Efficiency-aware Answering of Compositional Questions using Answer Type Prediction

David Ziegler, Abdalghani Abujabal, Rishiraj Saha Roy and Gerhard Weikum
Max Planck Institute for Informatics
Saarland Informatics Campus, Germany
{daziegler, abujabal, rishiraj, weikum}@mpi-inf.mpg.de

Abstract

This paper investigates the problem of answering compositional factoid questions over knowledge bases (KB) under efficiency constraints. The method, called TIPI, (i) decomposes compositional questions, (ii) predicts answer types for individual sub-questions, (iii) reasons over the compatibility of joint types, and finally, (iv) formulates compositional SPARQL queries respecting type constraints. TIPI's answer type predictor is trained using distant supervision, and exploits lexical, syntactic and embedding-based features to compute context- and hierarchy-aware candidate answer types for an input question. Experiments on a recent benchmark show that TIPI results in state-of-the-art performance under the real-world assumption that only a single SPARQL query can be executed over the KB, and substantial reduction in the number of queries in the more general case.

1 Introduction

Motivation. Question answering over knowledge bases (KB-QA) has gained attention, facilitated by the rise of large-scale knowledge bases such as Freebase (Bollacker et al., 2007), DBPedia (Auer et al., 2007) and YAGO (Suchanek et al., 2007). The key challenge for KB-QA is to align mentions and relational phrases in a natural language question to semantic items in the KB (entities and predicates), and to construct a valid SPARQL query that is then executed over the KB to retrieve crisp answers (Abujabal et al., 2017; Bast and Haussmann, 2015; Yahya et al., 2013; Yih et al., 2015).

Questions going beyond simple factoid questions (like *"Who won a Nobel Prize in Physics?"*) that are prevalent in popular KB-QA benchmarks are generally out of scope for most state-of-the-art KB-QA systems (Yih et al., 2015; Dong et al., 2015; Berant and Liang, 2015). However, questions like *"Who won a Nobel Prize in Physics and was born in Bavaria?"*, can be decomposed into a set of simpler questions *"Who won a Nobel Prize in Physics?"* and *"Who was born in Bavaria?"*. We refer to such questions as *compositional questions*, and these are the focus of this work.

Limitations of state-of-the-art. A few past approaches that can handle such compositional questions (Bao et al., 2016; Xu et al., 2016; Abujabal et al., 2017) generate and execute candidate SPARQL queries for each sub-question separately and/or use the intersection as the final answer (*Werner Heisenberg*, among others). This creates the challenge of deciding which queries from the different sub-questions fit together. Past efforts use information about answers to all generated SPARQL queries, and retrospectively choose a query pair from among these whose answer intersection is non-empty. However, this mode of operation is highly inefficient, since it necessitates execution of all generated queries, followed by a ranking or aggregation phase. We aim to address this concern, leveraging the compatibility of expected *answer types*. Such compatibility, as we show, has the potential to prune the search space of candidate SPARQL queries.

Approach. Given a complex question, our proposed method TIPI first decomposes it into simple sub-questions. Next, it predicts a ranked list of fine-grained answer types for each sub-question, respecting a type hierarchy. TIPI then seeks candidate SPARQL queries corresponding to these sub-questions as input from an underlying KB-QA model, and finds pairs of compatible queries from among these using hierarchy-based reasoning. Finally, it scores these pairs and finds the best

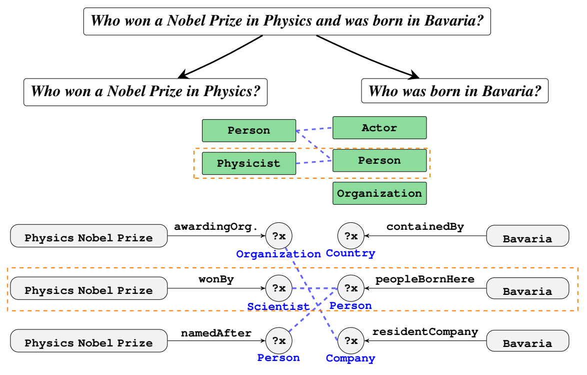

Figure 1: A toy example showing how TIPI works. Top: Decompose question. Middle: Determine compatible types among answer types of the two sub-questions. Bottom: Stitch compatible queries together and prune queries by answer type. This is followed by scoring and ranking of the query pairs to formulate the best compositional query.

pair to formulate a compositional query, to be executed over the KB. The final two steps contribute towards TIPI's efficiency-aware perspective.

Contributions. The paper makes the following three contributions:

- We are the first to exploit answer types for handling compositional questions in KB-QA;
- Experiments show state-of-the-art performance under practical efficiency constraints;
- Finally, TIPI contains a hierarchy- and context-aware fine-grained answer typing module that can be used as a plug-in by any KB-QA system.

2 Answering Compositional Questions

We now explain the steps by which TIPI handles the answering of a compositional question, taking the running example of *"Who won a Nobel Prize in Physics and was born in Bavaria?"*. A simplified workflow is shown in Figure 1.

Question decomposition. Given a compositional question, TIPI first decomposes it into simple sub-questions *"Who won a Nobel Prize in Physics?"* and *"Who was born in Bavaria?"*, using dependency parse patterns (Xu et al., 2016; Bao et al., 2014). These patterns can handle several kinds of compositional questions with multiple entities or relations (e.g., questions with relative clauses and coordinating conjunctions). Question decomposition has also been applied to the IBM Watson system (Boguraev et al., 2014), where the authors separated cases as being either *parallel* or *nested*. This work deals with parallel decomposition, and handling nesting like *"which daughter of john f. kennedy studied at radcliffe college?"* is future work.

Answer Type prediction. Next, we predict a set of *expected answer types* for each sub-question using an answer type predictor, described in the next section. The typing module is a context-aware hierarchical classifier that takes a question as input and produces a set of answer types along with confidence scores as output (like `Person`: 0.8, `Actor`: 0.3, `Scientist`: 0.6; scores need not add up to one).

Type compatibility scoring. We define two types to be *compatible* whenever they are the same, or one is an ancestor of the other, according to some type system. For example, `Scientist` and `Person` are compatible, as the former is a descendant of the latter. Now, let T_1 and T_2 be a pair of predicted types for the first and second sub-questions, respectively. We score each such pair with a linear combination of *granularity* and *confidence* as follows:

$$\begin{aligned}score(T_1, T_2) &= \gamma \cdot max(level(T_1), level(T_2)) \\ &+ (1-\gamma) \cdot avg(conf(T_1), conf(T_2))\end{aligned}$$

if T_1 and T_2 are compatible, else zero. Here, $level(T_i)$ and $conf(T_i)$ refer to the level of T_i in the type system, and the classifier prediction confidence for T_i, respectively. We take the root to be at level zero, with finer types having higher levels. Thus, pairs with fine-grained types (like `Physicist`), and with high prediction confidences, accumulate higher scores. Mixing parameter γ is chosen to be 0.5. We take the highest scoring type pair, and then take the finer of the two types as the final prediction (`Physicist` in this example). The intuition here is that one sub-question might reveal more typing information (*"Who won a Nobel Prize in Physics?"* gives `Physicist`) than the other (*"Who was born in Bavaria?"* only gives `Person`).

Compositional query formulation. We now let a KB-QA system generate a ranked list of SPARQL queries for each sub-question. In past work, all query pairs (one from each sub-question) are combined to create numerous compositional queries. Our goal is to stitch only type-compatible queries, which is achieved as follows. Each query predicate has a *type signature* from the KB, which states the expected types for the pred-

[.*]* [*which*|*what*] [lemma: be]? [det]? [adj|adv]* [noun]+ [.*]* *(E.g.: What is the current currency in Germany?)*
[.*]* [*which*|*what*] [lemma: be]? [det]? [adj|adv]* [noun] [*of*] [det]? [adj|adv]* [noun]+ [.*]* *(E.g.: What kind of currency does Germany have?)*
[.*]* [*which*|*what*] [lemma: be] [det]? [noun]+ [poss] [adj|adv]* [noun]+ [.*]* *(E.g.: What is Germany's currency?)*

Table 1: Patterns to extract lexical answer types (underlined). Symbols are borrowed from standard regular expression conventions.

icate. We use our final predicted type to remove all queries from either sub-question whose type signatures are not compatible to it. Thus, queries like `PhysicsNobelPrize wonBy x?` and `Bavaria peopleBornHere x?`, with compatible type signatures `Physicist` and `Person` are retained, while `Bavaria containedBy x?` and `PhysicsNobelPrize awardingOrg x?` with signatures `Country` and `Organization` are removed. All surviving query pairs are combined, and are then scored by the sum of the inverses of the ranks they had from the underlying KB-QA model. The pair that maximizes this rank inversion score is finally chosen for execution by the system: `PhysicsNobelPrize wonBy x? . Bavaria peopleBornHere x?` for our example.

3 Predicting Answer Types

Our strategy for answer type prediction is to harness explicit clues available in the question, and to resort to more implicit ones only when such signals are not present. We thus use a two-stage approach, inspired by work on named entity typing (Del Corro et al., 2015; Yosef et al., 2012)[1]: (1) candidate collection, using lexico-syntactic patterns to identify lexical types which can be mapped to a set of semantic types using lexicons; (2) type selection, using a hierarchical classifier to disambiguate among the candidates. *Note that in absence of explicit clues for candidates in the question, our method proceeds directly to the second stage.*

Candidate Collection. To extract the lexical answer type from a question, we use simple POS patterns (examples in Table 1), utilizing Stanford CoreNLP for tokenization and POS-tagging (Manning et al., 2014). In a second step, the lexical type extracted from the question is mapped to a set of semantic types (KB-types) using lexicons (Berant et al., 2013). Such lexicons can be constructed by mining entity-annotated relational phrases in CLUEWEB09 (Lin et al., 2012). For example, the lexical type *'physicist'* may map to the semantic types `Scientist`, `Theoretical physicist` and `Quantum physicist`.

Type Selection. From the candidate collection phase, we receive a set of candidate types, for each of which we run a binary classifier to get a confidence score. If no lexical answer type could be found in the question (like *"Where did Einstein study?"*) or there were no lexicon entries, the classifier makes predictions on all types from our type system. The hierarchical classifier works as follows: starting at the root of the type system we predict probabilities for its children that are candidates, using per-type binary classifiers. Iteratively, this process is repeated for sub-trees whose root has a confidence score above a global threshold α.

Training. We start with a dataset where each question is labeled with a set of expected answer types. We use the *siblings strategy* (Silla and Freitas, 2011) to train a classifier for each type: we use all questions labeled with `T` as positive instances for a type `T` ; we use *only* those questions as negative instances that are labeled with some sibling of `T` according to the type system. We use four sets of *features*: (i) *Surface features:* n-grams of length up to three from the question; (ii) *Dependency parse features:* n-grams constructed by hops over relations in the dependency tree to capture long-range dependencies; (iii) *Word2vec features:* Per question, we add ten words most similar to the question words (except for stop-words and entities) using a pre-trained model of word2vec (Mikolov et al., 2013); and (iv) *Question length:* The number of tokens in the question. To reduce data sparsity, we replace all question entities by a wildcard for pattern generalization, before computing the features.

4 Experiments

4.1 Setup

Dataset. We use the very recent dataset of 150 compositional questions[2] created by Abujabal et al. (2017) which were sampled from the 2013

[1]While the task in named entity typing is somewhat similar to answer typing, the crucial difference is that in the latter the entity (answer) itself is missing.

[2]Available for download at `http://people.mpi-inf.mpg.de/~abujabal/publications/quint/complex-questions-wikiasnwers-150.json`, Accessed 22 September 2017.

WikiAnswers resource (Fader et al., 2013). Every question in this dataset was constrained to have more than one named entity or relational phrase (e.g., *"Who directed Braveheart and Paparazzi?"* and *"Who directed Braveheart and played in Mad Max?"*). We use Freebase as the underlying KB.

Baselines. We use the following baselines: the open-source AQQU system (Bast and Haussmann, 2015) (best performing public KB-QA system on the popular WebQuestions benchmark (Berant et al., 2013)), and reimplemented versions of the answer stitching mechanism (ANSSTITCH) in Abujabal et al. (2017), and the query stitching mechanism (QUERYSTITCH) in Bao et al. (2016). As mentioned in Section 2, TIPI acts as a plug-in for KB-QA systems, and hence needs an underlying model to generate queries. To this end, we build and evaluate AQQU+TIPI and QUERYSTITCH+TIPI. If question decomposition fails, or if TIPI prunes away all queries with a non-empty answer set, we *back off* to the original model. Abujabal et al. (2017) use an alternative approach of stitching answers for sub-questions instead of queries, and hence it is not meaningful to have a comparable ANSSTITCH+TIPI system.

Training the type predictor. We use the 1000 most frequent types from Freebase as our type system. Since there is no dataset directly containing questions and answer types, we resort to *distant supervision* as follows: for each gold answer to a question in WebQuestions (WQ) (Berant et al., 2013) and SimpleQuestions (SQ) (Bordes et al., 2015), we look up its *notable types* from Freebase. This is often a large set, agnostic to the context of the specific question at hand: e.g. `Germany`, the answer to *"What is Albert Einstein's nationality?"*, has 53 notable types, including context-irrelevant ones like `Filming Location` and `Treaty Signatory`. We prune this set in two ways: (1) keep only those types that are in our type system, and (2) retrieve *expected types* of Freebase predicates corresponding to relations in the question (e.g. `hasNationality` has the expected object type `Country`) and only retain types compatible to these. This is possible via gold SPARQL queries available in WQ and SQ. We then train the hierarchical classifier on a total of $10k$ questions ($3k$ from WQ, $6.4k$ from SQ, and $0.6k$ Freebase type descriptions). Threshold α was tuned to 0.6 on a development set of $2k$ questions ($0.8k$ from WQ and $1.2k$ from SQ) by optimizing on F1.

Feature Sets	Prec (Auto)	Prec (Human)
Surface	71.9	65.6
Surface + DP	72.2	65.9
Surface + DP + w2v	73.2	67.1
Surface + DP + w2v + QLen	**73.9**	**67.3**

Table 2: Intrinsic evaluation of type prediction, showing effects of feature ablation.

4.2 Results

Intrinsic evaluation of type predictor. We evaluate our type predictor intrinsically in two ways: (i) automatic evaluation, treating the labels generated by distant supervision as gold labels, and (ii) human evaluation. Three human annotators marked, for each prediction, whether it is correct and context-aware: e.g. for *"Who developed the theory of relativity?"*, they would mark `Scientist` and `Person` as correct but not `NobelPrizeWinner` or `Teacher`. Note that in the automatic evaluation, the context-oblivious types might be deemed as correct. Table 2 shows the results of automatic evaluation in the left column and human evaluation in the right column. The predictions are of high precision, even under human evaluation which only considers context-aware predictions as correct. Mean Cohen's κ (Cohen, 1960) is 0.743, showing good inter-annotator agreement. Feature ablation shows that each of the four feature sets is useful.

Answering performance. Since each question potentially has a set of correct answers, we compute precision, recall and F-score for each question, and then average it over all 150 questions. We evaluate two setups: (i) when only the top-ranking compositional query is executed, and (ii) when the rank-based query scoring component is disabled and all surviving compositional queries (combinations of type-compatible queries) are executed. Results are presented in Table 3, where we make the following key observations:

TIPI results in state-of-the-art performance. In the case of best query execution, we find that TIPI's preferential scoring system for fine-grained types proves highly effective: QUERYSTITCH+TIPI achieves the highest F1 on the dataset (0.367). It also has the best overall recall (0.469), and is in the top-2 for precision. Thus, TIPI helps attain the best performance under the practical constraint of executing only a single query and eliminating the overhead of answer aggregation.

TIPI improves efficiency while maintaining

Approach	Only best query executed over KB			All queries executed over KB			
Metric	Precision	Recall	F1-Score	#Queries	Precision	Recall	F1-Score
AnsStitch (Abujabal et al., 2017)	0.345	0.409	0.344	97.4	**0.495**	**0.556**	**0.485**
AQQU (Bast and Haussmann, 2015)	0.245	0.466	0.253	54.5	0.245	0.466	0.253
AQQU+**TIPI**	0.209	**0.515**	0.231	**35.7**	0.209	0.515	0.231
QueryStitch (Bao et al., 2016)	**0.358**	0.428	**0.360**	57.5	**0.459**	0.544	**0.456**
QueryStitch+**TIPI**	**0.356**	**0.469**	**0.367**	**36.0**	0.439	**0.566**	0.445

Table 3: Answering performance on 150 compositional questions from Abujabal et al. (2017). The two best (*minimum* for #Queries, *maximum* for all other columns) values in every column are marked in **bold**.

comparable performance. In the unconstrained scenario, by reasoning over type compatibility, TIPI substantially reduces the number of queries that the KB-QA system executes (35.7 from 54.5 (AQQU), and 36.0 from 57.5 (QueryStitch)). This is achieved while maintaining comparable F1 performance. Moreover, note that the best performing method with TIPI (QueryStitch+TIPI) improves efficiency by a factor of 2.71 on the overall best performing AnsStitch (97.4 queries to 36.0). The generally lower values corresponding to the AQQU rows is because AQQU was originally designed for single-relation KB-QA.

Analysis. Question decomposition failures are the primary cause of error. TIPI's success hinges on the triggering of question decomposition rules adapted from Xu et al. (2016); it is worthwhile to note that results were even more encouraging when the 43 questions for which the rules did not fire were excluded from the analysis. Results averaged over the remaining 107 questions show that QueryStitch+TIPI performs the best on all three metrics (F1 = 0.372, Prec = 0.365, Rec = 0.402) under best-query execution. In the all-queries case, QueryStitch+TIPI reduces the number of queries by an even greater factor of 3.13 (w.r.t. AnsStitch) with only a 0.02 drop in F1. This error analysis suggests that better question decomposition, going beyond simple syntactic rules, will improve overall performance. Finally, representative questions that could only be answered when TIPI was used as plug-in, are shown in Table 4.

"who is the president of the us who played in bedtime for bonzo?"
"who played for ac milan and inter milan?"
"what movie did russell crowe and denzel washington work on together?"
"which country were the adidas and puma footwear founded?"

Table 4: Questions that could be answered only when TIPI was used as a plug-in.

5 Related Work

Answer typing has proved effective both for text-based QA (Ravichandran and Hovy, 2002) and KB-QA (Bast and Haussmann, 2015; Savenkov and Agichtein, 2016), for example, in ranking of answers (Murdock et al., 2012) or queries (Yavuz et al., 2016). Answer typing was mostly limited to considering coarse-grained types (Bast and Haussmann, 2015; Lally et al., 2012) and lexical answer types (Berant and Liang, 2015; Abujabal et al., 2017). Both such modes fail when the answer type is not explicit. More recently, Yavuz et al. (2016) exploit more implicit type cues for KB-QA: but their method of creating training data is context-agnostic, which we remedy in our work. An early line of work deals with *question classification* (Li and Roth, 2002; Blunsom et al., 2006; Huang et al., 2008), but they were designed for a handful of TREC types and is not really relevant for KB-QA with thousands of distinct classes. Finally, this work is the first to harness answer types for compositional KB-QA.

6 Conclusion

We presented TIPI, a mechanism for enabling KB-QA systems to answer compositional questions using answer type prediction. TIPI relies on a fine-grained answer typing module, that respects question context and type hierarchy. Experiments on a recent benchmark show that TIPI achieves state-of-the-art performance under single-query execution, and substantial query reduction when the top-1 query constraint is relaxed to admit more queries. Improving question decomposition, and handling more implicit forms of question compositionality, are promising future directions.

References

Abdalghani Abujabal, Mohamed Yahya, Mirek Riedewald, and Gerhard Weikum. 2017. Automated template generation for question answering over knowledge graphs. In *WWW*.

Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. 2007. Dbpedia: A nucleus for a web of open data. *The Semantic Web*.

Junwei Bao, Nan Duan, Zhao Yan, Ming Zhou, and Tiejun Zhao. 2016. Constraint-based question answering with knowledge graph. In *COLING*.

Junwei Bao, Nan Duan, Ming Zhou, and Tiejun Zhao. 2014. Knowledge-based question answering as machine translation. In *ACL*.

Hannah Bast and Elmar Haussmann. 2015. More accurate question answering on freebase. In *CIKM*.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on Freebase from question-answer pairs. In *EMNLP*.

Jonathan Berant and Percy Liang. 2015. Imitation learning of agenda-based semantic parsers. *TACL*.

Phil Blunsom, Krystle Kocik, and James R. Curran. 2006. Question classification with log-linear models. In *SIGIR*.

Branimir Boguraev, Siddharth Patwardhan, Aditya Kalyanpur, Jennifer Chu-Carroll, and Adam Lally. 2014. Parallel and nested decomposition for factoid questions. *Natural Language Engineering*, 20(4).

Kurt Bollacker, Patrick Tufts, Tomi Pierce, and Robert Cook. 2007. A platform for scalable, collaborative, structured information integration. In *Intl. Workshop on Information Integration on the Web*.

Antoine Bordes, Nicolas Usunier, Sumit Chopra, and Jason Weston. 2015. Large-scale simple question answering with memory networks. In *arXiv*.

Jacob Cohen. 1960. A coefficient of agreement for nominal scales. *Educational and psychological measurement*, 20(1).

Luciano Del Corro, Abdalghani Abujabal, Rainer Gemulla, and Gerhard Weikum. 2015. FINET: Context-aware fine-grained named entity typing. In *EMNLP*.

Li Dong, Furu Wei, Ming Zhou, and Ke Xu. 2015. Question answering over freebase with multi-column convolutional neural networks. In *ACL*.

Anthony Fader, Luke Zettlemoyer, and Oren Etzioni. 2013. Paraphrase-driven learning for open question answering. In *ACL*.

Zhiheng Huang, Marcus Thint, and Zengchang Qin. 2008. Question classification using head words and their hypernyms. In *EMNLP*.

Adam Lally, John M Prager, Michael C McCord, Branimir K Boguraev, Siddharth Patwardhan, James Fan, Paul Fodor, and Jennifer Chu-Carroll. 2012. Question analysis: How Watson reads a clue. *IBM Journal of Research and Development*.

Xin Li and Dan Roth. 2002. Learning question classifiers. In *COLING*.

Thomas Lin, Mausam, and Oren Etzioni. 2012. Entity linking at web scale. In *Joint Workshop on Automatic Knowledge Base Construction and Web-scale Knowledge Extraction*.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *ACL*.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. In *arXiv*.

J. William Murdock, Aditya Kalyanpur, Chris Welty, James Fan, David A Ferrucci, DC Gondek, Lei Zhang, and Hiroshi Kanayama. 2012. Typing candidate answers using type coercion. *IBM Journal of Research and Development*.

Deepak Ravichandran and Eduard Hovy. 2002. Learning surface text patterns for a question answering system. In *ACL*.

Denis Savenkov and Eugene Agichtein. 2016. When a Knowledge Base Is Not Enough: Question Answering over Knowledge Bases with External Text Data. In *SIGIR*.

Carlos Nascimento Silla and Alex Alves Freitas. 2011. A survey of hierarchical classification across different application domains. *Data Mining and Knowledge Discovery*.

Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. 2007. YAGO: A core of semantic knowledge. In *WWW*.

Kun Xu, Siva Reddy, Yansong Feng, Songfang Huang, and Dongyan Zhao. 2016. Question answering on freebase via relation extraction and textual evidence. In *ACL*.

Mohamed Yahya, Klaus Berberich, Shady Elbassuoni, and Gerhard Weikum. 2013. Robust question answering over the Web of linked data. In *CIKM*.

Semih Yavuz, Izzeddin Gur, Yu Su, Mudhakar Srivatsa, and Xifeng Yan. 2016. Improving semantic parsing via answer type inference. In *EMNLP*.

Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. 2015. Semantic parsing via staged query graph generation: Question answering with knowledge base. In *ACL*.

Mohamed Amir Yosef, Sandro Bauer, Johannes Hoffart, Marc Spaniol, and Gerhard Weikum. 2012. HYENA: Hierarchical Type Classification for Entity Names. In *COLING*.