Query-Driven On-The-Fly Knowledge Base Construction

Dat Ba Nguyen
MPI for Informatics
datnb@mpi-inf.mpg.de

Abdalghani Abujabal
MPI for Informatics
abujabal@mpi-inf.mpg.de

Nam Khanh Tran
L3S Research Center
ntran@l3s.de

Martin Theobald
University of Luxembourg
martin.theobald@uni.lu

Gerhard Weikum
MPI for Informatics
weikum@mpi-inf.mpg.de

ABSTRACT

Today's openly available knowledge bases, such as DBpedia, Yago, Wikidata or Freebase, capture billions of facts about the world's entities. However, even the largest among these (i) are still limited in up-to-date coverage of what happens in the real world, and (ii) miss out on many relevant predicates that precisely capture the wide variety of relationships among entities. To overcome both of these limitations, we propose a novel approach to build *on-the-fly knowledge bases* in a query-driven manner. Our system, called QKBfly, supports analysts and journalists as well as question answering on emerging topics, by dynamically acquiring relevant facts as timely and comprehensively as possible. QKBfly is based on a *semantic-graph representation* of sentences, by which we perform three key IE tasks, namely *named-entity disambiguation*, *co-reference resolution* and *relation extraction*, in a light-weight and integrated manner. In contrast to Open IE, our output is canonicalized. In contrast to traditional IE, we capture more predicates, including ternary and higher-arity ones. Our experiments demonstrate that QKBfly can build high-quality, on-the-fly knowledge bases that can readily be deployed, e.g., for the task of ad-hoc question answering.

PVLDB Reference Format:
D. B. Nguyen, A. Abujabal, N. K. Tran, M. Theobald, and G. Weikum. Query-Driven On-The-Fly Knowledge Base Construction. *PVLDB*, 11 (1): 66-79, 2017.
DOI: 10.14778/3136610.3136616

1. INTRODUCTION

Motivation & Problem Setting. Knowledge bases (KBs) contain subject-predicate-object triples about entities and their properties. Popular KBs include DBpedia [2], Yago [51], Wikidata [54] and Freebase [7]. Their commercial counterparts at Google, Microsoft, Baidu, and others, provide back-end support for search engines, online recommendations, and various knowledge-centric services. They cover many millions of entities with billions of triples for thousands of predicates. However, despite this impressive size, no KB is ever complete. In fact, even the largest KBs miss out on many interesting predicates and emerging entities, such as brand-new events or unknown people, who suddenly become notable. As an example, consider what KBs provide about Brad Pitt: his birthplace, his movies, wives, children, etc. However, they do not point out which children have been adopted, nor that Angelina Jolie recently filed for divorce from him. There is even interesting information about his movies that is absent from all KBs. An example is that he played the mountaineer Heinrich Harrer in Seven Years in Tibet, which would ideally be captured as a quadruple ⟨`Brad_Pitt`, `plays_role_in`, `Heinrich_Harrer`, `Seven_Years_in_Tibet`⟩. These gaps cannot be easily filled, as many predicates are completely missing; and even for known predicates, it is hard to keep up with the pace at which new facts appear in the real world. This calls for a more open-ended and dynamic KB construction.

The goal of dynamic and broader construction of KBs has received substantial attention in the database research community recently. The DeepDive project [42, 48, 57] has developed a highly versatile tool suite for information extraction (IE) and KB population (KBP), based on Markov Logic [14] and further techniques, including a variety of optimizations. Another ground-breaking project in this space is SystemT [11, 12, 45], which uses declarative rules for IE in a wide range of applications, including enterprise content analytics. However, these prior works still require a specification of which predicates are of interest to the IE/KBP process. Unless predicates like `has_adopted_child`, `filed_divorce_from` or `plays_role_in` are made explicit by the application architect (or "knowledge engineer"), they will not be discovered automatically. This may not be a problem for most use cases in enterprises or data science, but it does limit the ability of these approaches to extract knowledge without any prior setup phase.

State-of-the-Art & Limitations. The field of Open IE [4, 35] partially addresses the task of on-the-fly KB construction. In Open IE, however, the subject-predicate-object arguments of the extracted triples are usually not canonicalized. For example, triples with subjects "Brad Pitt", "Bradley Pitt", "Oscar winner Pitt", etc. will all be present even if their statements are equivalent. The DEFIE [8] system, for example, thus adds a post-processing stage to

Proceedings of the VLDB Endowment, Vol. 11, No. 1

DOI: 10.14778/3136610.3136616

disambiguate entity names, but still leaves predicates unresolved. Predicates like `wins_prize` and `receives_award` will co-exist, although they are synonymous. Recently, [22, 47] further canonicalized Open IE output by clustering noun phrases as subjects and objects, while verbal phrases are clustered into relations. However, these approaches have high overhead and are not geared for dynamic knowledge acquisition. Declarative approaches to IE and KBP, such as DeepDive [42, 48, 57] and SystemT [11, 12, 45], require specifications of predicates and rules. Thus, they cannot be used in a spontaneous "on-the-fly" manner. Query-time IE, as pursued in our work, resembles the notion of query-time inference over probabilistic databases [18, 53]. These methods operate on uncertain relational data as well as uncertain rules, and support flexible forms of top-k queries [17] and general inference [23, 24, 31]. However, all of these approaches require a relational schema that underlies the KB.

Approach & Contributions. This paper presents QKBfly, a novel system for constructing query-driven, on-the-fly KBs. Based on our experience with various IE tasks [26, 38, 40, 41, 56], our focus in this work is to develop an end-to-end system for KB construction, which may be triggered by an ad-hoc user query (e.g., when an analyst or journalist becomes interested in a particular person, organization or event). The system takes as input an entity-centric query or a natural-language question, automatically retrieves relevant source documents (via Wikipedia and news sources), runs a novel form of knowledge extraction on the sources, and builds a high-coverage KB that is focused on the entities of interest. Compared to mainstream KBs, we acquire facts for a much larger set of predicates. Compared to Open IE methods, arguments of facts are canonicalized, thus referring to unique entities with semantically typed predicates which are derived from precomputed clusters of phrases. Besides supporting analytical queries, QKBfly thus also facilitates the application of current question-answering (QA) frameworks [6, 5, 55], which increasingly rely on structured knowledge backends, to currently popular events and queries.

At the heart of QKBfly is a *semantic-graph* representation of sentences that captures per-sentence clauses, noun-phrases, pronouns, as well as their syntactic and semantic dependencies. Based on this graph, we devise an efficient inference technique that performs three key IE tasks, namely *named-entity disambiguation*, *entity co-reference resolution* and *relation extraction*, in a light-weight and integrated manner. Because of the clause-based representation of sentences, QKBfly is not limited to binary predicates but can also extract ternary (or higher-arity) predicates. To conclude our motivation for this work, we summarize the novel contributions of QKBfly as follows:

- we present an end-to-end system for on-the-fly KB construction that is triggered by an entity-centric user query or a natural-language question;
- QKBfly employs a novel graph-based approach for cleaning, canonicalizing and organizing noisy extractions from Open IE into a crisp KB;
- we conduct extensive experiments that demonstrate the viability of our approach under various IE and QA settings.

In our experiments, we evaluate QKBfly's capability of building on-the-fly KBs against the state-of-the-art baselines DEFIE [8] and DeepDive [48]. As an extrinsic use case, we employ QKBfly for ad-hoc QA on emerging topics derived from Google Trends.

2. DESIGN RATIONALE AND OVERVIEW

2.1 Design Space and Choices

KB construction generally faces an inherent *trade-off* between precision (i.e., fraction of correct tuples among the acquired ones) and recall (i.e., fraction of correct tuples among the ones that could possibly be acquired from the input). In traditional KB construction, the priority is usually precision, since large KBs (e.g., commercial knowledge graphs, DBpedia, Yago, Wikidata, Freebase, etc.) are an infrastructure asset meant to support a wide variety of applications. In contrast, on-the-fly KB building is intended to support analysts in ad-hoc exploration and querying. Therefore, recall is the primary priority, and good precision is a secondary goal within this regime.

This overriding design decision has consequences on the system architecture. While holistic methods with joint inference on all steps and sub-goals (e.g., probabilistic graphical models, constraint-based reasoners, etc.) are often attractive, they are much harder to control in their behavior towards separately tunable precision and recall. Moreover, tools like Alchemy [14], DeepDive [42, 48], Markov-TheBeast [46] or Sofie [52] require sophisticated modeling, training and configuration upfront, which is all but straightforward in our open-domain on-the-fly setting. These considerations are the rationale for splitting our approach to on-the-fly KB construction into two phases: a recall-oriented extraction phase followed by a precision-oriented cleaning phase. This separation gives us best control on the trade-off.

Extraction. Since on-the-fly KB construction aims for high recall, we adopt the Open IE paradigm [4] for this phase. To cope with the high diversity of input documents that a query-driven approach comes with, we employ a judiciously designed set of linguistic pre-processing steps. For these, we use standard tools that are state-of-the-art in NLP. One of these is ClausIE [13], which decomposes sentences into a set of clauses. For efficiency, we modified ClausIE, to use the MaltParser [43] instead of its original reliance on the Stanford parser [28]. In our experiments, we compare QKBfly against a variety of best-practice Open IE tools.

Cleaning. The extraction phase produces a large set of – still noisy – candidates for triples and tuples. To remove false positives and reconcile semantic redundancy, we perform two major cleaning steps: one resolving entity mentions and co-references (see Section 4), and one for canonicalizing relational predicates (see Section 5). Both are potentially expensive tasks. Since the case for on-the-fly KB construction are ad-hoc information needs that should support analysts in a same-day manner, we decided to devise light-weight algorithms for both tasks (see below), based on a graph model, but avoiding the heavy-duty joint inference that probabilistic graphical models (PGMs) usually incur. Our experiments provide some comparison points for precision and run-time of our method against the joint-inference paradigm. As MAP inference ("maximum a posteriori") in PGMs is related to solving a weighted MaxSat problem, which in turn is a form of constraint programming, we picked the Integer Linear Programming (ILP) solver Gurobi for comparison. This is a mature and highly optimized tool,

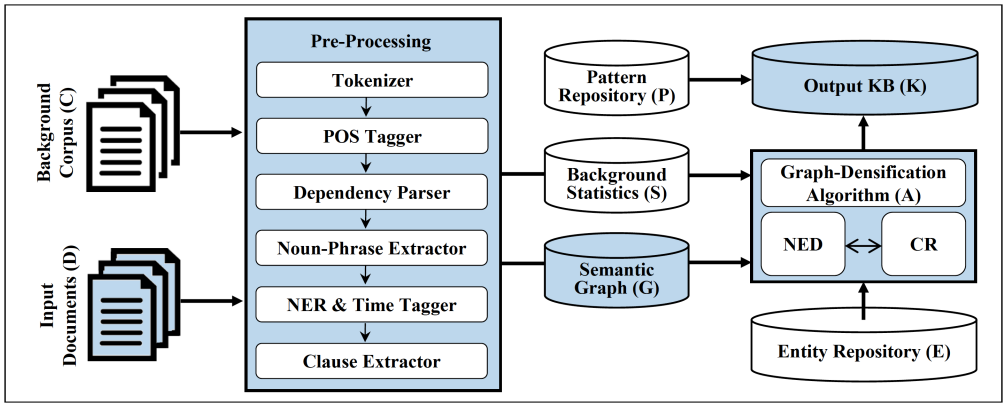

Figure 1: QKBfly overview. Blue-shaded components are processed on-the-fly.

performing very well on a wide range of constraint-based reasoning tasks. Note that some PGMs actually use ILP for efficient MAP inference (e.g., [46]).

2.2 QKBfly Overview

Figure 1 depicts the architecture of QKBfly. Given a set of *input documents* (D), retrieved in response to a *user query* (Q), QKBfly works in three stages. First, it builds a *semantic graph* (G) from clauses and initial co-references extracted from each of the sentences contained in (D). Second, it refines each semantic graph by jointly performing named-entity disambiguation and co-reference resolution via a *graph-densification algorithm* (A). Third, it canonicalizes the *on-the-fly KB* (K) by merging co-reference nodes and by mapping relational paraphrases to a canonical set of entities and relations.

Background Repositories. As static input, QKBfly employs three types of repositories, namely a *background corpus* (C), a *pattern repository* (P) and an *entity repository* (E). From (C), we also extract *background statistics* (S) used later by QKBfly. Each of these background repositories is exchangeable. For the experimental setup described in Section 7, we fixed them as follows: we employ a Wikipedia[1] full-text dump from Sep. 1, 2015 as (C), PATTY[2] [38] consisting of 127,811 relational paraphrases as (P), and Yago[3] [51] with 3,420,126 known entities as (E). As for Yago, we merely harness its knowledge about alias names of entities together with their gender attributes (for better pronoun resolution), while none of the actual KB facts are used in QKBfly. In particular, we do not require all entities we recognize during KB construction to be present in the given entity repository.

Statistics. Both the background corpus (C) and the input documents (D) are pre-processed by a pipeline of linguistic tools, consisting of tokenization, part-of-speech (POS) tagging, noun-phrase chunking and named-entity recognition (NER), all of which are performed by the Stanford CoreNLP toolkit [34]. In addition, we employ time tagging [10] and the Open-IE-tool ClausIE [13] to extract clause structures in which all arguments are annotated either as names or time expressions. As for the Wikipedia-based background corpus (C), we also map clause components to Wikipedia entities by their *href* links (if the NER type of the clause component matches the type of Wikipedia entity). Based on the resulting clause structures, we compute (co-)occurrence statistics for clause-argument-types and the relationships among them. These serve as input to the feature functions described in Section 4.

Stage 1: Semantic Graph. From the pre-processed input documents (D), we build a semantic graph for each sentence in (D) based on the clause structure detected by ClausIE. A leaf node in this graph represents an occurrence of an entity, while an edge among two leaf nodes represents a relation pattern in our on-the-fly KB (K). The per-sentence graphs are linked via an initial set of possible co-reference edges (based on the technique of [3]), thus connecting pairs of leaf nodes that potentially refer to the same entity (see Section 3).

Stage 2: Graph Algorithm. Next, the graph-densification algorithm refines each of the connected semantic graphs. It thereby jointly performs entity disambiguation and co-reference resolution based on an efficient algorithm for densifying the semantic graphs. The method is inspired by and generalizes the dense-subgraph algorithm introduced in [27], which we judiciously chose as a basis due to its good runtime performance and accuracy. The algorithm employs a greedy heuristic for pruning edges to obtain a dense subgraph under a set of given constraints. The remaining edges in the densified subgraph link mentions to unique entities in the resulting on-the-fly KB (see Section 4).

Stage 3: On-the-fly KB Canonicalization. In the final stage, QKBfly constructs the on-the-fly KB (K) by combining and canonicalizing the remaining nodes and edges in the semantic graph. Entities are either linked to the entity repository (E) or are identified as a cluster of new names (for emerging entities) which are connected by co-reference edges. At this stage, QKBfly uses the pattern repository (P) to map relational paraphrases to a canonicalized set of relations. Similarly to the entity repository, new relational paraphrases not contained in (P) are considered as new relations. Moreover, by considering the per-sentence clause structure, QKBfly is able to acquire triples as well as higher-arity facts (see Section 5).

[1] `https://dumps.wikimedia.org/enwiki/`
[2] `https://d5gate.ag5.mpi-sb.mpg.de/pattyweb/`
[3] `https://www.yago-knowledge.org/`

3. SEMANTIC GRAPH

When given a set of natural-language sentences (e.g., a Web page, a Wikipedia article, etc.) as input, QKBfly first builds one semantic graph for each input sentence. It then connects the per-sentence graphs by additional co-reference links among nodes that potentially refer to the same entity. QKBfly primarily employs ClausIE to construct these per-sentence graphs. To improve the efficiency of the extraction process, we use the MaltParser [43] in our implementation instead of the Stanford parser [28] used in the original version of ClausIE. Besides dependency parsing, ClausIE exploits further linguistic features such as POS tagging and noun-phrase chunking to detect so-called *clauses*.

Following [44], a clause is a coherent piece of information within a sentence that consists of one *subject* (S), one *verb* (V), an optional (either direct or indirect) *object* (O), an optional *complement* (C), and a variable amount of *adverbials* (A). A main observation of [44] is that only seven combinations of the above constituents, namely SV, SVA, SVC, SVO, SVOO, SVOA and SVOC, actually occur in the English language, which is a key for relation extraction in ClausIE. That is, one clause confirms to exactly one n-ary fact that has these constituents as arguments. In addition, we use the Stanford NER tagger [34] and the SUTime annotator [10] to detect named entities and time expressions within the clauses.

Nodes. A node in our semantic graph is a container for *clauses*, *noun-phrases*, *pronouns* and *entities* occurring in an input sentence and the entity repository, respectively. Specifically, we distinguish the following four types of nodes:

- A `clause` node is generated for each clause detected by ClausIE. A clause may be connected to multiple dependent clauses in the same sentence. Their dependency structure is also detected by ClausIE.
- A `noun-phrase` node is generated for each noun phrase detected by the noun-phrase chunker and for each named entity detected by the NER tagger (both using [34]). Additional time expressions are detected by the time tagger (using [10]).
- A `pronoun` node is generated for a pronoun (such as "*he*", "*she*", etc.). Noun-phrases and pronouns together form the leaves of the subtree built for each sentence.
- An `entity` node is generated for each entity candidate (e.g., `Brad_Pitt`) of a `noun-phrase` node that matches a known alias name in the entity repository.

Edges. Edges represent the syntactic and semantic *dependencies* among nodes in the semantic graph. Here, we distinguish the following four types of edges:

- A `depends` edge links two dependent clauses. It additionally links a `clause` node with the `noun-phrase` and `pronoun` nodes it contains.
- A `relation` edge represents a relation pattern (i.e., the lemmatized verb (V) constituent of the clause with an optional preposition such as "*to*", "*in*", etc.) that connects two `noun-phrase` or `pronoun` nodes in a clause.
- A `sameAs` edge links two `noun-phrase` or `pronoun` nodes which likely refer to the same entity (by following [3] for co-reference and pronoun resolution).
- A `means` edge links a `noun-phrase` or `pronoun` node with an `entity` node based on matching alias names in the entity repository.

We follow the method of [3] to initialize the `sameAs` edges between `noun-phrase` nodes and `pronoun` nodes, respectively. The `sameAs` edges among two `noun-phrase` nodes with the same NER label (e.g., `PERSON`) are determined by string matching (e.g., between "*Brad Pitt*" and "*Pitt*"). Additionally, `sameAs` edges are created between `pronoun` nodes and all `noun-phrase` nodes that precede the pronoun by at most five backward sentences. Our graph algorithm (see Section 4) will later remove all but the most likely `sameAs` edge between a `pronoun` node and its linked `noun-phrase` nodes. In addition to the verb (V) constituents detected by ClausIE, we apply one more heuristic to label `relation` edges. That is, for text patterns of the form "*'s ⟨noun⟩*" (e.g., "*Pitt's ex-wife Angelina Jolie*"), we consider the middle noun (i.e., "*ex-wife*") as the relation candidate between the two `noun-phrase` nodes.

Figure 2 depicts a semantic graph built from the two input sentences shown on top of the figure. The first sentence contains an SVC and an SVO clause, namely *"Brad Pitt is an actor"* and *"he supports the ONE Campaign"*, which results in two triples ⟨ *"Brad Pitt"*, *"be"*, *"actor"*⟩ and ⟨ *"he"*, *"support"*, *"ONE Campaign"*⟩ whose arguments are not yet canonicalized. Similarly, the quadruple ⟨ *"Pitt"*, *"donate to"*, *"$100,000"*, *"Daniel Pearl Foundation"*⟩ is extracted from the SVOO clause of the second sentence. The noun phrases *"actor"* and *"$100,000"* could not be linked to any entity in the entity repository. These will remain string literals in the respective arguments of the former two facts.

4. GRAPH ALGORITHM

We henceforth refer to the semantic graph built according to the previous section as $\mathcal{G} = (\mathcal{N}, \mathcal{R})$, where $\mathcal{N}$ denotes the set of nodes, and $\mathcal{R}$ denotes the set of edges (i.e., "relationships") among nodes in $\mathcal{G}$. The goal of our graph-densification algorithm then is to remove false-positive `means` and `sameAs` edges from $\mathcal{R}$ by solving a *constraint-based optimization* problem. In doing so, we perform a form of *joint inference* for the two key IE tasks of *named-entity disambiguation* and *co-reference resolution*.

Edge Weights. For a subgraph $\mathcal{S} = \langle \mathcal{N}' \subseteq \mathcal{N}, \mathcal{R}' \subseteq \mathcal{R} \rangle$ of $\mathcal{G}$, we first distinguish the following dependencies among nodes.

- For each `noun-phrase` node n_i, let $ent(n_i, \mathcal{S})$ be the set of all `entity` nodes linked to n_i by `means` edges in $\mathcal{R}'$.
- For each `pronoun` node p_i, let $np(p_i, \mathcal{S})$ be the set of all `noun- phrase` nodes linked to p_i by `sameAs` edges in $\mathcal{R}'$.
- Further, let $ent(p_i, \mathcal{S})$ denote the union of all $ent(n_t, \mathcal{S})$ sets, where $n_t \in np(p_i, \mathcal{S})$.

Next, we define the edge weights to establish our densest-subgraph objective as follows.

(1) The weight of a `means` edge between a `noun-phrase` node n_i and an `entity` node $e_{ij} \in ent(n_i, \mathcal{S})$ for a subgraph $\mathcal{S}$ is computed as

$$w(n_i, e_{ij}) = \alpha_1 \cdot prior(n_i, e_{ij}) + \alpha_2 \cdot sim\big(cxt(n_i), cxt(e_{ij})\big)$$

where α_1 and α_2 are hyper-parameters and:

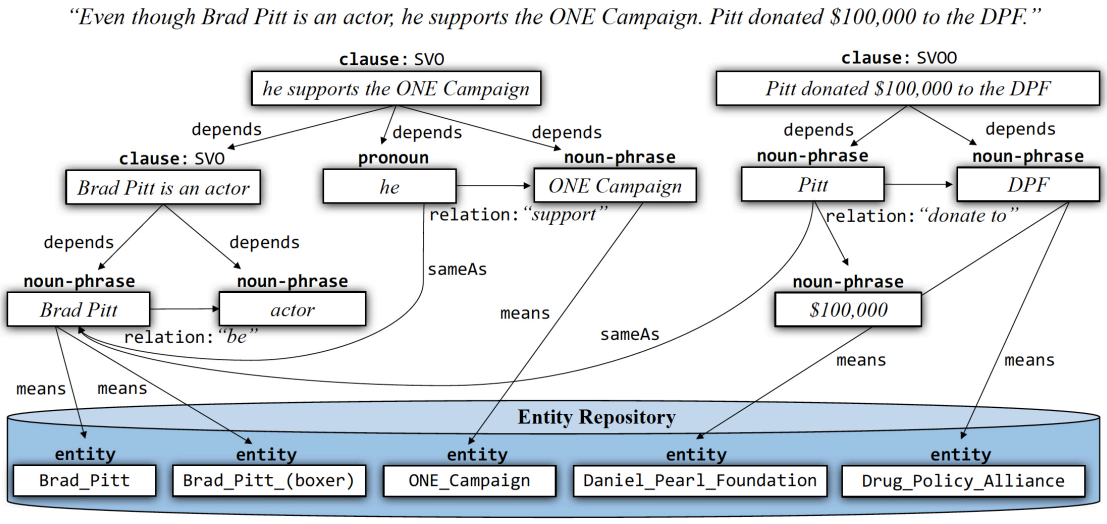

Figure 2: Semantic graph example.

- $prior(n_i, e_{i_j})$ captures the prior probability of a noun-phrase n_i denoting an entity candidate e_{i_j} with respect to the link structure of Wikipedia. Following recent entity disambiguation approaches [27, 36, 41], we compute this prior as the relative frequency under which a link with anchor n_i points to a Wikipedia article that represents entity e_{i_j}. The respective counts are obtained from the background corpus. Each such count is normalized by the number of times e_{i_j} occurs as an anchor among all links in the same corpus.
- $sim(cxt(n_i), cxt(e_{i_j}))$ captures the similarity of two context vectors. The context vector of a noun-phrase, $cxt(n_i)$, is built from tokens in the sentence that contains this noun phrase, while the context vector of an entity, $cxt(e_{i_j})$, is built from tokens occurring in the entity's Wikipedia article. Both vectors are weighted using the common TF-IDF scheme, which assigns the product of term frequency and inverse document frequency to each dimension in the vectors. As for similarity measure, we employ the weighted overlap coefficient between two vectors $cxt(n_i) = \langle v_1, v_2, \dots \rangle$ and $cxt(e_{i_j}) = \langle v'_1, v'_2, \dots \rangle$:

$$sim\big(cxt(n_i), cxt(e_{i_j})\big) = \frac{\sum_k \min(v_k, v'_k)}{\min\big(\sum_k v_k, \sum_k v'_k\big)}$$

(2) The weight of a `relation` edge between two `noun-phrase` or `pronoun` nodes n_i, n_t for a subgraph $\mathcal{S}$ is computed as

$$w(n_i, n_t, \mathcal{S}) = \alpha_3 \cdot \sum_{\substack{e_{i_j} \in ent(n_i, \mathcal{S}) \\ e_{t_k} \in ent(n_t, \mathcal{S})}} coh(e_{i_j}, e_{t_k}) + \alpha_4 \cdot \sum_{\substack{e_{i_j} \in ent(n_i, \mathcal{S}) \\ e_{t_k} \in ent(n_t, \mathcal{S})}} ts(e_{i_j}, e_{t_k}, r_{i,t})$$

where α_3 and α_4 are hyper-parameters and:

- $coh(e_{i_j}, e_{t_k})$ captures the coherence between two entity candidates. The coherence is computed as the weighted overlap similarity between the entities' context vectors.
- $ts(e_{i_j}, e_{t_k}, r_{i,t})$ is the relative frequency under which the semantic types of e_{i_j}, e_{t_k} occur under the relation pattern $r_{i,t}$ in the clauses detected by ClausIE. Since an entity can have several types (e.g., `ACTOR` and `PERSON`), we take the sum over all type combinations for the given entity pair.

Type Signatures. To compute the weights for the above type signatures, we run the Stanford NER tagger, SUTime, and ClausIE on all sentences in the background corpus. Subsequently, we compute (co-)occurrence statistics for the types of arguments and the relation patterns on clauses in which all arguments are mapped to Wikipedia entities, or are recognized as either names or time expressions. In addition to the five general NER types including `PERSON`, `ORGANIZATION`, `LOCATION`, `MISC` and `TIME`, we use an extended type system based on frequently used infobox templates[4]. In our experiments, we only use prominent types that have at least 1,000 articles as instances in Wikipedia, thereby reducing bias towards entities with very many fine-grained types. From the resulting 167 types, we manually built a subsumption hierarchy (e.g., `FOOTBALLER` ⊆ `ATHLETE` ⊆ `PERSON`, etc.).

We remark that we do not compute weights for `sameAs` and `depends` edges. While `sameAs` edges are used as constraints in the optimization model, while `depends` edges only contribute to the final KB construction (particularly for determining the fact boundaries as described in Section 5).

Optimization Objective. After assigning the edge weights, we next aim to find the densest subgraph $\mathcal{S}^* = \langle \mathcal{N}^* \subseteq \mathcal{N}, \mathcal{R}^* \subseteq \mathcal{R} \rangle$ that maximizes the following objective function.

- The sum of all edge weights in subgraph $\mathcal{S}^*$, denoted as $W(\mathcal{S}^*)$, is maximized,

which is subject to the following constraints:

(1) each `noun-phrase` node is connected to at most one `entity` node by a `means` edge in $\mathcal{R}^*$;

(2) each `pronoun` node is connected to at most one `noun-phrase` node by a `sameAs` edge in $\mathcal{R}^*$;

[4] `https://en.wikipedia.org/wiki/Wikipedia:List_of_infoboxes`

(3) all noun-phrase or pronoun nodes that are mutually linked by sameAs edges, are connected to the same entity;

(4) each pronoun node connected to a noun-phrase node that is connected to an entity node of type PERSON, for which the background KB provides gender information, must match that gender.

Approximation Algorithm. [50] show that the above formulation of a densest-subgraph problem with constraints is NP-hard. We thus resort to approximating our optimization objective by the following greedy algorithm. We first restrict the entity candidates for each noun-phrase node to the ones contained in the dictionary of the background KB, but allow unlinked noun-phrase nodes in the final subgraph for *out-of-KB* entities. For all noun-phrase nodes that are mutually connected via sameAs edges, the entity candidate sets are intersected. The approximation algorithm then greedily iterates on the graph's edges as long as none of the above constraints is violated. In each round, the algorithm removes the means or sameAs edge between two nodes (x, y) with the smallest contribution to the objective function of the current subgraph $\mathcal{S}$. This contribution to the objective function is defined as

$$c(x, y, \mathcal{S}) = W(\mathcal{S}) - W(\mathcal{S}')$$

where $\mathcal{S}'$ is the subgraph of $\mathcal{S}$ we obtain by removing edge (x, y) from $\mathcal{S}$. If removing an edge leaves an entity candidate isolated, then that entity node is removed as well. After each edge removal, the weights of all remaining edges need to be recomputed, as the cutting of sameAs edges modifies the influence of relation edges attached to noun-phrase or pronoun nodes. Our implementation performs this recomputation step in a selective and incremental way. Algorithm 1 shows pseudocode for our algorithm.

Confidence Scores. As an additional filtering step, we assign a normalized confidence score to each noun-phrase or pronoun node n_i that is disambiguated to an entity e_{i_j} as

$$score(n_i, e_{i_j}, \mathcal{S}^*) = \frac{c(n_i, e_{i_j}, \mathcal{S}^*)}{\sum_{e_{i_t} \in ent(n_i, \mathcal{G})} c(n_i, e_{i_t}, \mathcal{S}_t)}$$

where the denominator sums up over all subgraphs $\mathcal{S}_t$ constructed from $\mathcal{S}^*$ by replacing e_{i_j} (and its associated means edge) with one of the original candidates e_{i_t} (and its means edge). For the confidence of an extracted triple (or higher-arity fact), we choose the minimum of the confidence scores of all disambiguated entities that occur as the arguments of the fact. In our experiments, we use a score threshold $\tau = 0.5$ to distill high-quality facts.

Hyper-Parameter Tuning. We manually annotated 162 sentences from 5 Wikipedia articles about prominent person entities, including Andrew Ng, Angela Merkel, David Beckham, Larry Page and Paris Hilton (thus covering scientists, politicians, sports stars, business people and models). These annotations comprise 203 facts, each consisting of a pair of Yago entities and a relation pattern (e.g., ⟨Larry_Page, "*born in*", Michigan⟩). By independently constructing a graph $\mathcal{G}$ with two noun-phrase nodes n_i and n_t for each extracted fact, we define the probability of choosing entity node $e_{i_j} \in ent(n_i, \mathcal{G})$ and entity node $e_{t_k} \in ent(n_t, \mathcal{G})$ as

$$prob(n_i, e_{i_j}, n_t, e_{t_k}, \mathcal{G}) = \frac{W(\mathcal{S})}{W(\mathcal{G})}$$

Algorithm 1: Densest-Subgraph Algorithm.

Data: Semantic graph $\mathcal{G} = \langle \mathcal{N}, \mathcal{R} \rangle$ from the input text.
Result: The densest subgraph $\mathcal{S}^* = \langle \mathcal{N}^* \subseteq \mathcal{N}, \mathcal{R}^* \subseteq \mathcal{R} \rangle$ which satisfies the four constraints (1), (2), (3) and (4).

```
for subgraph S = ⟨N' ⊆ N, R' ⊆ R⟩ do
    for node x ∈ N' do
        np(x) ← all noun-phrase nodes linked to x by sameAs edges in R';
    for noun-phrase node n_i ∈ N' do
        ent(n_i) ← all entity nodes linked to n_i by means edges in R';
        if ent(n_i) = ∅ then
            report a new entity;
    for noun-phrase node n_i ∈ N' do
        for noun-phrase node n_t ∈ np(n_i) do
            ent(n_i) ← ent(n_i) ∩ ent(n_t);
    for pronoun node p_i ∈ N' do
        ent(p_i) ← ∪_{n_t ∈ np(p_i)} ent(n_t);
        for entity e ∈ ent(p_i) do
            if doesn't satisfy gender constraint (4) then
                remove e from ent(p_i);
    W(S) ← sum over all edge weights in S;
/* start with the original graph */
S ← G;
while all constraints are satisfied do
    for means or sameAs edge of two node (x, y) ∈ R' do
        S' ← subgraph of S by removing (x, y);
        c(x, y, S) = W(S) − W(S');
    find and remove the means or sameAs edge between
    two node (x, y) with the smallest contribution
    c(x, y, S) and no constraint will be violated;
    update S;
    if no edge is removed then
        S* ← S;
        return S*;
```

where the subgraph $\mathcal{S}$ is constructed from $\mathcal{G}$ by removing all entity nodes except e_{i_j} and e_{t_k}. Finally, the $\alpha_{1..4}$ parameters are learned by maximizing the probability of the ground-truth annotations using L-BFGS optimization [33], which implements a memory-efficient variant of stochastic gradient descent.

5. ON-THE-FLY KB CANONICALIZATION

In the final stage, QKBfly processes the output of the graph densification to perform our final IE task, *relation extraction*, to populate our on-the-fly KB with facts. Noun-phrases and pronouns at this time are either linked to unique entities in the entity repository or are identified by a group of noun-phrase nodes connected via sameAs edges. Specifically, a new entity is introduced for each group of sameAs nodes that consists only of out-of-repository names. We additionally consider all groups of noun-phrase nodes that link to entities in the background repository with very low confidence scores as new entities. These are added to the on-the-fly KB as emerging entities—an important asset for up-to-date knowledge.

Relation edges carry surface-form labels: patterns that denote predicates. To canonicalize also these patterns, we harness the pattern repository, specifically the PATTY dictionary of relational paraphrases [38]. All node-edge-node triples that have the same node labels and have edge labels that belong to the same synset in PATTY are combined into a single triple, thereby clustering the relation patterns together. For example, relation edges with labels "*play in*", "*act in*" and "*star in*" that connect the same actor-movie pair are combined. This way, we are not limited to the relations that are registered in an existing KB (such as the ca. 100 predicates in Yago or the ca. 6,000 property labels in DBpedia), but can also capture many interesting relations on-the-fly. Unlike all of the major KBs, we also construct facts for ternary or higher-arity relations. This is where the extraction of clauses pays off. Whenever `noun-phrase` or `pronoun` nodes are linked to the same `clause` node via `depends` edges, we merge those nodes into a single fact. For example, QKBfly can construct ternary facts from SVOO or SVOA clauses such as ⟨`Brad_Pitt`, `play_in`, `Heinrich_Harrer`, `Seven_Years_In_Tibet`⟩, ⟨`Brad_Pitt`, `adopt_in`, `Pax_Thien_Jolie-Pitt`, *"2008"*⟩, etc. Those higher-arity facts provide more complete information than triple facts, which is useful for many extrinsic use cases such as QA on complex questions (e.g., "*Who plays Achilles in Troy?*").

Table 1 shows sample results (entities and their mentions, relations and patterns, as well as facts) of the on-the-fly KB constructed by QKBfly from the Wikipedia page of Brad Pitt. QKBfly captures long-tail entities such as Brad Pitt's father `William_Alvin_Pitt`, who is missing in most KBs. QKBfly captures binary facts such as ⟨`Brad_Pitt`, `born_to`, `William_Alvin_Pitt`⟩, as well as ternary facts such as ⟨`Brad_Pitt`, `play_in`, `Achilles`, `Troy_(film)`⟩.

Table 1: Excerpt of QKBfly output on Brad Pitt page. Out-of-Yago entities are with an asterisk.

Entities & Mentions
`Brad_Pitt` → *"William Bradley Pitt"*, *"Brad Pitt"*, etc.
`William_Alvin_Pitt`* → *"William Alvin Pitt"*
`Achilles` → *"Achilles"*, *"warrior Achilles"*, etc.
`Troy_(film)` → *"Troy"*
Relations & Patterns
`born_to` → *"born to"*, *"father"*, etc.
`play_in` → *"act in"*, *"star in"*, *"play in"*, *"have role in"*, etc.
Facts
⟨`Brad_Pitt`, `born_to`, `William_Alvin_Pitt`*⟩
⟨`Brad_Pitt`, `play_in`, `Achilles`, `Troy_(film)`⟩

Table 2: Excerpt of QKBfly output from news articles. Out-of-Yago entities are with an asterisk.

Query	**Fact**
Brad Pitt	⟨`Brad_Pitt`, `divorce_from`, `Angelina_Jolie`⟩
	⟨`Angelina_Jolie`, `file_for_on`, *"divorce"*, *"September 19, 2016"*⟩
Bob Dylan	⟨`Bob_Dylan`,`win_for`,`Nobel_Prize_in_Literature`, *"having created new poetic expressions within the American song tradition"*⟩
Donald Trump	⟨`Jessica_Leeds`*, `accuse_of`, `Donald_Trump`, *"groping her on an airplane in the 1980s"*⟩

As a demonstration of QKBfly's ability to compile KBs in a query-driven manner, we ran various queries over news articles returned by the Google search engine. Table 2 shows sample results extracted from top-ranked news about celebrities. QKBfly compiles up-to-date knowledge like the Pitt and Jolie divorce, and the Nobel prize for Bob Dylan. Additionally, QKBfly captures emerging entities (i.e., entities not contained in the given entity repository) such as the woman accusing Donald Trump of sexual abuse: Jessica Leeds.

6. QKBfly AT WORK

We developed a user interface to demonstrate the ability of QKBfly to construct KBs on-the-fly. Given a set of input documents, such as a Wikipedia page or a collection of news articles, QKBfly can build a KB with hundreds to a few thousands of facts within a minute.

System Implementation. Figure 3 and Figure 4 show two screenshots of QKBfly in a browser.

- For the input, we let the user choose a query (e.g., "*Bob Dylan*"), the input source (Wikipedia or news articles) and the desired number of input documents. QKBfly processes relevant documents by restricting the search to `en.wikipedia.org` for Wikipedia, and to `bbc.com` for news articles.
- For the output, we show all facts from the on-the-fly KB. Prominent entities may be linked to entities in the entity repository. As the number of facts can be huge, we offer a string search on subjects, predicates or objects. QKBfly also supports *type* search if the user specifies the prefix `Type:` to the queried category. Figure 3 shows the result of 4 out of 721 facts in total by searching for `Type:MUSICAL_ARTIST` as the subject and `receive_in_from` as the predicate in the on-the-fly KB constructed from the Wikipedia page of Bob Dylan.

Figure 4 also illustrates the ability of QKBfly to capture up-to-date facts from news. For example, there is the fact that Patti Smith forgot the lyrics when performing Bob_Dylan's song at the Nobel Prize ceremony, which is extracted from 10 news articles. The predicate `forget` would not be covered by many of the existing KBs. Also, obtaining this kind of knowledge is not possible with state-of-the-art tools for IE and knowledge base population. Methods, like DeepDive or SystemT, would require substantial setup work by a knowledge engineer to obtain these results.

7. EXPERIMENTS

In our experiments, we first compare QKBfly's capability of building on-the-fly KBs against the state-of-the-art Open IE baseline DEFIE [8]. Second, we compare our greedy approximation algorithm for the joint inference of named entity disambiguation (NED) and co-reference resolution (CR) against an integer linear programming algorithm. Third, we compare QKBfly's capability of performing mentions of spouses extraction (i.e., the `married_to` relation) against DeepDive [57]. Finally, as an extrinsic use case, we dynamically construct ad-hoc KBs for question answering.

7.1 Experiments on KB Construction

Benchmarks. We focus on two datasets for our experiments as follows:

Figure 3: Sample of higher-arity facts extracted by QKBfly from Wikipedia.

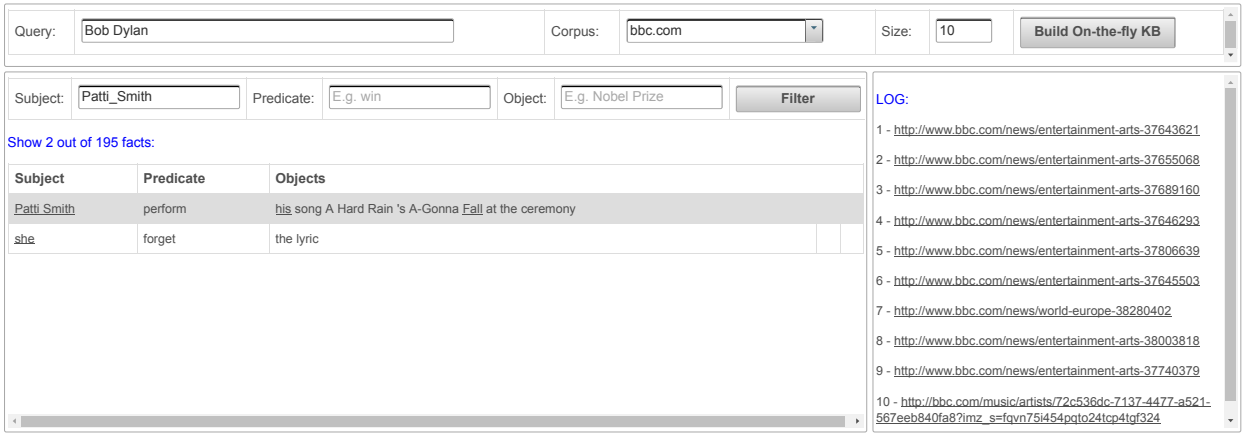

Figure 4: Sample of up-to-date facts extracted by QKBfly from news articles.

- DEFIE-Wikipedia dataset[5] [8], consisting of 14,072 randomly chosen Wikipedia pages with 225,867 sentences. We use this dataset to run experiments on end-to-end KB construction.
- Reverb dataset[6] [20], consisting of 500 sentences which have been obtained by the random-link service of Yahoo. This is used to run experiments on Open IE components.

Methods under Comparison. We compare several configurations of QKBfly against DEFIE:

- QKBfly jointly performs fact extraction, NED and CR.
- QKBfly-pipeline is a pipeline architecture with three separate stages for fact extraction, NED and CR. The type signature feature is omitted.
- QKBfly-noun only performs fact extraction and NED. CR is omitted.
- DEFIE is a pipeline architecture with two stages for Open IE and NED, using Babelfy [36] for NED.

Environment. In order to have a fair comparison among systems, all experiments are run single-threaded on an Intel Xeon X5650 server with 64GB RAM.

[5] provided by the authors
[6] `http://reverb.cs.washington.edu/`

Assessment. We asked 2 human assessors to evaluate the correctness of 200 randomly sampled extractions. Inter-assessor agreement was high, with Cohen's kappa being $\kappa = 0.7$. Precision values are reported with Wald confidence intervals at 95%.

Results on Fact Extraction. Table 3 shows experimental results for fact extraction on the DEFIE-Wikipedia dataset. QKBfly-noun achieves the highest precision: 73% for triple facts and 68% for quadruple facts. QKBfly significantly increases the number of extractions, with a relatively small loss in precision. This suggests that our method works fairly well for co-references. Compared to the pipeline architecture, the joint model of QKBfly increases precision by 5%. All QKBfly variants significantly outperform DEFIE in terms of both precision and absolute recall (i.e., the number of exractions). DEFIE has been optimized for short sentences (i.e., definitions) and loses effectiveness when processing complex texts with subordinate clauses and co-references. Additionally, DEFIE only yields triples, whereas QKBfly returns a large number of higher-arity facts with good precision.

In terms of runtime efficiency, all QKBfly variants perform similarly, less than a second for processing one document. Almost half of the runtime is for pre-processing via the Stanford CoreNLP pipeline and the MaltParser. Thus, all approaches, including the joint models, are efficient and scale to processing large input corpora on the fly. Note that

Table 3: Experimental results on fact extraction (at 95% confidence intervals).

Method	Triple Facts		Higher-arity Facts		Avg. Run-time (s) per Document
	Precision	#Extractions	Precision	#Extractions	
DEFIE	0.62 ± 0.06	39,684	—	—	unknown
QKBfly	0.67 ± 0.06	44,605	0.63 ± 0.06	25,025	0.88 ± 0.03
QKBfly-pipeline	0.62 ± 0.06	44,605	0.58 ± 0.06	25,025	0.85 ± 0.03
QKBfly-noun	0.73 ± 0.06	33,400	0.68 ± 0.06	16,626	0.76 ± 0.02

the runtimes for DEFIE are not known; all DEFIE numbers in Table 3 are from [8].

Results on Entity Disambiguation. We compare QKBfly variants to DEFIE/Babelfy on the NED sub-task: linking entities to the KB (Yago and BabelNet, cross-linked via Wikipedia). We remark that Babelfy [36] also is a graph-based approach to NED. It performs word sense disambiguation based on a loose identification of candidate meanings. This is coupled with a densest subgraph heuristic which selects high-coherence semantic interpretations. Since Babelfy does not consider pronouns, we omit the pronoun resolution. As shown in Table 4, QKBfly gains 4% while QKBfly-pipeline loses 2% against Babelfy. We observe subtle errors of QKBfly-pipeline and Babelfy coming from the missing type signature feature (e.g., for `Liverpool` the city versus `Liverpool_F.C.` the soccer club).

Table 4: Experimental results on linking entities to Yago (at 95% confidence intervals).

Method	Precision	#Extractions
$\text{DEFIE}_{\text{Babelfy}}$	0.82 ± 0.05	39,684
QKBfly	0.86 ± 0.04	50,026
QKBfly-pipeline	0.80 ± 0.05	50,026

Results on Initial Extraction. We compare the Open IE component of QKBfly against state-of-the-art methods including the original work of ClausIE, Reverb [20], Ollie [35] and Open IE 4.2[7]. Table 5 shows experimental results on the Reverb dataset. ClausIE performs best in terms of precision and the number of extractions. However, it does not provide any canoncalized output and is much slower than the other methods including QKBfly, Ollie, and Open IE 4.2 that benefit from using the MaltParser instead of the Stanford Parser. The purely pattern-based Reverb, which does not use any dependency parsing, is the fastest one. QKBfly shows decent performance in all regards.

Table 5: Experiments on Open IE component. Average runtime in ms/sentence (at 95% confidence intervals).

Method	Precision	#Extract.	Avg. Runtime (ms)
ClausIE	0.62	1,707	374 ± 127
QKBfly	0.57	1,308	36 ± 11
Reverb	0.53	727	8 ± 2
Ollie	0.44	1,242	24 ± 9
Open IE 4.2	0.56	1,153	59 ± 14

[7] `https://github.com/knowitall/openie`

7.2 Experiment on Joint NED and CR

Benchmark. In addition to DEFIE-Wikipedia dataset, we run experiments on two new benchmarks:

- News dataset, consisting of 100 sport news articles with 3,751 sentences extracted from more than 20 news websites such as `bbc.com`, `telegraph.co.uk`, `nytimes.com`, and more on 3rd June 2017.
- Wikia dataset, consisting of 10 Wikia pages with 880 sentences about Game of Thrones, Season 1[8]. Each page consists of text describing an episode of the series.

Methods under Comparison. We compare two configurations of QKBfly with different graph algorithms.

- QKBfly, which performs NED and CR by the greedy approximation algorithm (described in Section 4).
- QKBfly-ilp, which performs NED and CR by an Integer Linear Programming (ILP) approach (described in Appendix A).

Results. As shown in Table 6, even though QKBfly-ilp gains 1%-2% in precision, which we also found to be significant under a t-test with a p-value of 0.01 on the DEFIE-Wikipedia dataset, it is much slower than QKBfly, especially when processing long documents in the Wikia dataset. This is because QKBfly-ilp has to handle a very large number of variables in the ILP translation of the graph problem, which makes it less suitable for on-the-fly KB construction. The QKBfly variants generally lose around 10% in precision when working on the Wikia dataset in comparison to the News and DEFIE-Wikipedia datasets, since the former contains many emerging ("out-of-Yago") entities such as movie characters. We observe that 71% of entities extracted from the Wikia dataset are out-of-Yago, while only 24% of entities extracted from the News dataset and 13% of entities extracted from the DEFIE-Wikipedia dataset are new.

7.3 Experiments on Information Extraction

To understand how well our method works on a more traditional IE task, we compare QKBfly also against DeepDive [57] by extracting instances of the `spouse` relation from the entire DEFIE-Wikipedia dataset. The DeepDive tutorials[9] specifically provide a pre-configured extraction model for this particular relation, which we additionally retrained by feeding all instances of married couples in DBpedia as positive examples into the DeepDive learner. As in traditional IE, the priority here is on precision, such that we use a high confidence threshold $\tau = 0.9$ for both systems.

[8] `http://gameofthrones.wikia.com/wiki/Season_1`
[9] `http://deepdive.stanford.edu/example-spouse`

Table 6: Experiments on graph algorithms (at 95% confidence intervals).

DEFIE-Wikipedia dataset			
Method	**Precision**	**#Extract.**	**Avg. Runtime (s)**
QKBfly	0.65 ± 0.06	69,630	0.88 ± 0.03
QKBfly-ilp	0.66 ± 0.06	69,630	46.59 ± 16.41
News dataset			
Method	**Precision**	**#Extract.**	**Avg. Runtime (s)**
QKBfly	0.65 ± 0.06	2,096	1.43 ± 0.07
QKBfly-ilp	0.67 ± 0.06	2,096	71.18 ± 25.76
Wikia dataset			
Method	**Precision**	**#Extract.**	**Avg. Runtime (s)**
QKBfly	0.54 ± 0.06	917	4.29 ± 0.11
QKBfly-ilp	0.55 ± 0.06	917	542.36 ± 61.72

Results. Figure 5 shows the precision-recall curves of DeepDive and QKBfly over the results, which we ranked by the confidence scores that each of the two systems assigns to its extracted facts. Table 7 also depicts the precision values at various recall levels (measured in terms of the number of extractions). Both systems perform very well in terms of precision at the lower recall levels, while QKBfly tends to outperform DeepDive at the higher recall levels. We observe that many extractions of QKBfly come from co-reference resolution for pronouns, which is not part of the extraction model of DeepDive. On the downside, QKBfly is substantially slower than DeepDive in this setting. However, one should notice that QKBfly always performs extractions for all relations – not just for the spouse relation. Considering that DeepDive needs a separate (manually crafted) extraction model for each individual target relation, we believe that this is an excellent result for QKBfly.

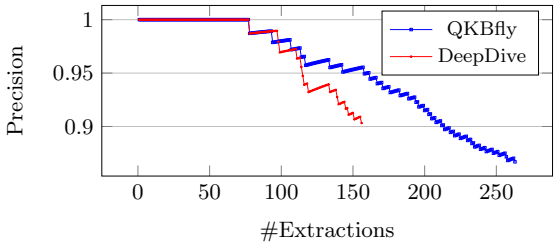

Figure 5: Precision-recall curves.

Table 7: Experiments on extracting instances of spouses on DEFIE-Wikipedia dataset.

Method	Precision	#Extractions	Runtime (minutes)
QKBfly	1.0	50	206
	0.95	150	
	0.87	250	
DeepDive	1.0	50	117
	0.91	150	
	—	250	

7.4 Experiments on On-the-fly KB for Ad-hoc Question Answering

Benchmark. As a final use-case, we run experiments on a newly designed QA benchmark which we coin "GoogleTrendsQuestions". First, we used the popular Google Trends service to identify 50 recent events of wider interest between January 2015 and October 2016. Second, we asked students to formulate meaningful questions about these events, and also provide the gold-standard answers. This resulted in 100 questions in total. An example question is: "*Which band was playing during the Paris attacks?*"

Evaluation Metric. We report the macro-averaged precision, recall and F1 score across all test questions. That is, given a set of questions $q_1 \dots q_n$, their gold answers $g_1 \dots g_n$ and the answer sets $a_1 \dots a_n$ returned by our system, where each answer set a_i can consist of a single value or a list of values, we compute the macro-average precision, recall and F1 score as

$$\text{avg. precision} = \frac{1}{n}\sum_{i=1}^{n} precision(g_i, a_i)$$

$$\text{avg. recall} = \frac{1}{n}\sum_{i=1}^{n} recall(g_i, a_i)$$

$$\text{avg. F1} = \frac{1}{n}\sum_{i=1}^{n} F1(g_i, a_i)$$

where each of $precision(g_i, a_i)$, $recall(g_i, a_i)$ and $F1(g_i, a_i)$ is computed in the regular way.

Methods under Comparison. We compare several configurations of QKBfly:

- QKBfly answers questions from an on-the-fly KB dynamically constructed by QKBfly. The KB contains triples as well as higher-arity facts.
- QKBfly-triples is an variant where the on-the-fly KB is limited to *subject-predicate-object* (SPO) triples.
- Sentence-Answers is a text-centric baseline where QKBfly is used to retrieve relevant sentences, but does not perform any fact extraction. Entities in these sentences then become answer candidates.
- QA-Freebase is baseline with a static KB where we apply the same QA method on the huge fact collection of Freebase.

A detailed description of how we implement QA in QKBfly is described in Appendix B. The Sentence-Answers baseline differs from the others in the step of collecting answer candidates (Step 3 in the Appendix). Here, all entities that co-occur with one of the question entities in the same sentence are considered as candidate answers. Additionally, the candidate features are the tokens in the sentences (rather than facts) where the candidate occurs. This is in the spirit of a traditional passage-retrieval-based QA method. It uses the same on-the-fly corpus as QKBfly, but does not perform explicit knowledge extraction.

Results. Table 9 shows the results on the GoogleTrendsQuestions benchmark. Here, QKBfly achieves an F1 score of 34.1%, and QKBfly-triples reaches 30.7%. QA-Freebase and Sentence-Answers perform far inferior. Particularly, QA-Freebase returns empty results in most cases due to the

Table 8: Sample questions and facts extracted by QKBfly. Final answers are marked with an asterisk.

Question	Fact
Who plays Han Solo in 'Star Wars: The Force Awakens'?	⟨Harrison_Ford*, act_in, Han_Solo, Star_Wars:_The_Force_Awakens⟩ ⟨Harrison_Ford*, return_in_as, *"Star Wars film series"*, Han_Solo⟩
Which band was playing during the Paris attacks?	⟨Eagles_of_Death_Metal*, play_in, *"a concert"*, Paris⟩ ⟨Eagles_of_Death_Metal*, injured_in_in, *"the attack"*, Paris⟩
Where was Pope Francis born?	⟨Pope_Francis, born_in_on, Buenos_Aires*, *"17 December 1936"*⟩ ⟨Pope_Francis, birth_place, Buenos_Aires*⟩
Who shot Keith Lamont Scott?	⟨Keith_Lamont_Scott, shot_by, *"a black officer"**⟩ ⟨Brently_Vinson*, shoot, *"Mr Scott"*⟩

lack of facts about recent events. QKBfly performs better than QKBfly-triples in questions which require ternary facts, such as "*Who plays Han Solo in 'Star Wars: The Force Awakens'?*" (see Table 8). To separate the effects of the two kinds of input documents we considered – Wikipedia and Google News –, we also ran QKBfly by using only Wikipedia articles and only the top-10 news articles, respectively. When using only Wikipedia articles to construct the on-the-fly KB, QKBfly achieves 32.4% in the F1 measure. Restricting it to using only Google news, on the other hand, leads to an F1 score of 33.2%.

Table 9: Results on GoogleTrendsQuestions.

Method	Precision	Recall	F1
QKBfly	0.330	0.383	0.341
QKBfly-triples	0.294	0.363	0.307
Sentence-Answers	0.173	0.199	0.179
QA-Freebase	0.095	0.100	0.096

As for an end-to-end experiment, we compare QKBfly against the state-of-the-art KB-QA system, namely AQQU [5]. AQQU achieves an F1 score of 10%. To be fair, we emphasize that this system has not been designed for a huge but static KB, namely Freebase, and thus cannot utilize any on-the-fly knowledge. Table 10 shows anecdotal samples for illustration.

Table 10: Samples of GoogleTrendsQuestions.

Question & Answers
Where was Pope Francis born?
Gold Answers: [Buenos_Aires]
AQQU: [Buenos_Aires]
QKBfly: [Buenos_Aires]
Who shot Keith Lamont Scott?
Gold Answer: [Brentley_Vinson]
AQQU: []
QKBfly: [*"a black officer"*, Brentley_Vinson]

8. RELATED WORK

Knowledge Base Population. KBs like Yago [51], DBpedia [2], Freebase [7], NELL [9], BabelNet [39] and Wikidata [54] extract SPO facts from Wikipedia and other sources. However, all of these are limited to a prespecified set of predicates: a few hundred in some, up to a few thousand in Freebase and Wikidata. Moreover, there is very limited support for higher-arity facts: Yago has temporal scopes for triples, Freebase contains compound objects for special kinds of higher-arity predicates (e.g., awards and marriages). IE methods for ternary facts have been developed by [30], based on distant supervision from large KBs. However, also these methods require prespecified relations and do not scale well. [29] proposes Sar-graphs, a semantic representation of higher-arity facts and their grounding in language. This approach relies on manual curation and is fairly limited in scope and scale. The idea of on-the-fly KB construction was also brought up by [37] as a challenge, but not pursued any further. Declarative methods to IE and KBP, including DeepDive [42, 48, 57] and SystemT [11, 12, 45], work on carefully-predefined sets of predicates and rules. Thus, they cannot directly be applied in a spontaneous "on-the-fly" manner.

Open IE. Open IE methods, like [1, 4, 8, 13, 19] overcome some of the limitations of KB population, but face issues regarding the quality and informativeness of their extractions, where neither entities nor predicates are canonicalized. Consequently, applications using Open IE output still need to deal with NED and the resolution of relational paraphrases. Some recent works have aimed to advance Open IE towards structured KBs. [1] map facts from Open IE techniques to a small number of pre-specified predicates. DEFIE [8] is a large-scale IE systems that feeds Open IE output into the NED tool Babelfy [36]. However, relational predicates are still not canonicalized. DEFIE is the main baseline against which we evaluate the coverage and quality of QKBfly. Related to Open IE are the TREC 2012–2014 Knowledge Base Acceleration (KBA) and 2015–2016 Dynamic Domain (DD) tracks. Both tracks aim to extract on-the-fly knowledge (coined "cold start") from text sources. However, virtually all contestants there also resort to extracting short text snippets or sentences rather than building a structured and canonicalized KB.

Finally, joint models for related NLP tasks include work on combinations of NER, NED, co-reference resolution (CR) and relation extraction. NER and CR have recently been jointly addressed by [25, 15]; this has been further extended by [16] to also include NED. [49] jointly tackles NER, CR relation extraction, but does not canonicalize entities and predicates. [32] jointly infers NER types and relational facts, again without canonicalization.

9. CONCLUSIONS

We presented QKBfly, a novel approach to build on-the-fly knowledge bases in a query-driven manner. We acquire facts for a much larger set of predicates than those in mainstream KBs. In contrast to the output of Open IE, ar-

guments of facts are canonicalized, so that they refer to unique entities with semantically typed predicates derived from clusters of phrases. Moreover, QKBfly is not limited to binary facts, but comprises also higher-arity ones by analyzing the clause structure of the input sentences. Use cases for QKBfly include ad-hoc question answering, summarization and other kinds of machine-reading applications. The QKBfly software is available[10] under the Apache License 2.0.

As for future work, on-the-fly KB construction faces a fundamental trade-off between extraction speed and output quality. While this paper has aimed to reconcile these goals, further improvements are needed. Additionally, on-the-fly relational paraphrase mining would be another important research direction.

10. ACKNOWLEDGMENTS

We would like to thank Christopher Ré for helpful discussions about using DeepDive, Claudio Delli Bovi for sharing the DEFIE-Wikipedia dataset, and Niket Tandon and Sreyasi Nag Chowdhury for their help with the ILP setup. We also thank the anonymous reviewers for their thoughtful and helpful comments.

APPENDIX

A. ILP SETUP

We next describe how we translate the densest-subgraph problem $\mathcal{S}^*$ into an ILP. We introduce a binary variable cnd_{i_j} for each `noun-phrase` or `pronoun` node n_i and each `entity` candidate $e_{i_j} \in ent(n_i, \mathcal{G})$. The variable $cnd_{i_j} = 1$ iff e_{i_j} is chosen for n_i in the densest subgraph $\mathcal{S}^*$. As constraints, we have (1) $\sum_j cnd_{i_j} = 1 \quad \forall i$, and (2) two `noun-phrase` or `pronoun` nodes n_i, n_t are linked by a `sameAs` edge in $\mathcal{S}^*$ iff $cnd_{i_j} = cnd_{t_j} \quad \forall j$. We additionally introduce a binary variable $joint\text{-}rel_{i_j t_k}$ for each pair of `noun-phrase` or `pronoun` nodes n_i, n_t, which are connected by a `relation` edge $r_{i,t}$, and similarly for each pair of `entity` nodes $e_{i_j} \in ent(n_i, \mathcal{G})$, $e_{t_k} \in ent(n_t, \mathcal{G})$. Thus, the variable $joint\text{-}rel_{i_j t_k} = 1$ iff e_{i_j} is chosen for n_i and e_{t_k} is chosen for n_t in $\mathcal{S}^*$. Consequently, $joint\text{-}rel_{i_j t_k} = 1$ iff $cnd_{i_j} = 1$ and $cnd_{t_k} = 1$. We then aim to maximize

$$\sum_{\substack{n_i \in \mathcal{G} \\ e_{i_j} \in ent(n_i, \mathcal{G})}} cnd_{i_j} \cdot w(n_i, e_{i_j}) + \sum_{n_i, n_t \in \mathcal{G}} joint\text{-}rel_{i_j t_k} \cdot w(n_i, n_t, \mathcal{S}^{i_j t_k})$$

where:

- $w(n_i, e_{i_j})$ is the `means` edge weight between n_i and e_{i_j},
- $w(n_i, n_t, \mathcal{S}^{i_j t_k})$ is the `relation` edge weight between n_i and n_t in a subgraph $\mathcal{S}^{i_j t_k}$ constructed from $\mathcal{G}$ by only considering e_{i_j} and e_{t_k} as the candidates for n_i and n_t, respectively.

In our experiments, we used the Gurobi[11] solver to find the solution to the above programs.

[10] `https://people.mpi-inf.mpg.de/~datnb/`
[11] `http://www.gurobi.com`

B. QA SETUP

Question answering over structured knowledge bases (KB-QA) [5, 6, 55] denotes the task of translating a natural language question into a structured query (e.g., using SPARQL for querying SPO triples), which is then executed over the underlying KB (e.g., Freebase [7]) to obtain answer entities. As an extrinsic use-case, we harness QKBfly for KB-QA. In contrast to mainstream works, we pursue the case where no fact repository is available upfront and all relevant facts need to be gathered on-the-fly, triggered by a natural-language question. Specifically, when given a question like "*who did vladimir lenin marry?*", QKBfly computes the answers in the following four steps.

Step 1. Entities in the question are detected and used to retrieve relevant documents in Wikipedia and Google News. For example, we use the Wikipedia article that has the id of `Vladimir_Lenin`, and we issue a Google News query with the full text of the input question. For each question, we retrieve the top-10 results from Google News.

Step 2. QKBfly processes the retrieved documents to extract facts. No pre-existing fact repository is used.

Step 3. As answer candidates, our method fetches all entities (or string literals like dates) from its question-specific ad-hoc KB. A type filter is applied to ensure that candidates satisfy the expected answer type(s). For example, a question starting with *"Who"* can be answered only by entities of types `PERSON`, `CHARACTER` or `ORGANIZATION`. Here, we use our type system based on infoboxes for entities, combined with Stanford NER and SUTime tags. Note that this step is focused on recall, ensuring that we do not miss good answers. The following step ensures high precision by further filtering the candidates and ranking them.

Step 4. We run each answer candidate through a pre-trained binary classifier, using an SVM model. The model is trained on the WebQuestions training questions and their gold-standard answers [6]. The positively labeled candidates are output as final answers. For single-answer factoid questions (if detectable), only the top-ranked answer is output.

Classifier Features. For each question, we extract all tokens: word-level lemmatized unigrams and entities. For example, the question "*who did vladimir lenin marry?*" contains "*who*", "*do*", "*marry*" and the entity `Vladimir_Lenin`. For each candidate answer, we similarly extract all tokens co-occurring in the same KB facts, again all word-level lemmatized unigrams and all entities. The feature set for a pair of a question and its candidate answer then are all token pairs (x, y) where x is a token occurring with the question and y is a token occurring with the candidate. For data sparseness and simplicity, we treat these as binary features.

Classifier Training. WebQuestions consists of 3,778 training questions, each of which is paired with its answer set. These were collected using the Google Suggest API and further crowdsourcing. Answers (i.e., Freebase entities) are finally converted to Wikipedia pages by the Freebase API. We use the gold answers of the WebQuestions training corpus and Wikipedia-based question-specific KBs produced by QKBfly for training the SVM classifier. Facts extracted by QKBfly that contain correct or incorrect answers are used as positive or negative training samples, respectively. The model is constructed using the Liblinear SVM library [21] by using the default settings.

C. REFERENCES

[1] G. Angeli, M. J. J. Premkumar, and C. D. Manning. Leveraging Linguistic Structure For Open Domain Information Extraction. In *ACL*, pages 344–354, 2015.

[2] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, and Z. Ives. DBpedia: A Nucleus for a Web of Open Data. In *ISWC*, pages 11–15, 2007.

[3] D. Bamman, T. Underwood, and N. Smith. A Bayesian Mixed Effects Model of Literary Character. In *ACL*, pages 370–379, 2014.

[4] M. Banko, M. J. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni. Open Information Extraction from the Web. In *IJCAI*, pages 2670–2676, 2007.

[5] H. Bast and E. Haussmann. More Accurate Question Answering on Freebase. In *CIKM*, pages 1431–1440, 2015.

[6] J. Berant, A. Chou, R. Frostig, and P. Liang. Semantic Parsing on Freebase from Question-Answer Pairs. In *EMNLP*, pages 1533–1544, 2013.

[7] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: A Collaboratively Created Graph Database for Structuring Human Knowledge. In *SIGMOD*, pages 1247–1250, 2008.

[8] C. D. Bovi, L. Telesca, and R. Navigli. Large-Scale Information Extraction from Textual Definitions through Deep Syntactic and Semantic Analysis. *TACL*, 3:529–543, 2015.

[9] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. H. Jr., and T. M. Mitchell. Toward an Architecture for Never-Ending Language Learning. In *AAAI*, pages 1306–1313, 2010.

[10] A. X. Chang and C. Manning. SUTime: A Library for Recognizing and Normalizing Time Expressions. In *LREC*, pages 3735–3740, 2012.

[11] L. Chiticariu, R. Krishnamurthy, Y. Li, S. Raghavan, F. R. Reiss, and S. Vaithyanathan. SystemT: An Algebraic Approach to Declarative Information Extraction. In *ACL*, pages 128–137, 2010.

[12] L. Chiticariu, Y. Li, and F. R. Reiss. Rule-Based Information Extraction is Dead! Long Live Rule-Based Information Extraction Systems! In *EMNLP*, pages 827–832, 2013.

[13] L. Del Corro and R. Gemulla. ClausIE: Clause-based Open Information Extraction. In *WWW*, pages 355–366, 2013.

[14] P. Domingos and D. Lowd. *Markov Logic: An Interface Layer for Artificial Intelligence*. Morgan and Claypool Publishers, 2009.

[15] G. Durrett and D. Klein. Easy Victories and Uphill Battles in Coreference Resolution. In *EMNLP*, pages 1971–1982, 2013.

[16] G. Durrett and D. Klein. A Joint Model for Entity Analysis: Coreference, Typing, and Linking. *TACL*, 2:477–490, 2014.

[17] M. Dylla, I. Miliaraki, and M. Theobald. Top-k Query Processing in Probabilistic Databases with Non-Materialized Views. In *ICDE*, pages 122–133, 2013.

[18] M. Dylla, M. Theobald, and I. Miliaraki. Querying and Learning in Probabilistic Databases. In *Reasoning Web*, pages 313–368, 2014.

[19] O. Etzioni, A. Fader, J. Christensen, S. Soderland, and M. Mausam. Open Information Extraction: The Second Generation. In *IJCAI*, pages 3–10, 2011.

[20] A. Fader, S. Soderland, and O. Etzioni. Identifying Relations for Open Information Extraction. In *EMNLP*, pages 1535–1545, 2011.

[21] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A Library for Large Linear Classification. *JMLR*, 9:1871–1874, 2008.

[22] L. Galárraga, G. Heitz, K. Murphy, and F. M. Suchanek. Canonicalizing Open Knowledge Bases. In *CIKM*, pages 1679–1688, 2014.

[23] W. Gatterbauer and D. Suciu. Dissociation and Propagation for Approximate Lifted Inference with Standard Relational Database Management Systems. *VLDB Journal*, 26(1):5–30, 2017.

[24] E. Gribkoff and D. Suciu. SlimShot: In-database Probabilistic Inference for Knowledge Bases. *PVLDB*, 9(7):552–563, 2016.

[25] A. Haghighi and D. Klein. Coreference Resolution in a Modular, Entity-centered Model. In *HLT*, pages 385–393, 2010.

[26] J. Hoffart, S. Seufert, D. B. Nguyen, M. Theobald, and G. Weikum. KORE: Keyphrase Overlap Relatedness for Entity Disambiguation. In *CIKM*, pages 545–554, 2012.

[27] J. Hoffart, M. A. Yosef, I. Bordino, H. Fürstenau, M. Pinkal, M. Spaniol, B. Taneva, S. Thater, and G. Weikum. Robust Disambiguation of Named Entities in Text. In *EMNLP*, pages 782–792, 2011.

[28] D. Klein and C. D. Manning. Accurate Unlexicalized Parsing. In *ACL*, pages 423–430, 2003.

[29] S. Krause, L. Hennig, A. Moro, D. Weissenborn, F. Xu, H. Uszkoreit, and R. Navigli. Sar-graphs: A Language Resource Connecting Linguistic Knowledge with Semantic Relations from Knowledge Graphs. *JWS*, 37:112–131, 2016.

[30] S. Krause, H. Li, H. Uszkoreit, and F. Xu. Large-Scale Learning of Relation-Extraction Rules with Distant Supervision from the Web. In *ISWC*, pages 263–278, 2012.

[31] K. Li, X. Zhou, D. Z. Wang, C. Grant, A. Dobra, and C. Dudley. In-database Batch and Query-time Inference over Probabilistic Graphical Models Using UDA-GIST. *VLDB Journal*, 26(2):177–201, 2017.

[32] Q. Li and H. Ji. Incremental Joint Extraction of Entity Mentions and Relations. In *ACL*, pages 402–412, 2014.

[33] D. C. Liu and J. Nocedal. On the Limited Memory BFGS Method for Large Scale Optimization. *Mathematical Programming*, 45(3):503–528, 1989.

[34] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky. The Stanford CoreNLP Natural Language Processing Toolkit. In *ACL*, pages 55–60, 2014.

[35] Mausam, M. Schmitz, R. Bart, S. Soderland, and O. Etzioni. Open Language Learning for Information Extraction. In *EMNLP-CoNLL*, pages 523–534, 2012.

[36] A. Moro, A. Raganato, and R. Navigli. Entity Linking meets Word Sense Disambiguation: a Unified Approach. *TACL*, 2:231–244, 2014.

[37] N. Nakashole and G. Weikum. Real-time Population of Knowledge Bases: Opportunities and Challenges. In *AKBC Workshop*, 2012.

[38] N. Nakashole, G. Weikum, and F. Suchanek. PATTY: A Taxonomy of Relational Patterns with Semantic Types. In *EMNLP-CoNLL*, pages 1135–1145, 2012.

[39] R. Navigli and S. P. Ponzetto. BabelNet: The Automatic Construction, Evaluation and Application of a Wide-coverage Multilingual Semantic Network. *AIJ*, 193:217–250, 2012.

[40] D. B. Nguyen, J. Hoffart, M. Theobald, and G. Weikum. AIDA-light: High-Throughput Named-Entity Disambiguation. In *LDOW*, 2014.

[41] D. B. Nguyen, M. Theobald, and G. Weikum. J-NERD: Joint Named Entity Recognition and Disambiguation with Rich Linguistic Features. *TACL*, 4:215–229, 2016.

[42] F. Niu, C. Zhang, C. Re, and J. W. Shavlik. DeepDive: Web-scale Knowledge-base Construction using Statistical Learning and Inference. In *VLDS*, pages 25–28, 2012.

[43] J. Nivre and J. Hall. Maltparser: A Language-Independent System for Data-Driven Dependency Parsing. In *TLT*, pages 13–95, 2005.

[44] R. Quirk, S. Greenbaum, G. Leech, and J. Svartvik. *A Comprehensive Grammar of the English Language*. Longman, 1985.

[45] F. Reiss, S. Raghavan, R. Krishnamurthy, H. Zhu, and S. Vaithyanathan. An Algebraic Approach to Rule-Based Information Extraction. In *ICDE*, pages 933–942, 2008.

[46] S. Riedel. Improving the Accuracy and Efficiency of MAP Inference for Markov Logic. In *UAI*, pages 468–475, 2008.

[47] S. Riedel, L. Yao, A. McCallum, and B. M. Marlin. Relation Extraction with Matrix Factorization and Universal Schemas. In *HLT-NAACL*, pages 74–84, 2013.

[48] J. Shin, S. Wu, F. Wang, C. De Sa, C. Zhang, and C. Ré. Incremental Knowledge Base Construction Using DeepDive. *PVLDB*, 8(11):1310–1321, 2015.

[49] S. Singh, S. Riedel, B. Martin, J. Zheng, and A. McCallum. Joint Inference of Entities, Relations, and Coreference. In *AKBC*, pages 1–6, 2013.

[50] M. Sozio and A. Gionis. The Community-search Problem and How to Plan a Successful Cocktail Party. In *KDD*, pages 939–948, 2010.

[51] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: A Core of Semantic Knowledge. In *WWW*, pages 697–706, 2007.

[52] F. M. Suchanek, M. Sozio, and G. Weikum. SOFIE: A Self-organizing Framework for Information Extraction. In *WWW*, pages 631–640, 2009.

[53] D. Suciu, D. Olteanu, R. Christopher, and C. Koch. *Probabilistic Databases*. Morgan & Claypool Publishers, 2011.

[54] D. Vrandečić and M. Krötzsch. Wikidata: A Free Collaborative Knowledgebase. *CACM*, 57(10):78–85, 2014.

[55] K. Xu, S. Reddy, Y. Feng, S. Huang, and D. Zhao. Question Answering on Freebase via Relation Extraction and Textual Evidence. In *ACL*, 2016.

[56] M. A. Yosef, J. Hoffart, I. Bordino, M. Spaniol, and G. Weikum. AIDA: An Online Tool for Accurate Disambiguation of Named Entities in Text and Tables. *PVLDB*, 4(12):1450–1453, 2011.

[57] C. Zhang, J. Shin, C. Ré, M. Cafarella, and F. Niu. Extracting Databases from Dark Data with DeepDive. In *SIGMOD*, pages 847–859, 2016.