge base consisting of a large number of real-world facts. WN18 is mined from WordNet, which is a large lexical knowledge graph. Both the graphs were introduced in [14]. The authors also fixed the training and test sets that have been traditionally used for benchmarking.

Target facts. Target facts are chosen from the test sets. We perform our experiments on two different types of target facts. First all the facts are ranked based on their plausibility scores generated by the KG completion model. Our first type of target facts are highly plausible facts. This set is created by choosing 100 random facts from the top 500 ranked test facts. Similarly the second set of low plausibility facts is formed by choosing target facts from the bottom 500 test facts. Note that for an attacker it is more beneficial to attack a highly plausible target fact. If an attacker can effectively corrupt highly plausible facts, more damage is inflicted on the users of the KG.

Risk. As we argued in the introduction, a simple defense system, a KG moderator can have, is to use the existing KG completion model to judge the exposure risk of a perturbation. Thus for an individual perturbation fact f, we set its risk as $\rho(f) = 1 - s_F(f)$. Thus if f has a high plausibility score $(s_F(f))$, since the completion model believes it as a true fact, its exposure risk is low and vice versa.

Exposure risk of a perturbation set D is a function of its constituent individual perturbations. We consider two such functions for our experiments- (i) SUM: $\rho(D) = \sum_{f \in D} \rho(f)$, and (ii) MAX: $\rho(D) = \max_{f \in D} \rho(f)$.

Graph	Algorithm	Clean		DirectAdd		CRIAGE		RATA@70		RATA@80		RATA@90		RATA@100	
		MRR	H@10	MRR	H@10	MRR	H@10	MRR	H@10	MRR	H@10	MRR	H@10	MRR	H@10
FB15K	TransE	0.33	0.51	0.27	0.44	-	-	0.30	0.46	**0.27**	**0.43**	**0.26**	**0.42**	**0.26**	**0.42**
	TransR	0.31	0.53	0.24	0.47	-	-	0.26	0.49	0.25	**0.47**	**0.24**	**0.45**	**0.24**	**0.44**
	DistMult	0.28	0.45	0.23	0.41	0.18	0.38	0.26	0.44	**0.22**	**0.41**	**0.21**	**0.40**	**0.21**	**0.39**
WIN18	TransE	0.41	0.75	0.29	0.61	-	-	**0.29**	0.64	**0.29**	**0.60**	**0.27**	**0.58**	**0.26**	**0.57**
	TransR	0.44	0.77	0.30	0.62	-	-	0.31	0.65	**0.29**	**0.62**	**0.27**	**0.59**	**0.27**	**0.57**
	DistMult	0.47	0.78	0.34	0.64	0.27	0.59	0.37	0.68	0.32	**0.64**	**0.29**	**0.61**	**0.28**	**0.59**

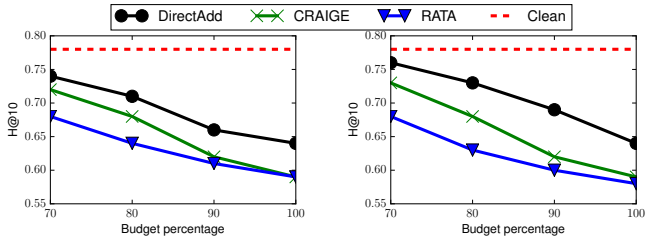
TABLE I: Performance comparison of RATA and baselines: DirectAdd and CRAIGE are the two baselines. RATA@x means RATA constrained by budget x% of the total budget used by any baseline. The risk function is **SUM** and the target facts are **highly plausible** facts. Bold entry denotes RATA's performance is at per with at least one baseline, boxed entry denotes RATA is at per with both the baselines (when applicable).

Graph	Algorithm	Clean		DirectAdd		CRIAGE		RATA@70		RATA@80		RATA@90		RATA@100	
		MRR	H@10	MRR	H@10	MRR	H@10	MRR	H@10	MRR	H@10	MRR	H@10	MRR	H@10
FB15K	TransE	0.27	0.48	0.23	0.43	-	-	0.27	0.46	0.26	0.44	**0.23**	**0.43**	**0.22**	**0.43**
	TransR	0.26	0.49	0.22	0.45	-	-	0.26	0.47	0.25	0.46	**0.22**	**0.45**	**0.21**	**0.44**
	DistMult	0.23	0.44	0.20	0.40	0.17	0.36	0.23	0.43	0.22	0.42	0.21	0.41	**0.20**	0.41
WIN18	TransE	0.35	0.61	0.26	0.49	-	-	0.32	0.56	0.28	0.53	**0.26**	0.50	**0.25**	**0.49**
	TransR	0.37	0.63	0.27	0.50	-	-	0.33	0.58	**0.27**	**0.50**	**0.26**	**0.49**	**0.25**	**0.48**
	DistMult	0.38	0.66	0.31	0.55	0.26	0.51	0.36	0.60	0.33	0.57	**0.31**	**0.55**	**0.30**	**0.53**

TABLE II: Same as Table I for **low plausibility** target facts.

Parameter settings. RATA has two specific parameters, the number of episodes, e and the maximum number of states in one episodes, m. Unless mentioned otherwise, we use $e = 100$ and $m = 50$ as default. For the baselines we use their default values as mentioned in [6] and [5]. The completion models are also set to their default parameter values as prescribed in [16].

Evaluation metric. The evaluation protocol we use is the standard one used for KG attacks [5], [6]. Given a target fact $f = (h, r, t)$, first either the head or the tail entity of the fact is removed from the KG. Then a trained completion model is used to predict the missing entity. The algorithm ranks all the entities in a descending order. Rank of the missing entity in that order is stored. Finally ranks from a completion model trained on the clean dataset and on perturbed datasets generated by different attack methods are compared. Two metrics used to do this comparison are - (i) Mean Reciprocal Rank (MRR): Mean Reciprocal Rank of the missing entities in all the targets, (ii) H@10 (Hits at 10): Fraction of times the missing entity appears in the top 10 ranked entities. Clearly under both metrics, a lower score indicates a better attack performance.



(a) SUM risk function (b) MAX risk function

Fig. 1: Performance comparison under different risk budgets

Attack performance of RATA against the baselines. Since the baselines cannot ingest an exposure risk constraint, in this set of experiments we run the baseline without any budget constraint and see how RATA performs in comparison when the budget constraint is imposed. We compute the total available budget as the risk incurred by the perturbation set produced by any of the baseline produces. Next we constrain RATA to use a fraction of that risk budget. Particularly in the following, RATA@x denotes that RATA is allowed to use x percent of the total budget used by the baseline. We compare the various attack algorithms under the SUM risk function, performance is similar for MAX and hence omitted.

We present the result of our experiments using this risk function for high plausibility and low plausibility target facts in Table I and II respectively, where the column *Clean* refers to the scores before adding any perturbation. After the attack, once the perturbation set is added, the scores drop, and *a lower score denotes a better attack performance.*

In Table I, the target facts are highly plausible facts, i.e., the facts for which the completion model produces a high plausibility score. As can be seen, RATA starts matching the attack performance of the baselines using about 80% of the budget. This is particularly true in comparison to a generic attack, DirectAdd. Since CRAIGE is a specialized attack against DistMult, outperforming CRAIGE is more difficult. Further on a denser graph, where there are more facts to choose from, the performance of RATA improves. Thus for WIN18, RATA starts matching even the attack performance of CRAIGE at a relatively lower budget.

For the low plausibility targets in Table II, the attack performance of every algorithm is relatively worse, as the drop in the score is less. The reason is these facts already have low plausibility scores and the other related facts to these facts also have low plausibility. Remember RATA aims to find low risk yet related facts as perturbations for a given target fact. Since the related facts of a low plausibility target also have low plausibility, using them as peturbations increases the risks. Still, RATA's performance is comparable to DirectAdd

for slightly higher budgets, i.e., 90%.

One may wonder at this point if it possible to extend the baselines to be risk budget aware and see how they perform compared to RATA. We answer this question in the following.

Attack performance of baselines under risk constraint. It is non-trivial to train the baselines to be budget aware from scratch. Instead, we take perturbation sets produced by the baseline algorithms under no budget constraint and sort the constituent facts based on their exposure risks. Then we keep discarding the high risk facts till the given budget constraint is satisfied. This ensures that the fewest facts from the perturbation sets obtained by the baselines need to be removed, to meet the risk budget. We choose DisMult as our target completion model since both the baselines can work on DisMult. The target facts are high plausibility facts. Our evaluation metric is H@10 and the graph used is Win18. Figures 1 (a) and (b) show the results corresponding to risk functions SUM and MAX respectively. Clearly under lower budget, the performance of all three methods deteriorates. However the baselines deteriorate much faster, than RATA. This also illustrates the importance of being risk aware during training.

V. Related Work and Future work

Adversarial attack on graph data has been studied recently for various application context such as graph clustering [17], spectral clustering [18] and link prediction [12]. [8] considered the risk aware adversarial attack on graphs. There have been recent advancements on data poisoning attacks on graphs as well. [11] first studied poisoning attacks on neural network for attributed graphs. [12] has shown reinforcement learning algorithms can be effectively used for node injection attacks on graph. [19] proposed data poisoning attack against factorization-based embedding methods on homogeneous graphs. In targeted data poisoning attack on graph data, [20] performed unnoticeable perturbations using iterative gradient methods, to hide targeted individuals from being detected by deep graph community detection models.

There are several key differences between graphs and knowledge graphs because of which the above mentioned attack methods cannot be directly applied on KGs. Firstly, graph algorithms primarily focus on exploiting the structural similarity or homophily of the nodes in the graph. In a KG, the structure alone is not significant, instead the relational similarity between entities and relations plays a more crucial role [21]. Secondly, the end goal of an attacker on the graph is to misclassify nodes or community-subgraphs based on their connectivity and structures [11]. In KG, the goal is manipulate the performance of the KG completion models, which relies on predicting the plausibility of a missing fact [22].

The two most closely related works to the problem we studied, are our baselines, [5] and [6]. [5] proposed a targeted data poisoning attack for general knowledge graph embeddings. [6] investigated robustness of multiplicative models such as DisMult [13] for (head, relation, entity)-triple prediction task. However, as we discussed, none of these two works considered the exposure risk while performing their attacks.

The attack considered in the paper is a targeted, black box attack. A natural future direction is to study whether it is possible to learn a general attack model which is not specific to a target. Secondly black box attacks are inferior in performance to white box attacks. Hence studying effective white box attack on knowledge graphs (while being risk conscious) remains an important challenge. Finally, defense against knowledge graph attacks has not been studied yet, and we hope that defense against RATA could lay a foundation.

References

[1] "DBpedia knowledge graph," http://dbpedia.org.

[2] "Wikidata knowledge base," http://wikidata.org.

[3] A. Fader, S. Soderland, and O. Etzioni, "Identifying relations for open information extraction," in *EMNLP*, 2011.

[4] M. Schmitz, R. Bart, S. Soderland, O. Etzioni *et al.*, "Open language learning for information extraction," in *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. Association for Computational Linguistics, 2012, pp. 523–534.

[5] H. Zhang *et al.*, "Data poisoning attack against knowledge graph embedding," in *IJCAI*, 2019.

[6] P. Pezeshkpour, Y. Tian, and S. Singh, "Investigating robustness and interpretability of link prediction via adversarial modifications," *arXiv preprint arXiv:1905.00563*, 2019.

[7] A. Shafahi, W. R. Huang, M. Najibi, O. Suciu, C. Studer, T. Dumitras, and T. Goldstein, "Poison frogs! targeted clean-label poisoning attacks on neural networks," in *Advances in Neural Information Processing Systems*, 2018, pp. 6103–6113.

[8] H. Dai *et al.*, "Adversarial attack on graph structured data," in *ICML*, 2018.

[9] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, 2016, pp. 1928–1937.

[10] Y. Lin *et al.*, "Learning entity and relation embeddings for knowledge graph completion," in *AAAI*, 2015.

[11] D. Zügner, A. Akbarnejad, and S. Günnemann, "Adversarial attacks on neural networks for graph data," in *KDD*, 2018.

[12] Y. Sun *et al.*, "Node injection attacks on graphs via reinforcement learning," in *WWW*, 2020.

[13] B. Yang *et al.*, "Embedding entities and relations for learning and inference in knowledge bases," *arXiv preprint arXiv:1412.6575*, 2014.

[14] A. Bordes *et al.*, "Translating embeddings for modeling multi-relational data," in *NeurIPS*, 2013.

[15] X. Han, S. Cao *et al.*, "Openke: An open toolkit for knowledge embedding," in *EMNLP*, 2018.

[16] "OpenKE KG embedding," https://github.com/thunlp/OpenKE.

[17] Y. Chen *et al.*, "Practical attacks against graph-based clustering," in *SIGSAC*, 2017.

[18] A. Bojchevski, Y. Matkovic, and S. Günnemann, "Robust spectral clustering for noisy data: Modeling sparse corruptions improves latent embeddings," in *KDD*, 2017.

[19] A. Bojchevski and S. Günnemann, "Adversarial attacks on node embeddings via graph poisoning," in *Proceedings of the 36th International Conference on Machine Learning, ICML*, ser. Proceedings of Machine Learning Research. PMLR, 2019.

[20] J. Li, H. Zhang, Z. Han, Y. Rong, H. Cheng, and J. Huang, "Adversarial attack on community detection by hiding individuals," in *Proceedings of The Web Conference*, 2020, pp. 917–927.

[21] M. Nickel, K. Murphy, V. Tresp, and E. Gabrilovich, "A review of relational machine learning for knowledge graphs," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 11–33, 2015.

[22] S. Ji, S. Pan, E. Cambria, P. Marttinen, and P. S. Yu, "A survey on knowledge graphs: Representation, acquisition and applications," *arXiv preprint arXiv:2002.00388*, 2020.