Knowledge Informed Sequential Scene Graph Verification Using VQA. Supplementary Material

Dao Thauvin
ONERA
Palaiseau, France
dao.thauvin@onera.fr

Stéphane Herbin
ONERA
Palaiseau, France
stephane.herbin@onera.fr

Abstract

The supplementary material document is organized in 5 sections:

- *Sec. 1 describes how we create inconsistencies in scene graphs;*
- *Sec. 2 adds more formal details on the different parts of our algorithm for better reproducibility;*
- *Sec. 3 explains how we adapted ECE [5] to our problem;*
- *Sec. 4 provides more analysis of our algorithm errors;*
- *Sec. 5 shows examples of sequences of our method on selected verification tasks.*

1. Creation of inconsistent scene graphs

First, we filter the data to keep only the graphs with more than one edge and we also remove nodes without connections: they have no context so the only way to find a correction is to verify that all hypotheses are true. We then generate inconsistencies from the ground-truth graph. We first perform a uniform sampling between *nodes* and *edges* to choose the type of element to modify. If the type of the selected element is a *node*, we carry out a uniform sampling among all the nodes connected to an edge to choose the node to modify. We choose an edge in a similar way. Finally, the new class of the element is sampled from all the possible classes minus the original true class.

2. Details of our approach

2.1. question module

The role of the question module is to provide plausible labels for an element q_U^t according to a context q_E^t and to provide a knowledge score $S_K^t(q_U^t, p)$ that takes a graph element and a label p.

Knowledge score To get the knowledge score $S_K^t(q_U^t, p)$, we compute a simple frequency estimation from the triplets contained in our training set.

More formally, we note $\#(q_E^t, c)$ the number of appearance of the triplet of labels $(C_V(s(e)), C_E(e), C_V(o(e)))$ whose q_U^t label has been replaced by c and $\#(q_E^t, \_)$ the same thing where label of q_U^t can be any label.
The final score is given by:

$$S_K^t(q_U^t, p) = \frac{\#(q_E^t, p)}{\#(q_E^t, \_)} \tag{1}$$

Proposal selection We select the 5 classes with the highest $S_K^t(q_U^t, p)$ to be the labels tested. We also test the current estimated class $C^t(q_U^t)$. To avoid never finding a class that is not in the 5 first classes, we allow to ask a same question multiple times: when asking a question a second time, we take the next 5 classes with highest score (the 6th to 10th classes with highest scores). And so on when asking the third, fourth, ... times. Its stops when all classes with non zero $S_K^t(q_U^t, p)$ are tested.

2.2. respond module

In the article, we define the VQA model and the questions to ask. We note the model V and the question $q_A(p)$ for a hypothesis p. The image based sore $S_I^t(q_U^t, p)$ is computed as follows:

$$\left(\frac{V(\text{"}yes\text{"}|q_A(p), I) + (1 - V(\text{"}no\text{"}|q_A(p), I))}{2} + b\right)/2 \tag{2}$$

To have a simple idea of the confidence of our network, we use the mean of two values as score: the output of V for the "yes" answer and the output for "no" answer of V. We don't use a softmax at the end of the network V but only a sigmoid (when using a softmax, the two values in our mean would be practically identical).
We also add b, an ambiguity prior according to the original

graph. This prior takes into account if p is the class of the element in the original graph to avoid changing the class to quickly (its value is then 1) and if p is already present in the graph to have an element multiple times in the graph (its value is then 0). If the two conditions are true we set it to 0.5 and if none of them is true, we set it to not modify the score obtained.

2.3. update module

We update the consistency score $S^t(u,c)$ of attributing the class value c to the element u in two ways.

We first compute the knowledge prior and image based score S^t_K and S^t_I and combine them to obtain $S^t_{I+K}(u,c) = \alpha S^{t'}_I(u,c) + (1-\alpha)S^{t'}_K(u,c)$.

In a second phase, we integrate the scores obtained at different time steps to obtain the final evaluation $S^t(u,c)$. For this, we define two sets of time steps by examining the history of the process. T_1: time steps t' where $q^{t'}_U = u$ and c is in the proposed labels of the time step. The time steps where u is verified. T_2 where $u \in \{s(q^{t'}_E), o(q^{t'}_E)\}, q^{t'}_E = q^{t'}_U$ and $C^{t'}(u) = c$. This set allows to modify the score of the subjects and objects of tested relationships. Indeed, if a relation is correct then its subject and object should also be correct. The score $S^t(u,c)$ is then:

$$\frac{\sum_{t' \in T_1} S^{t'}_{I+K}(v,c) + \sum_{t' \in T_2} \max_{c'} S^{t'}_{I+K}(q^{t'}_U, c')}{\#T_1 + \#T_2} \tag{3}$$

For ambiguity scores refer to Sec. 2.2

2.4. select module

2.4.1 RL-Based

Q-value Networks In this version, the selection of q^t_E and q^t_U is obtained by two separate Q-functions:

$$q^t_E = \arg\max_{E \cup \{\texttt{done}\}} Q_E(e|S^{t-1};\theta_E) \tag{4}$$
$$q^t_U = \arg\max_{u \in \{\texttt{s,o,r}\}} Q_U(u|S^{t-1}, q^t_E;\theta_U) \tag{5}$$

where a `done` output means that no interesting part needs to be verified and that the verification process can be halted. Note that an element can be verified more than one time, allowing to test more class hypotheses.

The two Q-functions are instantiated as Deep Q-Neural Networks (DQN) [3] parameterized by θ_E and θ_U and learned by reinforcement following a strategy comparable to [1].

We define the networks Q_E and Q_U used in our algorithm used to implement the Q-values as:

$$Q_E(e|\Omega^t_E) = M_E([\Psi_E(e); x^t(e); \Gamma(e)]) \tag{6}$$
$$Q_U(u|\Omega^t_U) = M_U([\Psi_U(q^t_E); x^t(q^t_E); \Gamma(q^t_E)]) \tag{7}$$

Here M is a Multi Layer Perceptron and Ψ a message passing network [2, 6, 7].
$x^t(e)$ is an edge representation defined as

$$x^t(e) = [\frac{\#(e, C^t(u))}{\#(e,\_)}, H^t(e,u) \quad \forall u \in \{e,s,o\}] \tag{8}$$

The first value uses the same mechanism as in Sec. 2.1 to extract information of the knowledge data.
$H^t(e,u)$ corresponds to the history of scores. It contains two values for each possible area selection (e,u):

- the highest score $max_{c,t'<t} S^{t'}_{I+K}(e,c)$ obtained by selecting the area (e,u) (-1 if no value exist) noted $H^t(e,u)[0]$
- another value noted $H^t(e,u)[1]$ being 1 if all non zero $S^t_K(e,u)$ label hypothesis for this area has been tested and 0 otherwise.

$\Gamma(e)$ is the function returning the stopping thresholds of the edge's object $o(e)$, the edge's subject $s(e)$ and its own threshold:

$$\Gamma(e) = [\gamma(C^t_E(e)), \gamma(C^t_V(s(e))), \gamma(C^t_V(o(e)))] \tag{9}$$

We want to take into account the edges adjacent to the processed edge in its processing. This allows the network to avoid asking a question on a node which has already been informed by the interrogation of another edge (in the case where a node is in common for two edges). For this we use message passing networks Ψ, defined such that $\Psi(e)$ is:

$$M_f\left([\mathbf{x}^t(e); \sum_{\substack{a \in \{s,o\} \\ b \in \{s,o\}}} \sum_{\substack{e' \in E \\ a(e)=b(e')}} l^{(a,b)}([\mathbf{x}^t(e); \mathbf{x}^t(e')])]\right) \tag{10}$$

$l^{(s,s)}$, $l^{(o,o)}$, $l^{(s,o)}$, $l^{(o,s)}$ are linear layers followed by a $\tanh$. They take into account the different types of relation between edges (same subject, same object, ...).

Stop sequential process Our sequential method must stop at some time when it is expected that its objective is satisfied. We must define a stopping rule based on the current verification state encoded as a consistency score function.

The idea for stopping is to detect when the process has found high consistency class label hypotheses for all the SG components, edges and nodes. The class labels can be the original verified SG, or other hypotheses that have been proposed during the process.

The highest consistency scores for each element is defined by $C^t(u)$ for an element $u \in E \cup V$. A simple strategy could be to use a fixed threshold and test if all the consistency scores are above this threshold. However, this strategy does not take into account the variability of prior and confidence scores provided by the image interpretation step

Criterion r_i^t	β_i	rt_E	rt_N
$\max_{u\in\{\mathtt{s},\mathtt{o},\mathtt{r}\}} S^{t-1}(u)$	-0.45	v	v
$1-\hat{c}^{t-1}(q_E^t)$	0.1	v	x
$1-\hat{c}^{t-1}(q_U^t)$	0.1	x	v
$\sum_{u\in\{\mathtt{s},\mathtt{r},\mathtt{o}\}} S^t(u)(\hat{c}^t(u)-\hat{c}^{t-1}(u))$	15	v	v
$H^t(q_U^t,q_E^t)[0]$	0.45	v	v
$1-H^t(q_U^t,q_E^t)[1]$	-15	x	v
$max_{u\in\{\mathtt{s},\mathtt{r},\mathtt{o}\}} (\lvert S^t(u)-S^{t-1}(u)\rvert)$	1	v	v

Table 1: Reward criteria for time step t. $\hat{c}^t(u)$ is 1 if u is correct at time t (the correct class is in the three highest scores), 0 otherwise. H is defined in Sec. 2.4.1. v indicates that the criterion is used for this agent reward, x otherwise.

Parameter	Value	Parameter	Value
ϵ exploration	50% to 5%	ϵ slope	0 to 33% of st
number of st	500000	max st	9 times E size.
training start	After 100 st	discount α	0.99
tn update	Every 500 st	lf	100 exchanges
buffer size	10000	batch size	128
θ_N, θ_E lr	2.5e-4	θ_γ lr	1.25e-2
optimizers	Adam	k	5
η	2	μ	100

Table 2: Learning Parameters: st means model steps, lr means learning rate, lf means learning frequency, E is the number of edges of the scene graph.

(VQA): several classes (entities or relations) are sources of error, and therefore require a higher coherence to be considered correct. However, the risk is to never output a stopping signal if the global threshold is to high. We propose instead to define for each class label $l \in \mathcal{L}_E \cup \mathcal{L}_V$ a learned threshold $\gamma(l)$ used to specialize the stopping rule to the nature of object and relations. Our solution is to let the algorithm learn how to stop if it considers that asking more questions will impact negatively the scores.

A stopping action `done` is added to the selection of q_E^t in select module when the stopping rule is verified.

Rewards Two rewards are used for reinforcement learning: r_E^t for q_E^t selection and r_U^t for q_U^t selection. Both a combination of different criteria in the following form:

$$\begin{cases} r_E^t = \sum_{i\in rt_E} \beta_i r_i^t & \text{for } Q_E \\ r_U^t = \sum_{i\in rt_N} \beta_i r_i^t & \text{for } Q_U \end{cases} \tag{11}$$

The exact criteria are given in Table 1 with their weights β_i. They have all different purposes. The first criteria is to penalize asking a question on an edge that has already high scores. The second and third are to promote the interrogation of incorrect elements. The fourth one is to take into account bad or good scores attributions (if the correct class has a high score or not). The fifth and sixth ones are used to avoid asking a question that has already a good score or that has no propositions to test. And the last one is to promote information gain on scores.

Stopping thresholds We describe how we obtain the thresholds ($\gamma(l)$ for a label l), the Q-value predicted to stop the process ans its rewars. To predict thresholds, we use a linear layer followed by a sigmoid that takes as input the current predicted label of an element. This layer is learned when predicting the two actions q_E^t, q_U^t but also when predicting the sequence end.

We now define the Q-value $Q_E(\mathtt{done}|S^{t-1};\theta_E)$ predicted to stop the model using the thresholds. The goal is to give a very low value when an element has no score to make the algorithm look at all graph elements. For this we define a value μ. If all scores are better than the thresholds a high value is given. Otherwise, a low score that remains higher than deteriorating the graph is given. For this we define a value η. Their values are given in Tab. 2.

When learning, we want to increase the thresholds when the graph is predicted to be correct when it is not and to reduce them when the graph is predicted to be incorrect when it is correct. For this, we use the distance between scores and thresholds $d^t(u) = S^t(u) - \gamma(C^t(u))$.

Giving:

$$\begin{cases} -\mu & \text{if } \exists u \in V\cup E.\quad S^t(u)=\mathtt{NA} \\ -\eta + \underset{u|d^t(u)>0}{mean}\, d^t(u) & \text{if } \exists u \in V\cup E.\quad d^t(u)\geq 0 \\ \eta - 1 + \underset{u\in V\cup E}{mean}\, d^t(u) & \text{otherwise} \end{cases} \tag{12}$$

We define a reward for the `done` action. To follow the learning process described before, it leads to the following rewards:

$$\begin{cases} -\mu & \text{if } \exists u \in V\cup E.\quad S^t(u)=\mathtt{NA} \\ \eta & \text{if all graph elements has been corrected} \\ -\eta & \text{otherwise} \end{cases} \tag{13}$$

Learning Our networks are DQNs networks [3]. Following previous works [3, 1], we use a MSE loss to learn them to predict future rewards. We give here the exact formula of the loss in the case of our reinforcement learning. The goals of the networks are to predict the rewards one can expect from asking the question:

$$\begin{cases} \hat{v}_E^t = r_E^t + \alpha \underset{u\in\{\mathtt{s},\mathtt{o},\mathtt{e}\}}{\max} Q_U(u|S^{t-1};\theta_E) & \text{for } Q_E \\ \hat{v}_U^t = r_U^t + \alpha \underset{e\in E\cup stop}{\max} Q_E(e|S^{t-1},q_E^t;\theta_U) & \text{for } Q_U \end{cases} \tag{14}$$

With $\theta_E = [W_f, W_Q, \theta_\gamma]$ and $\theta_U = [W'_f, W'_Q, \theta_\gamma]$ where W_f, W'_f are weights of M_f, W_Ψ and W'_Ψ are the weights of the message passing networks Ψ and finally θ_γ is the

weights of the linear layer used to predict the thresholds. We learn the networks through an MSE loss:

$$\begin{cases} loss_E = (\max\limits_{e \in E \cup \texttt{done}} Q_E(e|S^{t-1};\theta_E) - \hat{v}_E^t)^2 \\ loss_U = (\max\limits_{u \in \{\texttt{s},\texttt{o},\texttt{e}\}} Q_U(u|S^{t-1}, q_E^t;\theta_U) - \hat{v}_U^t)^2 \end{cases} \tag{15}$$

2.4.2 Rule-Based

The rule-based algorithm used is described below. It was designed to use the same inputs as RL-based and the same knowledge base. For the selection of the triplet to check q_E^t, the goal is to look first at edges that have not been verified by consulting the history H^t. Then, verify edges with low score. To select an element q_U^t , we first look at questions that have not been asked previously (primarily edges as it allows to verify the full triplet afterwards), and if the questions are all asked, we take the question that has received the lowest score.

Algorithm 1 q_E^t Selection

Require: G, S_K^t, H^t
  function Q_E
    $best \leftarrow \emptyset$
    $best\_score \leftarrow \inf$
    for $e \in E$ **do**
      $score \leftarrow \max_{u \in \{e, s(e), o(e)\}} S_K^t(u, C^t(u))$
      if $\forall n \in \{s(e), o(e), e\}, S^t(e,n) = \texttt{NA}$ **then**
        continue
      else if $\forall n \in \{s(e), o(e), e\}. \quad H^t(n,e)[0] = -1$ **then**
        return e
      else if $\exists n \in \{e, s(e), o(e)\} | S^t(e,n) = \texttt{NA}$ and $score < best\_score\}$ **then**
        $best\_score \leftarrow score$
        $best \leftarrow e$
      end if
    end for
    if $best \neq \emptyset$ **then**
      return $best$
    else
      return stop
    end if

Algorithm 2 q_U^t Selection

Require: G, S_K^t, H^t, q_E^t
  function Q_N
    $unseen = \{u \in \{\texttt{s},\texttt{o},\texttt{e}\} | H^t(q_E^t, u)[0] = \texttt{NA}\}$
    if $unseen = \{s, o, e\}$ **then**
      return e
    else if $unseen = \emptyset$ **then**
      $useful = \{n \in \{\texttt{s},\texttt{r},\texttt{o}\} \, | H^t(q_E^t, u)[1] = 1\}$
      return $\arg\min\limits_{u \in useful} H^t(q_E^t, u)[0]$
    else
      return $\arg\min\limits_{u \in unseen} S_K^t(u, C^t(u))$
    end if

2.5. VQA fine-tuning

We finetune the VQA model for edge questions (for node questions, the model is used as it is). The model is trained for 10 epochs on the training set. For each image, 8 questions are asked in the form "Is there <subject> <relation> <object>?" where (<subject>,<relation>,<object>) corresponds to a triplet of classes in the graph with a 1 in 2 chance that one class among the triplet is modified. If the triplet is modified then the expected answer is "no", otherwise "yes". We use the same parameters as those of the pre-trained model

2.6. Parameters

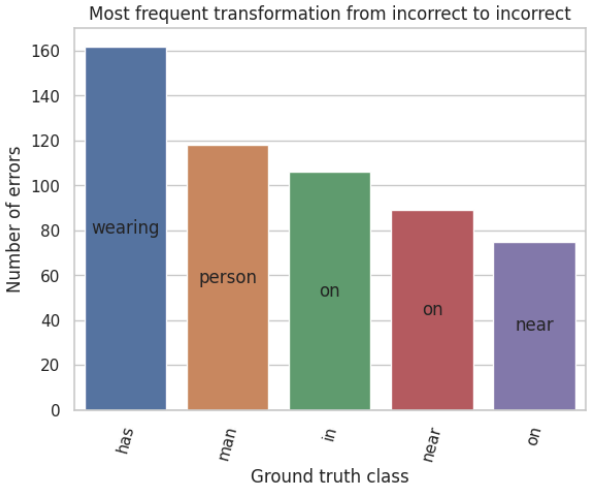
The training parameters are given in Tab. 2.

We also describe the different multi layer perceptrons: M_f (Eq. (10)) is a 2 linear layers with hidden layer of 16 neurons with a ReLU and a dropout of 0.5 between them and an output size of 16. M_U (Eq. (7)) and M_E (Eq. (6)) are 3 linear layers with a first hidden layer of size 24 and a second of size 48; they have an output of size 3 (`o`, `s` and `r`) for the first and 1 for the second. The layers are separated by a dropout of 0.5 and a tanh.

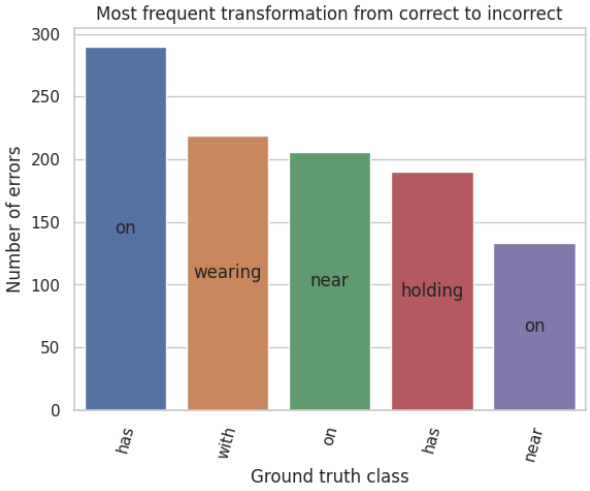
3. ECE baseline

In this part, we give more details on how we adapted ECE [5] to our task and the difficulties encountered. A first issue is that ECE takes as input a caption and not a scene graph. Several methods have proposed to use scene graphs to generate captions [8, 4]. However, those methods do not propose to use a ground-truth scene graph as input and don't ensure that all information of the scene graph will be present in the caption(s) making it difficult to apply these approaches for our task. An example of the lack of information encoded in a caption when compared to a scene graph is shown in Fig. 2

To translate scene graphs into captions, a first direct idea is to concatenate all triplet class labels of each edge e as simple phrases: "$C_V(s(e))C_E(e)C_V(o(e))$.". But the overall



(a) for not corrected corrupted elements

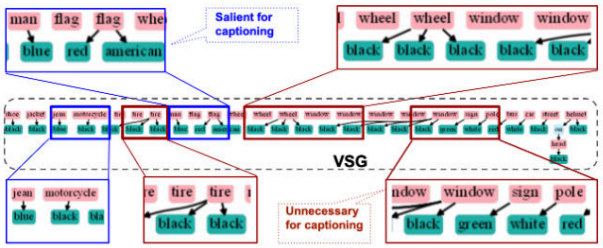


(b) for modified correct elements

Figure 1: Most frequent highest score labels for a ground-truth class when ground-truth is not found (if ground truth class is not in the 3 highest scores). Each bar is labeled with the label for which our algorithm gives the highest score and ground truth labels are on the x abscissa.

caption then becomes too long: when the original algorithm limits itself to 32 tokens in entry, our captions can easily exceed 100 tokens even when only looking at objects one by one and building a caption describing all its relations. Instead, we apply the ECE correction to each phrase generated by all single triplets to get the full correction. The problem with that setting is that a given node can be present in different triplets, making it possible to have different corrections for a same node. We take this phenomenon into account in our metrics: we rank the predicted class labels by their number of occurrence for each node in a triplet:

(a) An image and the TSG generated from its caption



(b) VSG containing all detected objects as nodes

Figure 2: Illustration from [8] of a caption and a scene graph (named TSG) generated by their method (a) compared to its ground-truth VSG (b). We observe that a lot of information is lost since it is being considered useless for captioning (the presence of windows for example).

if the correct class is in the 3 predicted classes that occur more frequently, the class is considered correct. This also allows ECE to take into account possible synonyms as we do. Another problem is that ECE sometimes have syntax problems: it removes a word without replacing it (for example "cat riding bike" becomes "cat riding"). In this case, we consider the deleted element class as its class before applying correction.

We trained ECE on our dataset in the same settings as the original paper. To create our training set, we randomly select for each image two triplets with a probability of 0.5 that one triplet is corrupted. We trained it using datasets with one inconsistency, three inconsistencies and generated graphs. This made 65 596 triplets to train the model for one and three inconsistencies and 16 884 triplets for generated graphs.

4. Distribution of errors

To understand the nature of the errors of our algorithm, we look for the labels that have been modified by our algorithm with 1 inconsistency per graph with our RL-based algorithm. To do so, we look at the most frequent labels with highest score when the ground-truth class is not found in the 3 labels with highest score. To have a better insight, we divide errors in 2: first, errors where the algorithm didn't correct a wrong element in Fig. 1a (false neg-

ative) and errors where the algorithm creates a false inconsistency in Fig. 1b (false positive). On Fig. 1a, we observe that for the edge labels generating inconsistency detection, our algorithm seems to give more precise predicates (`has` vs. `wearing`). We also observe that our algorithm creates false inconsistency detection for very general labels (`has` vs. `on`). This can be explained as general labels have a lot of synonyms with higher scores as they are more specifics. It seems to be the converse however for nodes with `man` and `person` labels. This problem can come from our biased knowledge base or our VQA model, preferring to predict person to not assume gender. From this first analysis, we see that the most frequent inconsistency detection errors come from confusion between labels with very close meaning. This first level of analysis is very coarse, and we show in Sec. 5 an analysis of our method behavior and causes of error.

5. Examples

In this section, we analyze the behavior of our RL-based algorithm, in particular the fact that the overall decision structure takes the form of a sequence and that each step is divided in several modules, each potentially generating errors of various nature.

The visualization of the sequence that generated each example is available as an animated `video` images in the `images` directory accessible or from `index.html`. It contains various examples of our algorithm result with different levels of inconsistency. Each `video` and final result contains:

- In the top right corner, the image.
- In the top left corner, the input scene graph, where the ground truth of corrupted elements is written in brackets.
- In the bottom left corner, the three highest scores at current step of our algorithm for each graph element. If it is not final step, the edge selected by our algorithm q_E^t is surrounded in red and the verified element has a yellow background.
- In the bottom right corner, the image and knowledge score for each proposition at current step if not final step.

We also give the final result of each example in this supplementary.

First, we give some examples that were successfully corrected by our algorithm in Fig. 3: Fig. 3a shows that the algorithm is able to give several relevant class labels for an edge, justifying the importance to look at the first three labels for each graph element to allow lexical ambiguity: here "on", "above" and "sitting on" are all acceptable classes for the edge "bowl sitting on table". Our algorithm is also capable of taking into account the fact that some information is already present in the scene graph as we see in Fig. 3b: instead of repeating "elephant has leg" or "elephant has tail", it gives the correct node class "elephant has trunk". Fig. 3c shows that our algorithm is able to correct connected erroneous information if remaining information are enough to deduce them. The last example Fig. 3d shows an example of correction for a generated graph.

We will now analyze in a more refined way the inconsistency detection errors and identify their possible cause.

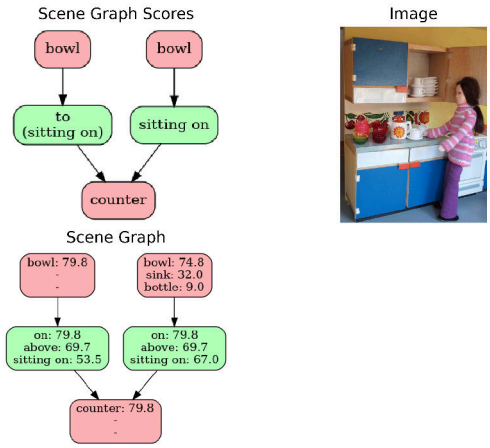
The more obvious errors are when the `respond` step (output of the VQA) gives bad results as we see in Fig. 4: in Fig. 4a, "lady wearing short" is considered true when no short is visible in the scene. In Fig. 4b, "ear" and "tail" are not detected correctly, making "sock" having a higher score. In Fig. 4c, "person holding phone" and "person watching man" are detected when they are not in the image.

We can also observe errors due to the `question` step (selection of propositions) in Fig. 5: in the first image, Fig. 5a, "tree" is never found for the triplet "tree behind fence". Fig. 5b shows another example of this with a triplet completely incorrect where no propositions are available. It explains the lower performance for 3 and generated inconsistencies.

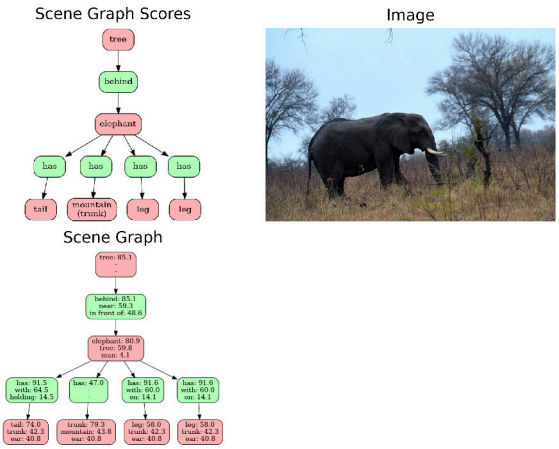
Some errors are due to stopping too early by only verifying edges of a node without checking other classes for it for example: Fig. 6a show this with the "shirt" node of ground-truth "short" that is not modified. We can also see this in Fig. 6b with the "arm" node.

Other errors are due to bad selection that leads to bad reasoning of image processing: in Fig. 7a, the algorithm first look at "car" and "window" when "vase" is the incorrect element, creating a relation "handle of street" with a high score that could be avoided by looking at "vase" first. In Fig. 7b, when looking at "fruit in cabinet", our algorithm first looks at "fruit", modifying it as "book" when the real triplet is "fruit in bowl". This could be avoided by looking at "cabinet" first.

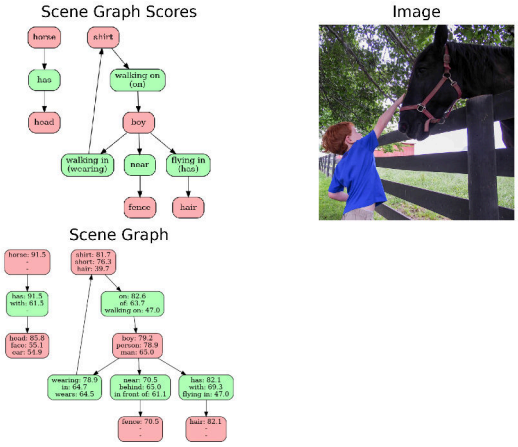
To finish, we give some examples of bad corrections that seem correct in Fig. 8: In Fig. 8a, "sign on pole" is replaced by "letter on sign" and in Fig. 8b, "guy with bag" and "person in shirt" are replaced by "guy holding bag" and "person wearing jean" respectively.



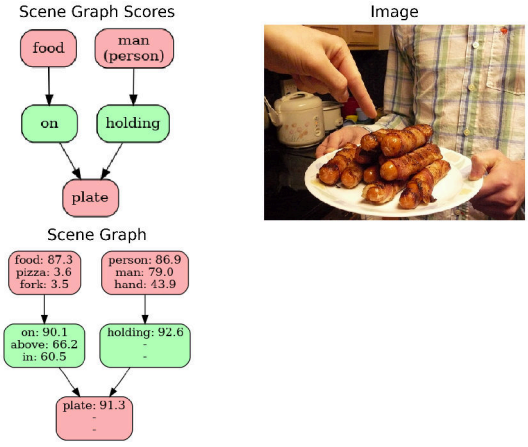
(a) 1 inconsistency



(b) 1 inconsistency

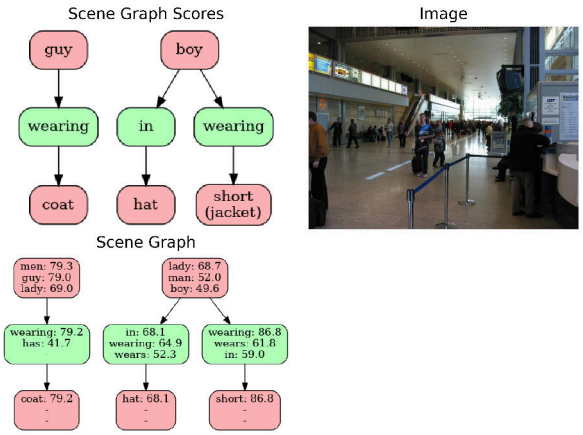


(c) 3 inconsistencies

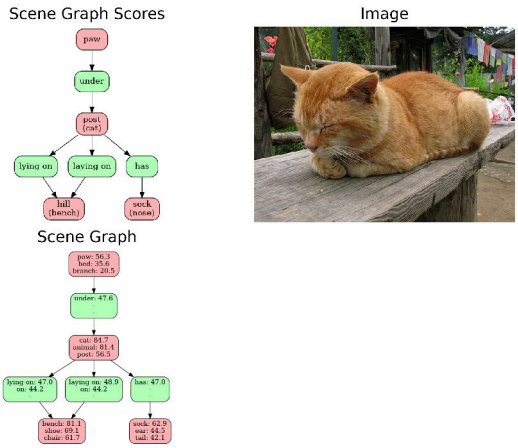


(d) Generated

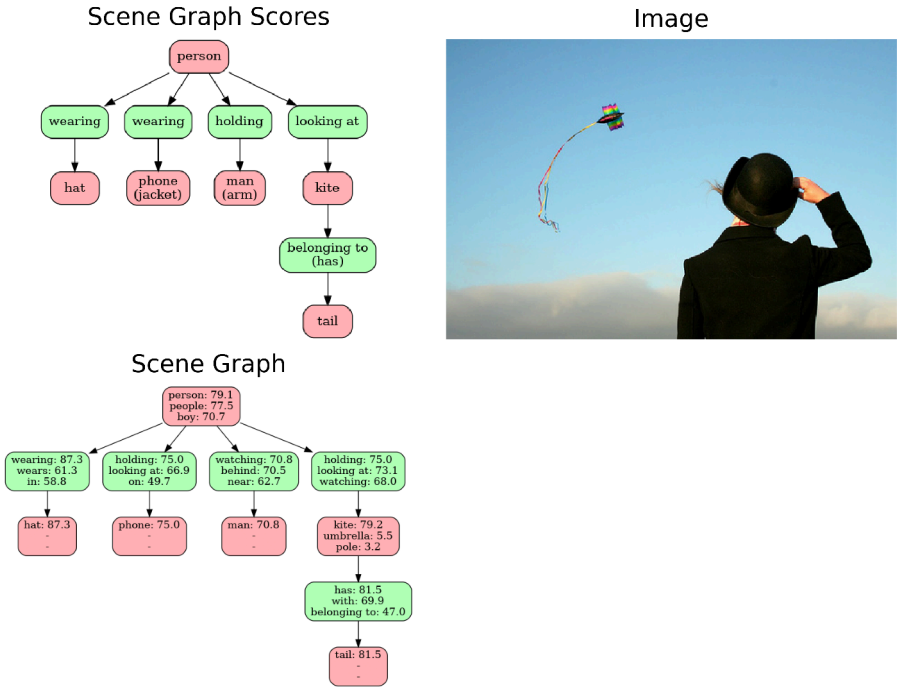
Figure 3: Examples of successful corrections of our algorithm. `video` files are available in the `correct` directory.



(a) 1 inconsistency

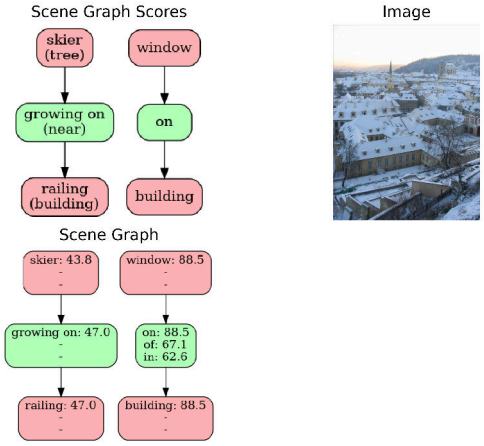


(b) 3 inconsistencies

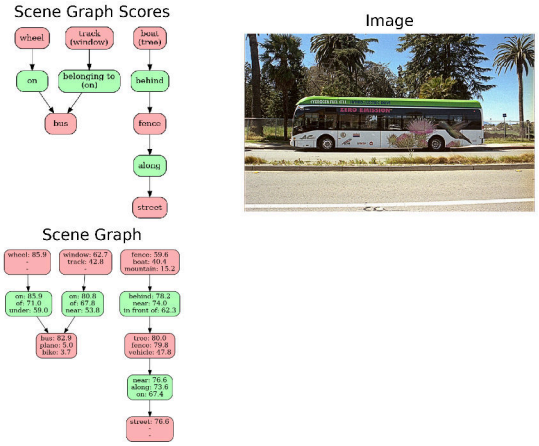


(c) 3 inconsistencies

Figure 4: Examples of incorrect corrections of our algorithm caused by respond component. `video` files are available in the `respond_error` directory.

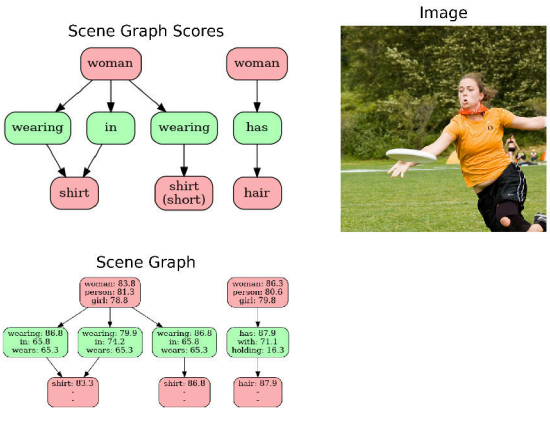


(a) 3 inconsistencies

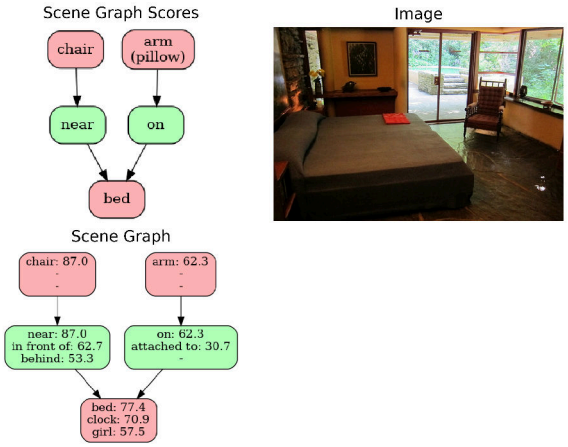


(b) 3 inconsistencies

Figure 5: Examples of incorrect corrections of our algorithm caused by question component. `video` files are available in the `question_error` directory.

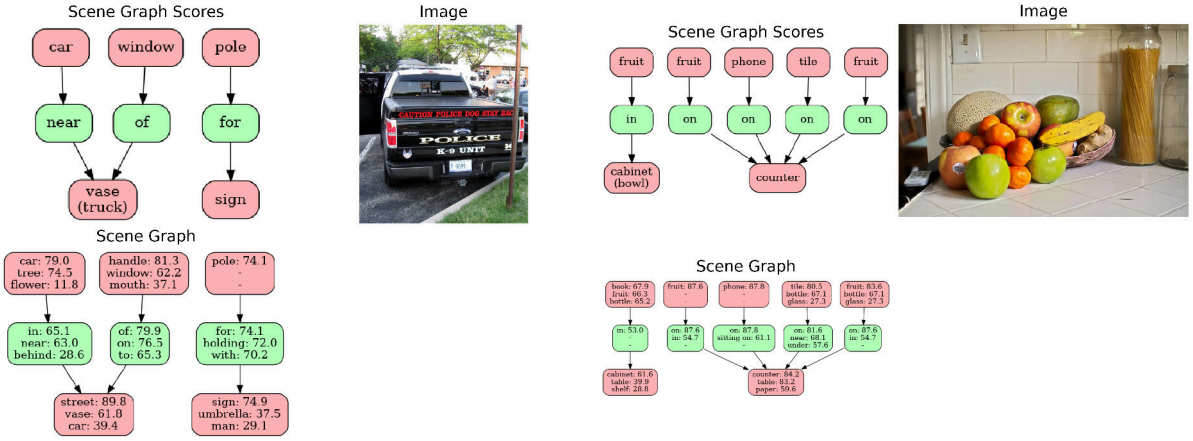


(a) 1 inconsistency



(b) generated

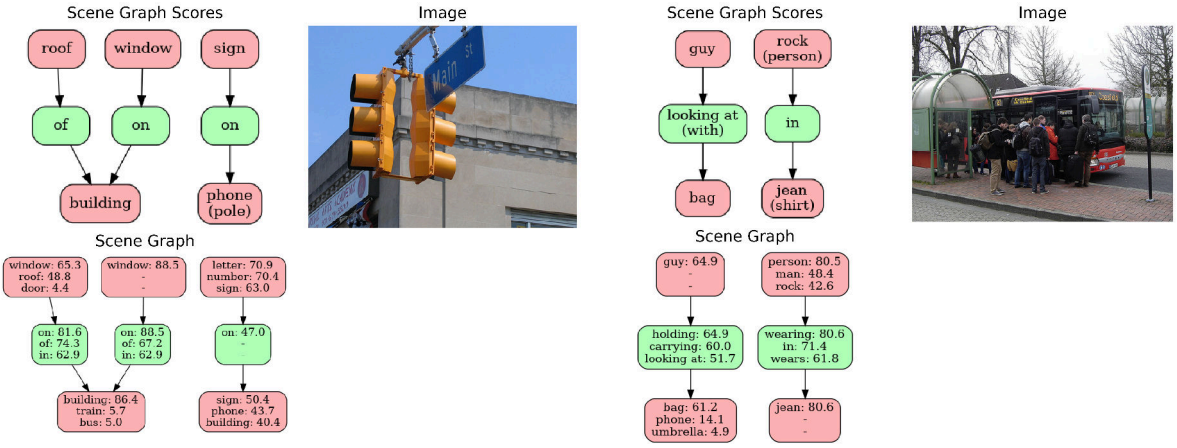
Figure 6: Examples of incorrect corrections of our algorithm caused by stop component. `video` files are available in the `stop_error` directory.



(a) 1 inconsistency

(b) 1 inconsistency

Figure 7: Examples of incorrect corrections of our algorithm caused by select component. `video` files are available in the `select_error` directory.



(a) 1 inconsistency

(b) 3 inconsistencies

Figure 8: Examples of incorrect corrections of our algorithm where a different correct relation appears in the graph. `video` files are available in the `new_triplet` directory.

References

[1] Hanjun Dai, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, and Le Song. Adversarial Attack on Graph Structured Data. *arXiv:1806.02371 [cs, stat]*, June 2018. arXiv: 1806.02371. 2, 3

[2] Sohir Maskey, Ron Levie, Yunseok Lee, and Gitta Kutyniok. Generalization Analysis of Message Passing Neural Networks on Large Random Graphs, Aug. 2022. arXiv:2202.00645 [cs, math]. 2

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning. *arXiv:1312.5602 [cs]*, Dec. 2013. arXiv: 1312.5602. 2, 3

[4] Kien Nguyen, Subarna Tripathi, Bang Du, Tanaya Guha, and Truong Q. Nguyen. In Defense of Scene Graphs for Image Captioning, Aug. 2021. arXiv:2102.04990 [cs]. 4

[5] Zhen Wang, Long Chen, Wenbo Ma, Guangxing Han, Yulei Niu, Jian Shao, and Jun Xiao. Explicit image caption editing. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXXVI*, pages 113–129. Springer, 2022. 1, 4

[6] Danfei Xu, Yuke Zhu, Christopher B. Choy, and Li Fei-Fei. Scene Graph Generation by Iterative Message Passing. pages 5410–5419, 2017. 2

[7] Xu Yang, Chongyang Gao, Hanwang Zhang, and Jianfei Cai. Hierarchical Scene Graph Encoder-Decoder for Image Paragraph Captioning. In *Proceedings of the 28th ACM International Conference on Multimedia*, MM '20, pages 4181–4189, New York, NY, USA, Oct. 2020. Association for Computing Machinery. 2

[8] Yiwu Zhong, Liwei Wang, Jianshu Chen, Dong Yu, and Yin Li. Comprehensive Image Captioning via Scene Graph Decomposition. *arXiv:2007.11731 [cs]*, July 2020. arXiv: 2007.11731. 4, 5