Holistic Query Evaluation over Information Extraction Pipelines

Ekaterini Ioannou*
Open University of Cyprus
ekaterini.ioannou@ouc.ac.cy

Minos Garofalakis
ATHENA Research and Innovation Centre &
Technical University of Crete
minos@softnet.tuc.gr

ABSTRACT

We introduce holistic in-database query processing over information extraction pipelines. This requires considering the joint conditional distribution over generic Conditional Random Fields that uses factor graphs to encode extraction tasks. Our approach introduces *Canopy Factor Graphs*, a novel probabilistic model for effectively capturing the joint conditional distribution given a canopy clustering of the data, and special query operators for retrieving resolution information. Since inference on such models is intractable, we introduce an approximate technique for query processing and optimizations that cut across the integrated tasks for reducing the required processing time. Effectiveness and scalability are verified through an extensive experimental evaluation using real and synthetic data.

PVLDB Reference Format:
Ekaterini Ioannou, and Minos Garofalakis. Holistic Query Evaluation over Information Extraction Pipelines. *PVLDB*, 11(2): 217-229, 2017.
DOI: 10.14778/3149193.3149201

1. INTRODUCTION

Entity Resolution (ER) is the vital task of creating **entities**, one for each set of **instances** that describe a distinct real world **object** (e.g., location, event). To successfully perform resolution in highly heterogeneous and unstructured data, the Information Extraction (IE) community uses *information extraction pipelines*. Important tasks in such a pipeline are: *segmentation* that divides text into a sequence of meaningful fields (e.g., to become tuples in a relational table); *coreference* that detects instances of the same objects; and *canonicalization* that creates entities representing the detected coreferent instances. IE research has shown that resolution by maintaining and combining hypotheses across these tasks achieves more accurate results than the sequential, in-isolation execution of tasks [14, 24].

The vast majority of existing resolution approaches from the database (DB) community focus only on the coreference task. This task is typically executed in isolation and the results are simply used normally during query processing; that is, once the coreference process is executed, a predefined threshold is employed to decide which instances are merged together. Each merge leads to an entity, which is typically mapped to one of the instances or to the union of all coreferent instances. The created entities permanently replace the original instances and are then used for answering queries. The focus of such earlier approaches is mainly on suggesting mechanisms to detect the coreferent instances, e.g., based on the use of instance similarities [5] or exploiting the relationships across instances (namely collective resolution) [8, 28]. Although some approaches enhance this basic methodology, e.g., with propagation of merged information to other possible matches [8], there is still a lack of methodologies providing a deep integration with more than one extraction task and the ability to integrate the complex probabilistic graphical models required to represent these tasks.

In this paper, we adopt state-of-the-art IE techniques and address the issues of probabilistic DB support for IE pipelines comprising the tasks of coreference and canonicalization. (We do not consider segmentation since it works on raw text and is, thus, typically executed outside relational DB systems.) We introduce a novel approach based on the joint probabilistic inference over generic Conditional Random Fields (CRFs) that encode the conditional distribution of the factor graphs representing these extraction tasks. We argue that, compared to existing probabilistic DB approaches addressing these tasks separately, such holistic processing using a joint graphical model results in higher accuracy answers by considering the complete resolution scenario rather than individual tasks. A key insight here is that canonical entities can affect the "possible worlds" distribution of ER — for instance, after canonicalization it may become obvious that two sets of coreferent instances, which we initially thought to be distinct, (i) are actually the same entity, or (ii) are indeed distinct as also reflected by the stronger evidence from the resulting canonical entities.

Holistic query processing over IE pipelines offers several benefits. First, it allows incorporating optimizations that cut across the integrated tasks. Furthermore, it allows query processing over the full "possible worlds" distribution, rather than focusing solely on individual high probability resolution decisions. The importance of tight integration of inference and query processing for both efficiency and accuracy has been convincingly demonstrated in earlier work, e.g., [27].

*Part of this work was done while the author was with the Technical University of Crete.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Articles from this volume were invited to present their results at The 44th International Conference on Very Large Data Bases, August 2018, Rio de Janeiro, Brazil.
Proceedings of the VLDB Endowment, Vol. 11, No. 2
Copyright 2017 VLDB Endowment 2150-8097/17/10... $ 10.00.
DOI: 10.14778/3149193.3149201

Buyer^D

id	*name*	*surname*	*gender*	*country*
r_1	Alexander	Hill	male	U.S.
r_2	Alexander	Hill, Mr.	M	USA
r_3	Alex	Hill		United St. of America
r_4	Alexandra	Hill, Dr.	Female	USA
r_5	A.	Hill	female	United States
r_6	Alexander	H., Mr.	male	United States
r_7	Alexander	Hiller	M	States
r_8	Alexander	Hiller	male	United States

History

order	*buyer*	*cost*	*type*	*returns*	*bonus*
o_1	r_1	150	electronics	y	10
o_2	r_2	50	books	y	10
o_3	r_3	20	electronics	y	5
o_4	r_4	80	toys	n	15
o_5	r_4	90	cloths	y	15
o_6	r_5	100	garden	n	20
o_7	r_5	110	electronics	y	20

Figure 1: A fraction of a database that contains tables with duplicates (i.e., duplicate instances in table Buyer^D).

Possible world 1 - Coreferent set {{r_1,r_2,r_3},{r_4,r_5},{r_6,r_7,r_8}} p_1

Id.	Coref.	Canonical Entity	History
e_{123}	r_1,r_2,r_3	⟨Alexander; Hill, Mr.; male; United St. of America⟩	o_1-o_3
e_{45}	r_4,r_5	⟨Alexandra; Hill, Dr.; female; United States⟩	o_4-o_7
e_{678}	r_6,r_7,r_8	⟨Alexander; Hiller; male; United States⟩	-

Possible world 2 - Coreferent set {{r_1,r_2},{r_3,r_4,r_5},{r_6,r_7,r_8}} p_2

Id.	Coref.	Canonical Entity	History
e_{12}	r_1,r_2	⟨Alexander; Hill, Mr.; male; USA⟩	o_1-o_2
e_{345}	r_3,r_4,r_5	⟨Alexandra; Hill, Dr.; female; United St. of America⟩	o_3-o_7
e_{678}	r_6,r_7,r_8	⟨Alexander; Hiller; male; United States⟩	-

Possible world 3 - Coreferent set {{r_1,r_2},{r_3},{r_4,r_5},{r_6,r_7,r_8}} p_3

Id.	Coref.	Canonical Entity	History
e_{12}	r_1,r_2	⟨Alexander; Hill, Mr.; male; USA⟩	o_1-o_2
e_3	r_3	⟨Alex; Hill; ; United St. of America⟩	o_3
e_{45}	r_4,r_5	⟨Alexandra; Hill, Dr.; female; United States⟩	o_4-o_7
e_{678}	r_6,r_7,r_8	⟨Alexander; Hiller; male; United States⟩	-

Figure 2: The join between tables History and Buyer^D. Each answer in the possible worlds of table Buyer^D contains the canonical entity, the coreferent set, and the probability.

Another important aspect of our work is that it investigates the integration of complex IE pipelines within relational query processing. Recently, the DB community has shown increasing interest in a deeper integration of IE tasks and, more specifically, in incorporating a single extraction task within database query processing. This is discussed, for instance, in [27] and [4]: both perform segmentation (encoded using linear-chain CRFs) through in-database inference during query processing. Results show improvements in accuracy and efficiency, e.g., [27]. In this aspect, our approach complements existing approaches, since it also deals with the additional challenges that arise from handling more than one extraction task and more complex graphical models (i.e., generic rather than linear-chain CRFs).

A main technical challenge is that exact inference on generic CRFs is intractable. We alleviate this through three novel mechanisms. The first is moving away from instances and towards "fuzzy" instance clusters, i.e., namely *canopies*. More specifically, we introduce a new probabilistic model structure that is based on canopies (i.e., *Canopy Factor Graphs (CFG)*) and approximately model the joint conditional probability. The second mechanism is a novel process that exploits our CFG model to approximately estimate and generate the best possible coreferent sets. Combining this with queries minimizes the overall execution time. Finally, we also introduce algebraic optimizations that further reduce the time required for in-database query processing.

Holistic Query Evaluation: An Example. Fig. 1 shows a small fragment of a store's database that monitors and continuously integrates data from other systems. Table Buyer contains instances of the same real world objects, and table History contains information related to them. Query answers must reflect the canonical entities of Buyer, e.g., entity e_{45} resulting from merging coreferent instances r_4 and r_5. The joins are also performed given the canonical entities, for example entity e_{45} will be joined with o_4-o_7. Users are able to request entities given the data found either in the corresponding canonical entity (i.e., e_{45}) or in the detected coreferent instances (i.e., r_4 and r_5). The latter is important since it ensures that users will retrieve answers even if they do not know the exact data in the final canonical entities.

Consider searching the "electronics" department returns for person (i.e., canonical entity) having surname "Hill%" and country "United St. of America", with the query:

```
1  SELECT name, surname, gender, order, cost
2  FROM   History join Buyer^D
3  WHERE type="electronics" and returns="y" and          (Q.1)
4        canonical-entity.surname like "Hill%" and
5        canonical-entity.country="United St. of America"
6  ORDER BY cost desc
7  LIMIT with-k 2
```

In order to use table Buyer, we must first derive its alternative resolution solutions, following the "possible worlds" semantics. Fig. 2 shows the three most probable worlds (i.e., solutions). The important point to note here is that solutions and their probabilities are affected by the coreferent instances as well as the resulting canonical entities.

Since we are interested in the returns, which are stored in table History, we need to perform the join between tables History and Buyer. The join is based on the resolution solutions, and more specifically the coreferent instances. For example, the first entity in possible world 1 has instances r_1, r_2, and r_3 as coreferent. Thus, it must be joined with the History records involving orders o_1 to o_3.

The huge volume of entities that will be created (as is typically the situation with possible worlds) will overwhelm users and not allow them to spot useful information. To remedy this, we introduce operators expressing resolution needs. First, conditions can be expressed using the coreferent instances or/and the canonical entities. E.g., the user in Q.1 is interested in resolution results that contain a canonical entity with the *surname* containing *"Hill"* (line 4). Second, we allow users to specify the number of possible worlds to be considered, by specifying either a minimum probability or an exact number of the result rows (as in Q.1, line 7).

After applying the given conditions on the two tables, we need to generate the possible resolution solutions for table Buyer, and then construct the records resulting from their join with table History. For instance, given the conditions of Q.1 for Buyer (i.e., lines 4-5) we have $e_{1,2,3}$, $e_{3,4,5}$, and e_3, and, given the conditions for table History (i.e., line 3), we have o_1, o_3, and o_7. Thus, we perform the join between: (1) $e_{1,2,3}$ and o_1; (2) $e_{1,2,3}$ and o_3; (3) $e_{3,4,5}$ and o_3; (4) $e_{3,4,5}$ and o_7; and (5) e_3 and o_3. The final step is to execute the ORDER BY and LIMIT operators (lines 6-7). Hence, only answers 1 and 4 remain, and the final result set for Q.1 becomes:

{ ⟨Alexander; Hill, Mr.; male; United St. of America, o_1, 150⟩,
⟨Alexandra; Hill, Dr.; female; United St. of America, o_7, 110⟩ }

Note that the conventional approach of addressing the problem would execute the coreference process, select the highest probability coreferent sets, and replace the instances

in each coreferent set with one of the instances (e.g., the most recently added instance). In the data of Fig. 1 this process might result in the coreferent sets $\{r_1, r_2\}$ and $\{r_3, r_4, r_5\}$, with r_1 replacing the instances of the first set and r_3 replacing the instances of the second. This means that query answers can include only r_1 and r_3 as the entities of Buyer^D.

Overview. We support new resolution querying semantics (Sec. 2). The proposed approach is based on a novel combination of overlapping canopies with the joint conditional distribution using CRFs (Sec. 3). Query processing is performed using an approximate technique (Sec. 4) with query plans being optimized to reduce execution time (Sec. 5).

Our Contributions. We advocate a novel approach for holistic query processing over complex IE pipelines, focusing on two important tasks: coreference and canonicalization. Our contributions can be summarized as follows:

1. We provide new semantics for probabilistic querying of duplicated instances using information extraction pipelines.
2. We introduce *Canopy Factor Graphs (CFGs)* — a generic, approximate CRF encoding the joint conditional distribution of the factor graphs representing the IE tasks, given a canopy clustering of the instances.
3. We introduce approximate query processing that considers the joint conditional distribution of the defined CFGs.
4. To improve query performance, we propose optimizations (based on equivalent query rewritings) that cut across the integrated tasks and significantly reduce processing time.
5. We verify effectiveness and efficiency through an extensive evaluation based on real and synthetic data.

2. DATA MODEL AND QUERIES

To represent the resolution-related information we introduce a data and query model that has the ability to tightly integrate probabilistic IE semantics. More specifically, we consider a typical relational database composed of relations, with each relation containing a large number of instances. Some of the database relations have instances that describe the same real world objects.

Definition 1. *A **relation** is a set of instances. It is denoted as $R(a_1,...,a_k)=\{r_1,...,r_n\}$, with R being the name of the relation, a_1, ..., a_k its attributes, and r_1, ..., r_n **instances** of R. Each r_i provides values v_1, ..., v_k for each of the attributes a_1, ...a_k of the relation.* ∎

A database contains various relations, with and without duplicates. We use R^D to denote a relation R that contains duplicates, i.e., contains multiple instances for the same real world object. In the remaining text, we use notation $r_i \cong r_j$ to denote that r_i and r_j are duplicated instances.

Definition 2. *A **coreferent set** M is a subset of the R^D instances, i.e., $M \subseteq R^D$, which describe the same real world object: $\forall\ r_i, r_j \in M \Rightarrow r_i \cong r_j$.* ∎

In essence, providing a solution to the entity resolution problem for relation R^D requires *partitioning* the instances into disjoint coreferent sets, i.e., $CM_j=\{M_1, M_2, ..., M_k\}$ where $\cup_{i=1}^{k} M_i = R^D$. Each coreferent set M_i means that the instances it contains describe a particular real world object and should be considered as such.

We consider entities as a complementary design artifact of our data model. An entity e is modeled exactly as an instance, but describes a particular and distinct real world object. Since each coreferent set contains instances that describe the same object, we can create a single entity e to represent them. This entity acts as the canonical representation of all these instances.

```
1  SELECT R_i.attr_1, R_i.attr_2, ..., R_j^D.attr_1, R_j^D.attr_2, ..., [prob]
2  FROM   R_i  join  R_j^D
3  WHERE R_i-conditions
4        [ and|or corefer-instance conditions]   // loose mode
5        [ and|or canonical-entity conditions]   // tight mode
6  [ ORDER BY  R_i.attr asc/desc ]    // default is entity prob.
7  LIMIT with-k  max_entities |                  // iceberg option
8        with-p  min_probability                 // threshold option
```

Figure 3: The query syntax.

Definition 3. *Given a coreferent set M for R^D, then the **canonical entity** e corresponds to an instance that represents all instances in M. An entity e provides values v_1, ..., v_k for the attributes of R^D, with $v_i \in \bigcup_{\forall r \in M} \{r.a_i\}$.* ∎

All in-database resolution approaches either do not deal with the issue of canonicalization or simply return the union of all instances. In contrast to those approaches, we follow the IE approach [31] and construct canonical entities by defining canonical values for each relation's attribute.

Our technique allows users to express queries over the possible coreferent sets and their canonical entities. Fig. 3 shows the complete query syntax. Queries perform the join between tables T_i and T_{dup}, with the latter being the table that contains duplicates. This requires joining the data from table T_i with the possible entities from table T_{dup}.

The query syntax allows users to retrieve results that do not consider all entities, but are limited to specific entities as specified in the given query. More specifically, there are two options: (i) restrict based on threshold, in which the results consider all entities that have an overall probability higher that a given *min_probability* (line 8), and (ii) restrict based on the number of results, in which the results consider the entities placed within the first *max_entities* (line 7). Sorting is performed using a given *attr* (line 6) with the default being the overall probability.

Another aspect of our query syntax is the capability to express queries in which the conditions are based on either the canonical entities (line 5), or the instances from which the canonical entity was created (line 4). Recall that a canonical entity is generated from a set of coreferent instances, and that each attribute value in the canonical entity is taken from the union of attribute values of these instances. Thus, the results given conditions over the canonical entity (tight mode) will be significantly less than the results given conditions over the corresponding instances (loose mode). Note that although expressing query conditions using the original data is rarely allowed in the literature (e.g., [28, 10]), it is considered an important feature with respect to resolution. This is because it allows returning answers even when query contains data that are not in the final entities (if this not supported, users would simply receive an empty result set).

3. CANOPY FACTOR GRAPHS

We now introduce our canopy-based CRF model, termed *Canopy Factor Graphs*, a novel approach for encoding the approximate joint conditional distribution of the factor graphs representing the tasks of coreference and canonicalization.

Existing Methodologies. Our work follows existing IE approaches that jointly model extraction tasks, e.g., [15, 31]. In short, the underlying model requires considering all possible arbitrary partitions (clusterings) of the R^D instances into coreferent sets. For each partition, the model assesses

the compatibility of the instances in each cluster, as well as the compatibility across clusters (details provided in the following paragraphs). Solving resolution using all partitions of the R^D instances implies investigating a total of $2^{|R^D|}$ sets. Since probabilistic models of this size are clearly impractical, existing approaches typically rely on simple heuristics (e.g., greedy agglomerative clustering [29]).

Restricting Possible Worlds through Canopy Clustering. We introduce a novel, approximate graphical model that basically creates and uses sets of instances (based on canopy clustering [16]), rather than individual instances. This drastically reduces the number of partitions that must be considered in comparison to existing techniques.

Consider function SIM(r_i,r_j) that measures the similarity of instances r_i and r_j. Existing literature on resolution, indicates that it is easy to detect the instances that do not match each other because these will return a low similarity, i.e., SIM(r_i,r_j)$\leq t_2$, as well as the instances that match because these will return a high similarity, i.e., SIM(r_i,r_j)$>t_1$. The challenge lies with instances that return a similarity that does not allow us to distinguish whether they match or not, i.e., $t_2<$SIM(r_i,r_j)$\leq t_1$. Dealing with the latter is the focus of canopy clustering [16], which we also adopt and extend in this work.

Definition 4. *A* ***canopy***$_i$ *is a set of instances. It contains at least one instance r_c, named centroid, and all instances r_i of R^D with SIM(r_c,r_i)$>t_2$. In addition, instance set C_i contains the "core" instances of canopy$_i$ with SIM(r_c, r_i)$>t_1$. Core instances in C_i are present only in canopy$_i$, whereas the remaining instances, i.e., canopy$_i\backslash C_i$, might also participate in other canopies.* ∎

Given a relation R^D, canopy clustering generates a set of (possibly) overlapping canopies. For example, let us assume that the instances of Buyer create 3 canopies canopy$_1$, canopy$_2$, and canopy$_3$, despicted in the Fig. 4.a using colors blue, red, and green, respectively. As shown, canopy$_1$ contains r_1, r_2, r_3, r_7, and r_8. The centroid is r_1 and it has similarity higher than t_1 only with r_2, thus C_A={r_1,r_2}. Similarly, r_4 and r_5 belong only to canopy$_2$ (i.e., C_B) and r_6 to canopy$_3$ (i.e., C_C). This means that for resolution we now need to process C_A, C_B and C_C instead of the five corresponding instances.

Definition 5. *Let canopies={canopy$_i$} denote the set of canopies generated by canopy clustering over R^D. Canopy intersections create* ***overlapping areas****, one for each possible subset of the canopies: $\exists\, O^S$ for each $S\in 2^{canopies}$ (i.e., the power set of the canopies set). An instance $r_i\in O^S$ iff $r_i\in$canopy$_i$ $\forall$canopy$_i\in S$.* ∎

Generating the canopies of R^D allows us to detect sets of core instances that should be considered together, i.e., the C_i of each canopy$_i$. Def. 5 states that we can create sets from the non-core instances (i.e., the canopy overlaps), and thus allow resolution to also process them as sets rather than individual instances. We consider one overlapping area O for each possible intersection between canopies (given by the power set of canopies). Clearly, an instance can belong to more than one canopy intersections, e.g., r_3 in Fig. 4.a belongs to the overlapping areas of the intersections between (i) canopy$_1$, canopy$_2$, and canopy$_3$, (ii) canopy$_1$ and canopy$_2$, etc. We avoid having an instance in more than one O's by considering it as part of the intersection with the

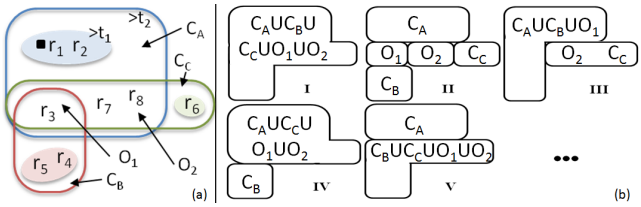

Figure 4: An illustration of (a) three overlapping canopies and (b) some of their possible coreferent sets.

maximum number of canopies, e.g., r_3 is included in O_1 that denotes the intersection of all three canopies.

Following Def. 5, a canopy$_i$ corresponds to $C_i\cup\{O_j\}$. We now continue by explaining how to use the O's and C's from the canopies to create partitions over R^D.

Definition 6. *The* ***alternatives*** *give the possible coreferent instances (i) locally for each canopy$_i$=$C_i\cup\{O_j\}$ as alternatives(canopy$_i$) = $\{C_i\}\times 2^{\{O_j\}}$ (i.e., C_i and all possible subsets of overlaps in canopy$_i$); and, (ii) globally over all canopies as alternatives(canopies) = $A_1 \times \ldots \times A_k$ where A_i=alternatives(canopy$_i$).* ∎

In other words, the alternatives for canopy$_i$ allow for C_i to be merged with any possible subset of $\{O_j\}$. Intuitively, these results can be seen as possible coreferent instances that our resolution algorithm must process. As an example, consider canopy$_1$=$C_A\cup O_1\cup O_2$. The *alternatives* results in {{C_A}, {C_A,O_1}, {C_A,O_2}, {C_A,O_1, O_2}}. Thus, e.g., we should consider instances of C_A and O_1 as coreferences. Similarly, the *alternatives* for canopy$_2$ gives {{C_B}, {C_B,O_1}}, and for canopy$_3$ {{C_C}, {C_C,O_1}, {C_C, O_2}, {C_C,O_1,O_2}}. The cartesian product of all canopy alternatives creates a global view of all possible coreferent instance sets, e.g., {{C_A}, {C_B}, {C_C}} and {{C_A}, {C_B,O_1}, {C_C,O_2}}. (Slightly abusing notation, we also assume that {C_i, O_j, O_k, ...} is equivalent to $C_i\cup O_j\cup O_k\cup\ldots$, i.e., the set of all instances in the corresponding alternative).

A global alternative can essentially be viewed as a partitioning of instances that is "compatible" with the derived canopies. More specifically, given a global alternative {set_1, ..., set_k} $\in A_1 \times \ldots \times A_k$, we create a partitioning by (i) merging overlapping sets (with common O's), since these provide instances that are part of the same coreferent partition, and (ii) including as a coreferent partition instances of overlaps O's not in the alternatives. This construction is formalized in the following definition.

Definition 7. *A global alternative {set_1, ..., set_k} $\in A_1 \times \ldots \times A_k$ corresponds to a partitioning of instances CM={M_1,...,M_j} constructed as follows:*

- *Merge overlapping sets: $set^i_1\cup\ldots\cup set^i_k\in CM$ if for each set^i_j there exists a set^i_l with $l\neq j$ and an overlap set $O\in set^i_j\cap set^i_l$*
- *Retain sets not overlapping any others: $set_i\in CM$ if $\forall O\in set_i$ $\nexists set_j$ s.t. $O\in set_j$*
- *Include all remaining overlaps: {O_l}$\in CM$ if $O_l\notin set_1$, $O_l\notin set_2$, ..., $O_l\notin set_k$* ∎

As an example, global alternative {{C_A,O_1,O_2}, {C_B}, {C_C,O_1}} becomes partition {{C_B}, {C_A,C_C,O_1,O_2}}, illustrated in Fig. 4.b(V); and {{C_A}, {C_B}, {C_C}} becomes partition {{C_A}, {C_B}, {C_C}, {O_1}, {O_2}}, Fig. 4.b(II).

Note that the numnber of local alternatives for canopy$_i$ is 2^{n_i}, with n_i being the number of O's it contains. Given

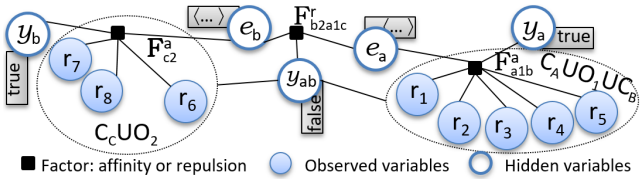

Figure 5: Illustration of one variable assignment.

k canopies, it is easy to see that there is a maximum total of $2^{n_1} \times 2^{n_2} \times ... 2^{n_k}$ possible coreferent instance partitions. Thus, solving resolution over our small example data set through the possible R^D partitions results in 256 (i.e., 2^8) coreferent partitions, while canopies give only 32 (i.e., $2^2 + 2^1 + 2^2$) partitions. Clearly, the reduction can be much more dramatic for larger $|R^D|$.

Canopy Factor Graphs (CFGs). The previous paragraphs discussed the generation of possible coreferent sets (e.g., Fig. 4.b). Still, this is not sufficient to address our resolution problem, since we also *need to decide which are the most probable coreferent sets*. For example, in Fig. 4.b, coreferent set III might be more probable than I and II. This can be decided by also encoding additional important resolution information, primarily *affinity* between the instances of a coreferent set and *repulsion* across two distinct coreferent sets. We model this information through Conditional Random Fields (CRFs) [25] — a graphical model that has been successfully used by IE for handling entity resolution in a principled manner. It captures the conditional distribution of hidden/unknown variables given a collection of observations (i.e., known variables) [13]. This model can be used for representing the relationships between the collection of observations and for generating a possible interpretation (i.e., an assignment over the unknown variables) by executing an inference algorithm.

In our model, the observed variables are the R^D instances along with the canopies generated by these instances. Let x_i denote a particular coreferent set. Given the canopies, and thus $\{C_1, ..., C_m\}$ and $\{O_1, ..., O_k\}$, the domain of x_i is $\cup_{j=1}^{n} alternatives(\text{canopy}_j)\ \cup_{l=1}^{k} O_l$ (i.e., the set of all possible instance partitions compatible with our derived canopies). The unobserved variables indicate the compatibility within and across x_1, x_2, ... More specifically, the (binary) variables denoted with y_i indicate whether the instances in set x_i are coreferent, and the (binary) variables denoted with y_{ij} indicate whether the instances in set x_i are coreferent with the ones in set x_j. The set Y=$\{y_i, ..., y_{ij}, ...\}$ contains all unobserved binary variables that we wish to predict. In addition, set E=$\{e_i\}$ contains the canonical entities corresponding to the collection of coreferent sets, i.e., $\{x_i\}$.

Our proposed CFG graphical model employs factor graphs as the associated graphical structure for the CRF [25] over the canopies. A factor graph is constructed using observed and hidden variables that represent the various resolution scenarios. Conditional independence across variables is used to decompose complex probability distributions into a product of *factors*, each consisting of a smaller subset of variables. In essence, having multiple factors for the same variable allows us to express preferences of certain assignments with respect to other assignments. We consider two factor types:

I. Affinity Factors - $F^a(y_i, x_i)$: measure the compatibility between the assignment to variables y_i with respect to a given instance set x_i, i.e., if y_i correctly indicates whether the x_i instances are coreferent.

II. Repulsion Factors - $F^r(y_{ij}, e_i, e_j)$: measure the compatibility of the assignment to variable y_{ij}, i.e., coreferent given the canonical entities (i.e., e_i and e_j) of two instance sets (i.e., x_i and x_j).

Definition 8. *The conditional probability distribution given the C's and O's is computed using the following formula:*

$$P(Y,\ E \mid \Lambda, \{O_1, ...\}, \{C_1, ...\}) = \frac{1}{Z} \times \prod_{\forall y_i \in Y} F^a(y_i, x_i) \times \prod_{\forall y_{ij} \in Y} F^r(y_{ij}, e_i, e_j) \quad (1)$$

where Z is the input-dependent normalizer and Λ a vector containing the weight of each feature function. The two factor types have this form: $F^T(.) = exp\big(\sum_k \lambda_k^T \phi(T, k, .)\ \big)$ with ϕ being a positive real-valued feature function that equals to $\phi_k^T(.)$ for F^a and 1-$\phi_k^T(.)$ for F^r. Symbol $\lambda_k^T \in \Lambda$ denotes the weight for the corresponding function $\phi(T, k, .)$. ∎

This model follows existing IE approaches, e.g., [31]. Each affinity factor indicates if the corresponding instance set indeed contain coreferent instances, and each repulsion factor indicates if two instance sets should be actually merged together. Note that this a particular graphical model in which inference corresponds to graph partitioning. Each possible partition of the instances creates a different set of Y variables, i.e., Y=$\{y_i, ..., y_{ij}, ...\}$ as well as a different assignment to those instantiated variables. Intuitively, this means that the graph can have any arbitrary structure, i.e., the set of factors depend on the particular entity assignment.

The collection of feature functions is associated with the specific data domain on which we apply the model. Sec. 6 presents the feature functions, i.e., $\phi(T, k, .)$. The values of all $\lambda_k^T \in \Lambda$ are determined through training.

Example 1. *Fig. 5 gives a graphical illustration of the resulting factor graph for one of the possible variable assignments of the Fig. 4 canopies. There are two x variables: x_1=$C_A \cup C_B \cup O_1$ and x_2=$C_C \cup O_2$ (i.e., Fig. 4.b-III). The probability of the particular variable assignment is computed using Eq. 1 and corresponds to entities e_a and e_b.* ∎

Def. 8 requires executing an inference algorithm over our CFG graphical model. However, exact inference and learning in such models is known to be intractable (e.g., [31]). To address the problem, in the following section, we introduce an approximate solution, and we experimentally demonstrate that our solution has no serious drawbacks on the quality of created entities.

Canonicalization. Our model uses one canonical entity e_i for each of the detected coreferent sets M_i (Def. 3). Generating the canonical entity is performed through the canonicalization process. We consider a generic form of canonicalization, and thus different possible semantics that canonicalization must be able to capture.

The most typical method, which is followed by the majority of the existing resolution approaches [3], is to consider a canonicalization process that uses one of the instances of the coreferent set as the canonical entity, i.e.,: $c_{basic} \mid M \mapsto r_i$, where $r_i \in M$. We will refer to this process as *basic canonicalization*. An example implementation for the basic canonicalization is to return the instance that was more recently included in the data.

Another, more complex than the previous methodology, is to create canonical entities composed using data from all the corresponding coreferent, as for example discussed in [28]. We will refer to this process as *generic canonicalization*.

The processing is based on the average similarity between the value of an instance r to the corresponding values of all other instances $r' \in M$. It is computed as follows:

$$avg\_sim(r.a_i, M) = \frac{1}{|M|} \times \sum_{\forall r' \in M,\ r \neq r'} SIM(r.a_i, r'.a_i),$$

where function $SIM(r.a_i, r'.a_i)$ gives the similarity between $r.a_i$ and $r'.a_i$. The value $r'.a_i$ that has the highest similarity with $r.a_i$ is then incorporated in the canonical entity. Thus, given the coreferent set M that contains duplicate instances of the same real world object, the canonical value for each attribute a_i is retrieved by: $c_{generic} \mid M \mapsto r, if\ \forall r, r' \in M$ avg_sim$(r.a_i, M) \geq$ avg_sim$(r'.a_i, M)$.

4. HOLISTIC QUERY PROCESSING

Consider again the conditional probability distribution in Def. 8. As already explained, inference corresponds to graph partitioning, i.e., detecting which grouping of the instances into coreferent sets gives the highest probability based on Eq.1. This section introduces an approximate solution for detecting the most probable coreferent sets. The process starts from the canopies (Sec. 4.1) and derives the k best coreferent sets by deciding the merges between Os and Cs. This is achieved by deriving and using a simpler correlation between the canopy subsets, i.e., absorb/detach (Sec. 4.2). Each of these k best coreferent sets provides variable instantiation of Y, which basically corresponds to defining the structure of the factor graph. Over the resulting factor graphs we compute the overall probability. The process also uses the query information to generate a partial instantiation and assignment over the Y variables (Sec. 4.3) and thus allows focusing only on situations that provide entities that are also related to query results (Sec. 5).

Note that our query processing approach introduces two distinct levels of approximation. The first is an approximation over the probabilistic model in order to estimate the joint conditional distribution using a novel, more concise graphical model structure (exploiting the canopies). The second is a new approximate technique for performing inference over our CFG model. Our proposed optimizations (based on equivalent query rewritings) improve query performance by significantly reducing processing time.

4.1 Generation of Canopies

Generating the canopies is a preparation step as it considers only the duplicated instances, without taking into consideration the query information. As such, it is executed only once, and the generated structures are updated only when new instances are added in the specific relation. The process follows the original canopy clustering algorithm with an extension for considering C/Os.

The algorithm is as follows: First, we include in list S all R^D instances. We randomly select an instance r from S and compute its similarity with all other instances from S. Instances with a similarity higher than threshold t_2 are included in the canopy of r, and excluded from S when this similarity is higher than t_1. The idea behind this process is that we detect two type of instances with respect to a specific r: (i) instances we are confident that are coreferent with r and as such we also do not need to continue comparing them, and (ii) instances that might be coreferent with r, but as the similarity is not high enough for a confident decision we need to continue comparing them. The remaining instances are not similar to r and are thus ignored. The process results in a set of Cs and Os (e.g., see Fig. 4.a).

Table 1: Meaning of canopy-related information.

$a(C_i,O_j) = \{\ C_i \cup O_j\ \}$	$d(C_i,C_j) = \{\ C_i,\ C_j\ \}$
$\neg a(C_i,O_j) = \{\ C_i,\ O_j\ \}$	$\neg d(C_i,C_j) = \{\ C_i \cup C_j\ \}$

4.2 Best Coreferent Sets

Given the generated canopies, we can now generate the coreferent sets with the highest scores. Consider again Eq. 1, and in particular the included product. The k best coreferent sets correspond to the k combinations for affinity and repulsion factors that have the highest overall scores among all possible combinations. Generating the k best coreferent sets is achieved through four steps, which are described in the following paragraphs.

I. Independent Canopy Partitions. The first step is to partition the set of canopies into non-overlapping subsets, i.e., canopies of the same partition have common instances, but canopies of different partitions do not have a common instance. Our algorithm bounds each partition to a set of canopies, and thus also to a set of instances, i.e., the ones contained in the canopies of the specific partition.

To detect the independent partitions, we consider an empty set of partitions, and start processing each instance of every canopy. We first retrieve the partition that contains the specific canopy, and the partition that contains the specific instance. There are three possible situations: (a) The instance and the canopy are not yet placed in a partition, so we create a new partition and include both of them; (b) Only one of them is not placed in a partition, we thus include it in the partition that other belongs to; and (c) The instance is placed in a different partition than the canopy, let's say partitions a and b. In this case, we create another partition in which we include the specific canopy and instance, and also all instances/canopies from partitions a and b.

This step results in a collection of canopy partitions, which allows us to process each canopy partition individually.

II. Canopy-related Information. Detecting the best coreferent sets is based on canopy-related information, which we generate using the Os/Cs groups. In particular, we consider: (i) an absorb $a(C_i,O_j)$ for each pair C_i and O_j from the same canopy, and (ii) a detach $d(C_i,C_j)$ for pair C_i and O_k of canopy$_i$ and pair C_j and O_k of canopy$_j$.

An $a(C_i,O_j)$ represents that the instances of C_i must absorb the instances of O_j. A detach $d(C_i,C_j)$ means that the instances from C_i and C_j must not be merged together. Table 1 provides the meaning of possible canopy-related information. Evidently, $a(C_i,O_j)$ and $\neg d(C_i,C_j)$ provide information about F^as, i.e., we consider a x_i with all Cs/Os for all $a(C_i,O_j)$ and $\neg d(C_i,C_j)$ with common groups. Also, $\neg a(C_i,O_j)$ and $d(C_i,C_j)$ provide information about F^rs, i.e., there is an F^r for sets x_i and x_j with $C_i \in x_i$ and $O_j/C_j \in x_j$.

Each absorb and detach is given a score. The score for $a(C_i,O_j)$ is the average similarity of the canopy's center, i.e., r_c of C_i, with all the instances of the overlapping O_j. The score for $d(C_i,C_j)$ is the dissimilarity between the centers of the two canopies, i.e., r_c of C_i with r_c of C_j. The particular similarities were computed during the creation of the canopies and thus we do not need to perform any computation but simply reuse the previous results.

As stated in Eq. 1, F^a and F^r use weighted real-value

features with F^a considering all instances of the corresponding x_i and F^r the two canopy entities for x_i and x_j. The absorb/detach scores used in the current version of our approach provides an estimation, since these use similarities that are actually part of the ones used for the F^a and F^r. Incorporating other scores would require additional comparisons and increase the overall execution time.

The final score of the k best coreferent sets will be revised during the next query processing phases.

III. Partition's best coreferent sets. We now explain how to retrieve the best coreferent sets directly from the canopy-related information created in the previous step. The main idea is that each absorb/detach can be either included in or excluded from a coreferent set, with the latter meaning that we include its negation. The effect of including the absorb/detach, or its negation, is to "append" the coreferent set with the corresponding absorb/detach result listed in Table 1. Alg. 1 gives an overview of the process. It start from the canopy-related information (algorithm's input) and removes absorb/detach at each step (line 9). The purpose of the process, explained in the following paragraphs, is removing absorb/detach (lines 5&7) such that the coreferent sets are generated in decreasing order of probability.

Example 2. *Consider again our example and more specifically Fig. 4.a, which illustrates that for the particular data we have canopy*$_1$ = $C_A \cup O_1 \cup O_2$, *canopy*$_2$ = $C_B \cup O_1$ *and canopy*$_3$ = $C_C \cup O_1 \cup O_2$. *Given these canopies, we create absorb* $a(C_A,O_1)$, $a(C_A,O_2)$, $a(C_B,O_1)$, $a(C_C,O_1)$, *and* $a(C_C,O_2)$, *and detach* $d(C_A,C_B)$, $d(C_A,C_C)$, *and* $d(C_B,C_C)$.

One of the possible combinations for accepting/rejecting the canopy-related information is: $a(C_A,O_1)$; $\neg a(C_A,O_2)$; $a(C_B,O_1)$; $\neg a(C_C,O_1)$; $\neg a(C_C,O_2)$; $\neg d(C_A,C_B)$; $d(C_A,C_C)$; *and* $d(C_B,C_C)$. *It results in the following coreferent set:* $M_1 = C_A \cup C_B \cup O_1$; $M_2 = O_2$; *and* $M_3 = C_C$. *This corresponds to graphical illustration* III *in Fig. 4.b.* ∎

It is easy to see that we can generate a large number of coreferent sets by selecting which absorb/detach to include as is or to negate. We are not interested in retrieving all of them, but just the k with the highest score. To achieve this, we follow an exhaustive search approach in which we use the canopy-related information to first generate the possible coreferent sets in decreasing score and then check whether a generated coreferent set is "valid" based on the specific canopy-related information. A coreferent set is not valid when it contains an independent O_l while none of the C_js that participate in the same absorb with O_l (i.e., $a(C_j,O_l)$) are not independent. The intuition behind this is that instances between Cs of overlapping canopies have lower similarity that instances between the an O and C of overlapping canopies. It is thus senseless to not merge the O and C and merge just the Cs. For example, in Fig. 4.a we cannot the have the merge $C_A \cup C_B \cup C_C \cup O_2$ but leave O_1 independent.

Initially, we check all absorb/detach and negate it when its score is less than 0.5. E.g., if $a(C_A,O_1)$ has score 0.48 we will consider $\neg a(C_A,O_1)$ with 0.52. The canopy-related information (i.e., absorb/detach) is placed in a list I, sorted according to score, from lowest to highest.

Generating coreferent sets in decreasing score is achieved by negating only one absorb or detach from I. A similar idea was used in the *retrieval of linkage combinations* from [9]. A vital difference with our situation is that each node results in a different outcome (in [9], two different combinations could have the same outcome). Thus, the process we introduce here visits only the required combinations.

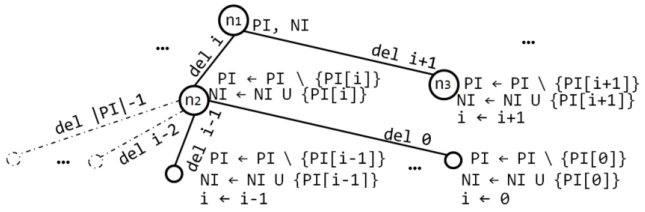

Figure 6: Node n_2 was created by deleting $PI[i]$. We thus generate its children that correspond to deletion of n's PI items with locations $i-1$ until 0.

The process is based on a tree structure with each node corresponding to a partition of the instances. A node provides the absorb/detach from I that is included or excluded. As such, the probability of a node $P(n_i)$ is the product between the scores of included absorb/detach and the complement of the scores of the excluded ones. Each node has two lists, list PI giving the locations of the I items that are included and list NI the ones that are excluded. Initially, i.e., root node, all items are included in the combination and thus PI={0,...,|I|-1} and NI={}. To generate the descendants of a node n, we use the PI and NI lists of node n and exclude an item from PI and include it in the NI.

It is easy to show that a node has score less or equal to the score (i) of its parent, i.e., P(n_3)≤P(n_1) in Fig. 6 and (ii) of its left sibling, i.e., P(n_3)≤P(n_2), since sibling and children nodes are different in just one item, the one with the lower probability. Thus, generating the combinations in a decreasing score is achieved through a depth-first visiting of the nodes. Each time we visit a node we generate its right sibling and its leftmost child. These nodes are included in a list sorted by score. The processing always continues from the highest-score node in the list. In the worst case, we would need to visit all nodes of the tree with the total number of nodes being $1 + \sum_{i=1}^{|I|-1} \left(\prod_{j=1}^{i} (|I| - j + 1) \right)$.

Theorem 1. *Let n be the node that was created by removing the item at location i from its parent PI, i.e., $PI \leftarrow PI \setminus \{PI[i]\}$. We do not need to consider the children of n that correspond to the deletion of the items with locations $|PI|-1$ until $i-2$ from PI since these combinations are found as descendants of the left siblings of node n.* ∎
(Proof in the technical report)

Given the above theorem, we only generate the children nodes of n that correspond to the deletion of the items with locations $i-1$ until 0 from PI (illustrated in Figure 6). This reduces the number of nodes we need to visit. In the worst case the number of nodes we need to visit is now:

$$1 + \sum_{i=1}^{|I|-1} \left(\frac{\prod_{j=1}^{i} (|I| - j + 1)}{i!} \right)$$

IV. Probabilities. During the previous steps we derived the k best coreferent sets, with each coreferent set {M_1, ..., M_n} encoding a possible assignment for the coreferent variables Y. In this step, we apply the canonicalization process on each included coreferent set M. This will provide an assignment for the canonical entity variables in E.

The probability computation during this step corresponds to the conditional probability distribution that was defined and discussed in Def. 8. For computing this probability, we

Algorithm 1: Retrieve the k best coreferent sets

Input: I: canopy-related information (i.e., absorb/detach), k: number of results
Output: *kbest*: a set with the k best coreferent set

```
1  node←new Node(); node.PI←I; node.NI←{};
2  node.delpos←I.size(); // position of PI deletion
3  kbest←{}; max_heap←{node};
4  while ( kbest.size()<k ) do
5      node←max_heap.removeFirst();
6      kbest←kbest∪node;
7      for ( i=node.delpos-1 to 0 ) do
8          node_child ←new Node(); node_child.delpos← i;
9          node_child.PI← node.PI / {PI[i]};
10         node_child.NI← node.NI ∪ {PI[i]};
11         max_heap←max_heap∪{node_child};
12 return kbest;
```

use the canonicalization process presented in Sec. 3. Note that in the described query execution technique we always assume that the generic canonicalization process is used. The basic canonicalization process is easier to apply and it requires less computational resources. This is further discussed with the algebraic optimizations introduced in Sec. 5.

During this step, we first apply the canonicalization process over each coreferent M. Applying the generic canonicalization process initially computes the average string similarity between the value of each instance r with the corresponding values of all other instances. Since this step is part of the query processing, we include the value in the canonical entity when we detect that this has the highest average similarity and that it also satisfies the query conditions.

Each of the k best coreferent sets results in an instantiation and assignment of the Y variables as well as an e_i for each M_i. The assignment of the Y variables is then used for computing the overall probability, i.e., Def. 8. Recall that each of the k best coreferent sets corresponds to a possible world, as for example the one shown in Fig. 5.

4.3 Partial Assignment

As explained at the beginning of this section, we want to reduce the query processing time using information from the query. In particular, we use the CANONICAL-ENTITY conditions (line 5, Fig. 3) to fix values within the resulting canonical entities (step 1) and the COREFER-INSTANCE conditions (line 4) to fix the instances of the final entities (step 2).

Step 1 - Canonical-entity Conditions. The variable denoted with E is the final set of canonical entities for the coreferent sets generated for the R^D instances. The CANONICAL-ENTITY conditions are converted into a partial assignment over E, i.e., variable assignments that satisfy the query. The result is denoted with symbol E_p.

We consider the CANONICAL-ENTITY conditions of the query as one expression and convert it into disjunctive normal form (DNF). We thus have $\text{clause}_1 \vee ... \vee ... \text{clause}_n$. Each clause_i is equal to $\text{cond}_1 \wedge \text{cond}_2 \wedge ....$ For relation R^D, defined as tuple $\langle a_1, a_2, ...\rangle$, then $E_p = \bigcup\{\langle v_1, v_2, ...\rangle\}$, where each $\langle v_1, v_2, ...\rangle$ corresponds to a clause_k and

$$v_i = \begin{cases} \text{cond}_j.\text{value}, & \exists\ \text{cond}_j \in \text{clause}_k \ \ \& \ \ \text{cond}_j.\text{attr} = a_i \\ \textit{unspecified}, & \text{otherwise.} \end{cases}$$

Step 2 - Corefer-instance Conditions. We convert these conditions into a partial assignment over the hidden variables representing the coreferent between an instance set (i.e., y_i) or between two instance sets (i.e., y_{ij}). The result is denoted with Y_p. As the Y variables refer to instances, the process for creating Y_p involves converting the information of the query condition, which specify the desired values for the attributes, into instances.

As with CANONICAL-ENTITY conditions, we initially consider an expression with the COREFER-INSTANCE conditions and convert it into DNF. Thus, we have $\text{clause}_1 \vee \text{clause}_2 \vee ...$, and each element of the list is a conjunction of attribute conditions, i.e., $\text{clause}_k = \text{cond}_i \wedge ... \wedge \text{cond}_n$. The next step, takes each element of this list and detects the instances of R^D that satisfy each of the attribute conditions, thus $\text{cond}_i = \{r_j\}$. The next step creates all the possible combinations between the instances detected for each clause, i.e., $\text{clause}_j = \wedge r_i$. The partial assignment for the COREFER-INSTANCE is the disjunction between all these clauses. It has the following form: $Y_p = \bigvee (\bigwedge r_i)$.

Step 3 - Creating the Partial Assignment. Generating the possible worlds can be performed based on E_p and Y_p over the hidden variables E and Y. More specifically, the possible worlds (denoted with symbol *pwd*) are:

$$pwd(E_p, Y_p) = \cup\{\ \langle Y, E\rangle \mid \exists e_i \in E \Rightarrow \exists E_p.clause \in e_i \ and \ \exists y_i \in Y \Rightarrow y_i = true \wedge \exists Y_p.clause \subseteq M_i\} \quad (2)$$

The partial assignment can be seen as constraints that E and Y must satisfy for being valid answers to the given query. For this, in the rest of the document the terms partial assignment and constraints will be considered equivalent.

Example 3. *Consider a query with a DNF* CANONICAL-ENTITY *condition $cond_1 \wedge cond_2$, where $cond_1$ is surname like "H%, Dr." and $cond_2$ is country = "USA". Then, E_p is $\{\langle \cdot, \cdot, \cdot, \text{"Hill, Dr."}, \cdot, \text{"USA"}\rangle\}$. This means that we need to find possible worlds with an entity containing such a tuple. Consider now this is a* COREFER-INSTANCE *condition. The DNF is the same as with the* CANONICAL-ENTITY *conditions. Detecting the instances satisfying each DNF condition gives: $\{r_4\}$ for $cond_1$ and $\{r_2, r_4\}$ for $cond_2$. Thus, $Y_p = (r_4 \wedge r_2) \vee (r_4)$, meaning we need to find possible worlds with a coreferent set containing either r_4 and r_2, or just r_4.* ∎

4.4 Complexity Analysis

The processing time for canopy generation depends on the number of Cs and Os, which basically corresponds to the resolution characteristics. The worst-time scenario is $O(|R^D|^2)$, occurring when each instance is placed in a separate C and the algorithm compares all instances between them. The best-time scenario is $O(|R^D|)$, occurring when all instances are in a single C. Rather unsurprisingly, real life data do not correspond to such scenarios. We further discuss this in Sec. 6. Given the canopies, we generate the best coreferent sets. The first step is creating the independent canopy partitions, which has a complexity $O(|R^D|)$. This is followed by the generation of canopy-related information with complexity $O(n)$, where n is the number of Cs and Os that must be processed. Finally, the complexity of the best coreferent set per partition is $O(k \cdot \log k)$, where k is the number of best coreferent sets we need to retrieve.

Efficient and scalability is achieved through two mechanisms: (a) using canopies and their partitions, i.e., sets of instances, instead of individual instances; and (b) retrieving the variable assignments over the model using the best coreferent sets algorithm.

5. QUERY PLANS & OPTIMIZATIONS

Let us now consider the R_j^D relation. For processing queries we must consider that it contains duplicates; hence

we need to generate all possible entities. This is achieved by applying the k best coreferent, denoted with b_k, followed by canonicalization, denoted with c_f. The related conditions (i.e., lines 4-5 in Fig. 3), are applied over the resulting relations. For this we use the generated partial assignments, i.e., Sec. 4.3. Thus, the resulting relations are given by the following expression: $L = \sigma_{Y_p \wedge E_p} (c_f [b_k (R_j^D)])$

As relation R_i is deterministic, we follow the traditional process, which is a selection given the related conditions (line 3 in Fig. 3). The join is performed between the relations created from R_i and from R_j^D. Over the resulting relation, we apply a monotonic scoring function $F_{min\_p/top\_k,attr,order}$ that keeps a subset of the entities. There are two options for this scoring function: select the records with a probability highest that the specified min_p (line 8), or select the k records that have the highest values for attribute $attr$ in ascending or descending order as specified by the given $order$ (line 7). The final operator is a projection for selecting the attributes specified by the user (line 1). Thus, the overall expression is as follows: $\pi_{\cdot} (F_{\cdot} [L \bowtie (\sigma_c [R_i])])$

The most challenging issue of managing probabilistic data, is the amount of data (i.e., size of relations) yielded through the generation for the possible worlds of the instances in R_j^D. We are able to reduce the considered possible worlds by generating the k best coreferent sets satisfying both Y_p and E_p. The rationale behind the use of E_p also here (and not just in the process of Sec. 4.2) is that one cannot create a canonical entity satisfying E_p when the instances used do not satisfy E_p, i.e., if the user requests a canonical entity with value v_i for attribute a_i then at least one of the instances used for generating the entity should contain v_i. We denote this operator with b_{k,Y_p,E_p}.

Starting point of this process is the partial assignments, E_p and Y_p. In short, we first convert them into canopy-related information, i.e., a set of absorb/detach conditions I'. We then *force* the k best coreferent sets process (Alg. 1) to treat them as obligatory elements. We achieve this by not negating them and always including them as are in the generated representations, i.e., Alg. 1 receives as input $I \backslash I$' and not I. The detailed process for creating I' follows.

Consider again the partial assignments (Sec. 4.3). They are a disjunction of clauses, i.e., $Y_p = \bigvee clause_{yi}$ and $E_p = \bigvee clause_{ej}$ where $clause_{yi} = \wedge r_i$ and $clause_{ei} = \wedge (a_j = v_j)$. A canonical entity would satisfy both of them if it satisfies at least one of their clauses, i.e., $clause_{yi} \wedge clause_{ej}$. All such expressions can be derived by considering the possible combinations between the clauses of Y_p with the clauses of E_p. Recall that each expression is a conjunction of instances, and that we actually know the group (i.e., C_i/O_j) in which each instance belongs. By replacing each instance with its group, we derive the groups that should be merged in order to create an entity that satisfies the given query conditions. We consider only the C_i groups, so in case an instance is part of an O_j group we instead use all C_i for which there is an $a(C_i,O_k)$. We denote the resulting expression with symbol YE_p^o and this has the form: $\bigvee (\bigwedge C_i)$.

The challenge here is to efficiently detect the corresponding detach information to place as obligatory elements during the k best coreferent sets process using YE_p^o. This is achieved by converting it into a graph traversal problem. To create the graph we use only the detach information. More specifically, for each $d(C_i,C_j)$ we create node C_i, node C_j, and an edge between C_i and C_j. The probability

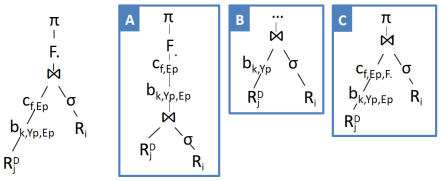

Figure 7: The generic and optimized query plans.

that the edge between C_i and C_j exists is $[1\text{-P}(d(C_i,C_j))] \cdot \text{P}(a(C_i,O_k)) \cdot \text{P}(a(C_j,O_k))$. The probability that this edge does not exist is $\text{P}(d(C_i,C_j)) \cdot [1\text{-P}(a(C_i,O_k)) \cdot \text{P}(a(C_j,O_k))]$. The latter implies that the instances in C_i should be merged into one entity, the instances in C_j should be merged into a different entity, and the instances in O_k might either be in one of those two entities or even in an a third independent entity. Given this graph, we detect the paths satisfying the YE_p^o expression, i.e., connecting all required C_i. A path is a set of detach information, which we then use to create I'.

Example 4. *Consider having the absorb/detach of Ex. 3 with Y_p=$r_4 \wedge r_2$. Instance $r_2 \in C_A$ and $r_4 \in C_B$, which means that for satisfying Y_p we need $C_A \cup C_B$. We thus exclude $d(C_A,C_C)$ from the input I of Alg. 1 and consider always $\neg d(C_A,C_C)$. This forces the generated best coreferent sets to satisfy the given conditions.* ∎

Consider again the canonicalization process for each of the k coreferent sets. For each coreferent set M it generates a canonical entity by combining values from the instances of M, and then verifies that a value satisfies the given query. We can push the query conditions in the canonicalization process and thus avoid processing that will be then ignored because it's results do not satisfy the query. To achieve this, we use E_p and define the attribute values that should be included in the canonical entities. These attribute values are excluded from the canonicalization process. This is performed by operator c_{f,E_P}.

We present algebraic optimizations that rewrite the generic query plan into optimized query plans that require less computational resources (illustrated in Fig. 7).

A. Performing the join earlier. The optimization introduced in A and B modify the original query plan to into:

$$\pi_{\cdot} [F_{\cdot} (L' \bowtie N)], \quad \text{where} \qquad (3)$$

$$L' = c_{f,E_P} [b_{k,Y_p,E_p} (R_j^D)] \quad \text{and} \quad N = \sigma_c (R_i)$$

Executing the c_{f,E_P} and b_{k,Y_p,E_p} operators, i.e., best coreferent sets and computing probabilities, has a higher cost when increasing the considered instances.

For some queries, we can reduce the number of instances given to these operators by performing the join earlier, i.e., before the c_{f,E_P} and b_{k,Y_p,E_p} operators. These are queries in which the selectivity of the distinct instances of N is higher than the selectivity of R_j^D instances satisfying Y_p and E_p. Thus, (in this case) performing the join between the N and R_j^D will give less instances to the c_{f,E_P} and b_{k,Y_p,E_p} operators, and this reduces the overall execution cost. The optimization is: $L' \bowtie N \equiv c_{f,E_P} [b_{k,Y_p,E_p} (R_j^D \bowtie N)]$

B. Basic Canonicalization Process. Using the basic canonicalization process means that analyzing the CANONICAL-ENTITY conditions (Sec. 4.3) of a query (line 5, Fig. 3) involves instances and not pairs between attributes and values. Thus, E_p is now a $\bigvee (\bigwedge r_i)$. Since this is similar to the information in Y_p we consider the information from E_p as part of Y_p and discard E_p from query processing. We therefore have an optimization with two modifications: (i)

Table 2: Statistics for the collections with the synthetic data sets as well as for the three real data sets.

	C-1	C-2	B-1	B-2	B-3	B-4	A-1	A-2	A-3	A-4	A-5	Cora	CiteSeer	DBLP+ACM
size (#instances)	20.000		20.000	11.000	8.800	3.900	20.000	16.000	12.000	8.000	2.000	1.504	1.031	4.671
duplic. vs. clean	10%	6%	10%				10%	8%	7%	6%	4%	107%	142%	166%
max inst. per entity	7	6	7	6	5	4	7					21	21	2
number of entities	18.528	19.048	18.528	10.209	8.188	3.647	18.528	15.034	11.515	7.836	1.935	862	558	2.552
canopies	17.739	18.115	17.739	9.765	7.803	3.480	17.739	15.096	10.953	7.398	1.825	816	399	2.436
areas	262	171	262	145	102	47	262	180	116	53	26	46	20	138

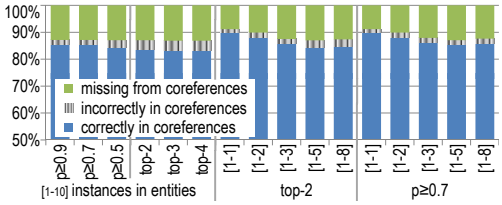

Figure 8: Query accuracy for Cora.

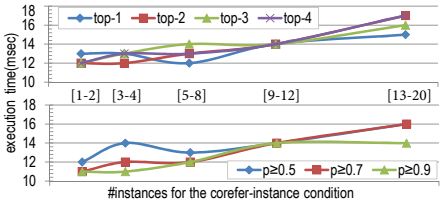

Figure 9: Execution time for Cora.

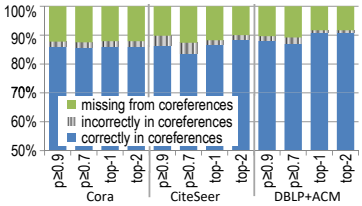

Figure 10: Accuracy for real data.

the c_{f,E_P} operator is no longer needed; and (ii) the b_{k,Y_p,E_p} is converted into b_{k,Y_p} because the information from E_p is now transferred into Y_p. The resulting optimization is: $L' \bowtie N \equiv [\, b_{k,Y_p} \, (\, R_j^D \bowtie N \,) \,]$

C. Iceberg Queries with Alternative Attribute. The scoring function $F_.$ can be based either on sorting the resulting entities according to probability, or on sorting them according the values of a specified attribute. For being able to perform this operator when this is based on the entity probabilities we must first execute the generic query plan. However, this is not needed for being able to perform this operator when this is based on another attribute. This means that we can execute the process over a smaller number of entities, which will reduce the total execution time.

To incorporate this modification in the query processing, we introduce a modified version of operator for canonicalization, i.e., c_{f,E_p}. More specifically, this operator now also includes the functionality of the $F_.$ operator. The new operator, denoted with $c_{f,E_p,F_.}$, first selects the k records that have the highest values for attribute *attr*, sorted either ascending or descending according to the query, and then executes the process only over these entities.

6. EXPERIMENTAL EVALUATION

We now present the evaluation results. All experiments were executed on a single core of an Intel Core i7, clocked at 2.5GHz, and used a maximum of 1GB memory. Data was stored in MySQL 5.6, maintained on a 5400rpm HDD.

Data sets. We used three data sets with real publications: (i) *Cora*, which is typically used to evaluate resolution techniques; (ii) *CiteSeer* used in the Alchemy project [1]; and (iii) *DBLP+ACM* from a recent Entity Blocking survey [19]. We also used synthetic data to study the following characteristics: (A) maximum number of coreferent instances; (B) percentage of duplicated vs. clean instances; and (C) size, i.e., number of instances. We used the data generator [11] to create clean and duplicated instances (using misspellings over attributes and values, abbreviations, and permutations). We then created various collections of data sets, with each collection having a fixed value on one of the three characteristics and an increased number for the other two. Table 2 shows data set statistics while http://www.softnet.tuc.gr/~ioannou/data.html gives access to the synthetic data and the generator's configuration.

Queries. We created queries by converting a randomly selected attribute-value pairs into conditions. Depending on the experiment, queries contained a COREFER-INSTANCE, a CANONICAL-ENTITY condition, or both conditions. Each of the following evaluations used 500 queries.

Feature Sets. For the features of Eq. 1, we followed [29] and used *real-value* and *boolean-value*. The real-value features correspond to the TFxIDF cosine similarity between the same attributes of two publications, i.e., (i) titles, (ii) authors, and (iii) venues. Aggregating over a set of publications means aggregating the above results to all pairs in the set, i.e., considering the (i) average of their results, (ii) maximum result, (iii) minimum result, and (iv) number of results that are above a threshold. The boolean-value features are based on testing whether there is a full match between the same attributes of two publications, i.e., (i) titles, (ii) dates, (iii) venues, (iv) authors, (v) page numbers, (vi) volumes, and (vii) publishers. Aggregating over a set of publications includes taking the (i) average number of times the feature is true/valid, (ii) testing whether the majority results are true, and (iii) testing whether the minority results are true. The values of all $\lambda_k^T \in \Lambda$ were determined through training by randomly selecting 5% from the records of each data set. Training requires 550 msec for A1 (i.e., the largest data set).

1. Existing Methodologies. Existing approaches do not support the proposed query syntax (Sec. 7). To remedy this, we compare against methodologies closely related to the processing of our approach and more specifically on clustering techniques: **(i) GAC-CanopyBased** [16] is a greedy agglomerative clustering over the results returned by canopy clustering; **(ii) MRCL-Standard** [20] corresponds to a naive Bayes classifier in which each feature contributes independently to the overall probability; and **(iii) MRCL-Collective** [20] performs simultaneous inference of all candidate matching pairs (modeled using CRFs) while allowing information to propagate based on relationships.

These three methodologies return the data partitioned in clusters and the instances in each cluster correspond to a different object, i.e., the solution with the highest probability. We simulate this behavior by the following process: We start from a list with all publications and issue one query for each publication in the list. This is an iceberg with k=1 and a CANONICAL-ENTITY condition for the publication's title. This publication along with all returned coreferent publications are removed from the list. The final resolution solution (i.e., clustering of the publications) occurs when

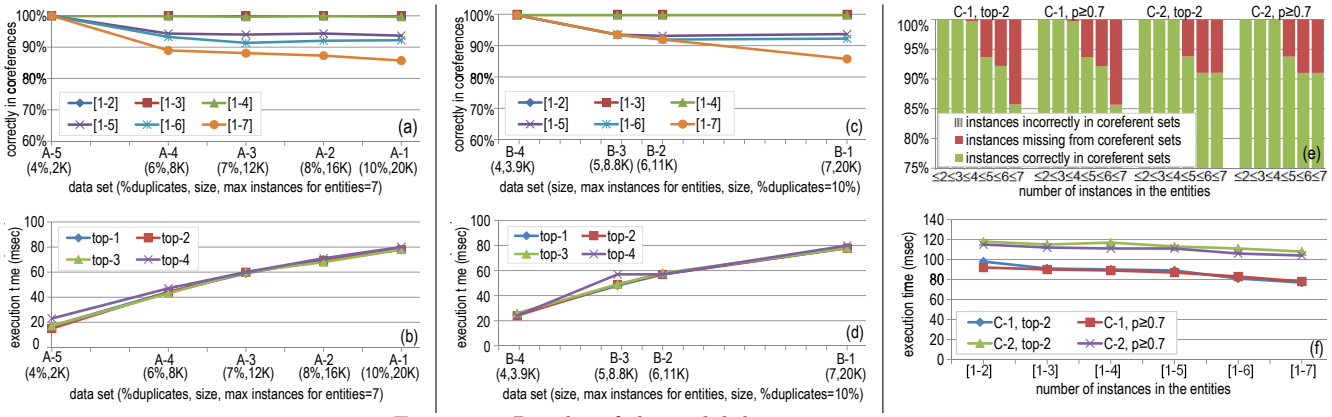

Figure 11: Results of the scalability experiments.

the list is empty. Given this solution we check the decision of each publication pair and measure (as in [16]): precision as the fraction of correct pairs among all pairs of publications found to be coreferent; recall as the fraction of correct pairs among all pairs of publications found as coreferent; and f-measure as weighted average of the precision and recall. Precision, recall and f-measure of the different approaches are: our approach 0.75, 0.87, 0.81; GAC-CanopyBased [16] 0.74, 0.97, 0.84; MRCL-Standard [20] 0.74, 0.92, 0.80; and MRCL-Collective [20] 0.84, 0.91, 0.87. The results indicate that our approach has similar effectiveness as GAC-CanopyBased and MRCL-Standard. MRCL-Collective has higher effectiveness but this is a specialized approach that focuses on collective resolution. Although this evaluation allowed us to compare effectiveness, we need to note that our capabilities are beyond clustering (Sec. 1).

2. Accuracy of Results. Accuracy depends on the resulted coreferent sets as well as canonical entities. The former is directly measured using the instances placed in the coreferent sets with respect to the ones included in the ground truth. Thus, we compute the number of instances that are: (i) correctly placed in the coreferent set, (ii) missing from the coreferent set, and (iii) incorrectly placed in the coreferent set, i.e., belong to another set. Fig. 8 reports results over Cora. As shown, for threshold queries with probability 0.5 to 0.9 a total of 84% instances are correctly placed in the coreferent sets. Decreasing the probability reduces the instances correctly placed in the coreferent sets, i.e., 1.22% between probabilities 0.5 and 0.7. This is an expected behavior, since decreasing the probability *forces* the approach to return more answers that might include less accurate or incomplete results, i.e., instances are missing from the coreferences. The figure also includes iceberg queries with k being 2, 3 and 4. The results indicate high accuracy, since 84% percent of instances are placed in the correct coreference sets (i.e., considering top-2). As expected, quality slightly reduces as the value of k gets larger.

3. Real Data Sets. We also examined accuracy on other data. Fig. 10 reports accuracy for all real data sets. As shown, results have the same behavior, e.g., accuracy slightly reduced when probability goes from 0.9 to 0.7. We also notice better results for DBLP+ACM, e.g., 0.02 for top-1. This occurs as DBLP+ACM has a maximum of 2 coreferences; much lower than the 21 of Cora and CiteSeer.

4. Execution Time. Fig. 9 reports results for threshold queries over Cora with a probability 0.5, 0.7 and 0.9. As shown, our approach has an average execution time of 16 msec. Reducing the probability increases the execution time, e.g., 15 msec for probability equal to 0.9 and 17 msec for 0.7. This time difference occurs due to the increased number of data/results that the approach needs to deal when reducing the probability. The average execution time of iceberg queries is similar, i.e., 16 msec. Increasing k implies handling a larger number of data and results, and thus slightly increases the execution time.

5. Instances in Entities. As shown in Fig. 8, both iceberg and threshold queries have a high percentage of correctly placed instances in coreferent sets when the entities have a very small number of instances. E.g., threshold queries with probability 0.7 have 90% correctly placed instances when entities have 1 instance and 88% for 2 instances. This percentage is reduced when increasing the number entity instances (for both query types). Execution time, Fig. 8&9, is increased when entities have more instances. E.g., for threshold queries and $p \geq 0.7$ we have 2msec for entities with [13-20] instances than with [9-12].

6. Canopy Generation. The generation is an offline process based on random instances (Sec. 4.1). Thus, each generation creates different canopies. Our evaluations showed that canopy generation takes around the same time, with 195 seconds over Cora (avg. over 5 executions). Also, query processing over different executions of this process (i.e., different canopies) showed that the efficiency and effectiveness results are only slightly different. We have also conduct a series of experiments studying performance when using different thresholds, i.e., reducing and increasing the two thresholds. These showed that small modifications (e.g., [0.1-0.5]) have no significant differences on performance. This is expected since one of the benefits of our approach is achieving more accurate results by maintaining and combining decisions/results of the tasks involved in the resolution process.

7. Algebraic Optimizations. The generic canonicalization imposes additional execution time over the basic canonicalization, which was measured to 3 msec when less that 10 instances are involved and 5 msec for less than 21 instances. This is quite low and does cause a considerable overhead, even when queries require the execution of a number of canonicalizations. We also investigated the effect of performing the join earlier and detected that this optimization is especially beneficial when queries contains conditions over R_i that select a small number of instances. Time was reduced by a factor of 5 for DBLP+ACM and 10 for Cora (differences in accuracy are less than 0.02). Cora has a higher benefit since its entities are composed by a

larger maximum of instances. Using the optimization for sorting by an attribute other than probability showed that time was reduced by a factor of 3 for DBLP+ACM and 2 for Cora (with negligible accuracy differences).

8. Scalability. For these experiments we used the synthetic data. Fig. 11.a&b show accuracy and execution time for collection A and Fig. 11.c&d for collection B. The results show that increasing 10 times the data set size along with 2.5 times the number of duplicates or 2 times the instances per entity, causes a 4 times increment in the execution time with 0.1 accuracy reduction. This behavior mainly results from using canopies (instead of individual instances) as well as the algorithm for generating best coreferent sets. C-1 and C-2 have 20000 instances. From Fig. 11.e&f, we see that C-2 has higher execution time, and, when more than 5 instances in entities, a lower accuracy. This is caused by the higher number of entities in C-2 than C-1 (Table 2), which forces the algorithm to generate and process more canopies.

7. RELATED WORK

We now discuss recent resolution methods and mechanisms related to extraction tasks and probabilistic models.

Statistical Reasoning within Databases. Following the IE community, recent papers [27] and [4] introduced methodologies that incorporate extraction tasks directly into the relational query processing; thereby achieving interaction between tasks that were previously performed sequentially and in isolation. The suggested approaches [27, 4], perform segmentation through in-database inference during query processing, typically using linear-chain CRFs. In contrast to the separate execution of the segmentation task and query evaluation, their combination improved the overall efficiency of processing, accuracy of results, and enhanced the expressivity of queries (e.g., querying the results with the higher probabilities and not just the highest one). We focus on multiple extraction tasks and executing joint inference over a (possibly large) model that represents these tasks.

Joint Probabilistic Inference. The goals of our work are closely related to recent approaches focusing on joint probabilistic inference. The *DeepDive* project [23], considers the problem of processing unstructured sources for populating structured databases. Random variables and their correlations are modeled using factor graphs. A previous version of the project, i.e., Tuffy [17], focused on variables assigned values using one of their possible labels and a bottom-up approach for the grounding combined with in-memory search and partitioning for scaling execution. The DeepDive model is more similar to our model. It supports boolean variables with each possible world being a particular assignment of the variables and the probability of each world based on the weights of factor functions. Our work goes beyond this. It (i) incorporates two extraction tasks, which covers a larger portion of the IE pipeline; (ii) handles the more complex model that is generated by efficiently considering not only the possible worlds arising from different variable assignments over factor graphs as DeepDive but the worlds over the combination of factor graphs with canopy clustering; and (iii) computes inference over a large class of SQL queries.

Tuffy and DeepDive are based on / related with Markov Logic Networks; a more expressive model. With respect to related work in this area [21], we are able to retrieve the most probable variable assignments and not just the one with the highest probability. Also, our technique incorporates mechanisms for efficiently detecting these variable assignments (note that Tuffy has mechanisms with similar goals). In addition, the followed approach provides advantages by following relational database mechanisms, such as the performance benefits from the algebraic optimizations.

Incremental processing is another contribution of the recent DeepDive publications [23]. It is achieved by techniques incorporated in the grounding and inference phases in order to enable incremental maintenance that will influence both the output data as well as output probabilities. Our current work focuses on the data model and query processing; we plan to study this aspect in the next steps of our work.

Efficiency & Scalability. One methodology is by scaling the inference process. For instance, Elementary [18] employs specialized inference algorithms and [30] sampling but instead of generating the possible worlds from scratch it uses MCMC to hypothesize modifications to worlds and executes the query over the resulting hypothesized worlds. Our approach uses approximate inference based on substructures, which detects the k best variable assignments and then executes inference on the resulting factor graphs. This means that our approach' scalability can be further improved by scaling the inference process and even by adjusting and incorporating existing methodologies (e.g., [30]).

The area of probabilistic databases has also related mechanisms. In particularly, partitioning separates data into a series of groups that are then combined to generate the possible worlds [2, 22]. Our technique works on a more complex model since our groups (i.e., canopies) are not disjoint.

In addition, there are approaches executing queries directly on the representations following different materialization strategies as in MauveDB [7] or even without materialization as in FunctionDB [26]. MauveDB and FunctionDB focus on the particular aspect whereas our technique just builds on the basic idea of this mechanism.

Another option with respect to scalability is incorporating techniques introduced for processing large data sets. Suggestions in the context of resolution include blocking mechanisms, i.e., partitioning the data into independent blocks and comparing only the ones inside the same block [6, 12, 28]. In this work, we build on canopy clustering that partitions the data into overlapping blocks [16]. We basically reduce the overall execution process by generating substructures of our model through canopy clustering. To the best of our knowledge, this is the first time that canopy clustering has been used in such a combination with graphical models (both in IE and DB methodologies).

8. CONCLUSIONS & FUTURE WORK

We introduced holistic query processing over IE pipelines with special operators for retrieving resolution information. We also defined canopy factor graphs and introduced an approximate technique for processing queries as well as optimizations that cut across the integrated tasks. We verified effectiveness and scalability through an extensive evaluation.

This work revealed additional challenges worth investigating. The first is incorporating additional extraction tasks, especially segmentation, which requires extending the model and query processing. An additional challenge is enhancing query expressivity, with a final goal being the generic SPJ.

Acknowledgements. This work was partially supported by the European Commission under ICT-FP7-QualiMaster-619525.

References

[1] Alchemy: Open source AI. Citeseer Dataset. https://alchemy.cs.washington.edu/data/.

[2] L. Antova, C. Koch, and D. Olteanu. $10^{(10^6)}$ worlds and beyond: efficient representation and processing of incomplete information. *VLDB J.*, 18(5):1021–1040, 2009.

[3] M. Bilenko, R. Mooney, W. Cohen, P. Ravikumar, and S. Fienberg. Adaptive name matching in information integration. *IEEE Intelligent Systems*, 18(5):16–23, 2003.

[4] F. Chen, X. Feng, C. Ré, and M. Wang. Optimizing statistical information extraction programs over evolving text. *ICDE*, pages 870–881, 2012.

[5] W. Cohen, P. Ravikumar, and S. Fienberg. A comparison of string distance metrics for name-matching tasks. In *IIWeb*, 2003.

[6] T. de Vries, H. Ke, S. Chawla, and P. Christen. Robust record linkage blocking using suffix arrays. In *CIKM*, pages 305–314, 2009.

[7] A. Deshpande and S. Madden. Mauvedb: supporting model-based user views in database systems. In *SIGMOD*, pages 73–84, 2006.

[8] X. Dong, A. Halevy, and J. Madhavan. Reference reconciliation in complex information spaces. In *SIGMOD*, pages 85–96, 2005.

[9] E. Ioannou and M. Garofalakis. Query analytics over probabilistic databases with unmerged duplicates. *TDKE*, 27(8):2245–2260, 2015.

[10] E. Ioannou, W. Nejdl, C. Niederée, and Y. Velegrakis. On-the-fly entity-aware query processing in the presence of linkage. *PVLDB*, 3(1):429–438, 2010.

[11] E. Ioannou, N. Rassadko, and Y. Velegrakis. On generating benchmark data for entity matching. *JoDS*, 2(1):37–56, 2013.

[12] L. Jin, C. Li, and S. Mehrotra. Efficient record linkage in large data sets. In *DASFAA*, pages 137–146, 2003.

[13] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*, pages 282–289, 2001.

[14] A. McCallum. Information extraction: distilling structured data from unstructured text. *ACM Queue*, 3(9):48–57, 2005.

[15] A. McCallum. Information extraction, data mining and joint inference (tutorial). In *KDD*, page 835, 2006.

[16] A. McCallum, K. Nigam, and L. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *KDD*, pages 169–178, 2000.

[17] F. Niu, C. Ré, A. Doan, and J. W. Shavlik. Tuffy: Scaling up statistical inference in markov logic networks using an RDBMS. *PVLDB*, 4(6):373–384, 2011.

[18] F. Niu, C. Zhang, C. Ré, and J. W. Shavlik. Elementary: Large-scale knowledge-base construction via machine learning and statistical inference. *IJSWIS*, 8(3):42–73, 2012.

[19] G. Papadakis, G. Koutrika, T. Palpanas, and W. Nejdl. Meta-blocking: Taking entity resolutionto the next level. *TKDE*, 26(8):1946–1960, 2014.

[20] Parag and P. Domingos. Multi-relational record linkage. In *MRDM workshop, co-located with KDD*, 2004.

[21] M. Richardson and P. M. Domingos. Markov logic networks. *Machine Learning*, 62(1-2):107–136, 2006.

[22] P. Sen and A. Deshpande. Representing and querying correlated tuples in probabilistic databases. In *ICDE*, pages 596–605, 2007.

[23] J. Shin, S. Wu, F. Wang, C. D. Sa, C. Zhang, and C. Ré. Incremental knowledge base construction using deepdive. *PVLDB*, 8(11):1310–1321, 2015.

[24] S. Singh, K. Schultz, and A. McCallum. Bi-directional joint inference for entity resolution and segmentation using imperatively-defined factor graphs. In *ECML/PKDD (2)*, pages 414–429, 2009.

[25] C. Sutton and A. McCallum. An introduction to conditional random fields. *Foundations and Trends in Machine Learning*, 4(4):267–373, 2012.

[26] A. Thiagarajan and S. Madden. Querying continuous functions in a database system. In *SIGMOD*, pages 791–804, 2008.

[27] D. Wang, M. Franklin, M. Garofalakis, and J. Hellerstein. Querying probabilistic information extraction. *PVLDB*, 3(1):1057–1067, 2010.

[28] S. Whang, D. Menestrina, G. Koutrika, M. Theobald, and H. Garcia-Molina. Entity resolution with iterative blocking. In *SIGMOD*, pages 219–232, 2009.

[29] M. Wick, A. Culotta, K. Rohanimanesh, and A. McCallum. An entity based model for coreference resolution. In *SDM*, pages 365–376, 2009.

[30] M. Wick, A. McCallum, and G. Miklau. Scalable probabilistic databases with factor graphs and MCMC. *PVLDB*, 3(1):794–804, 2010.

[31] M. Wick, K. Rohanimanesh, K. Schultz, and A. McCallum. A unified approach for schema matching, coreference and canonicalization. In *KDD*, pages 722–730, 2008.