Chapter 9
Utilizing Entities for an Enhanced Search Experience

Over the past decade, search engines have not only improved the quality of search results, but also evolved in how they interact with users. Modern search engines provide assistance to users throughout the entire search process, from formulating their information needs to presenting results and recommending additional content. This chapter presents a selection of topics, where entities are utilized with the overall aim of improving users' search experiences.

First, in Sect. 9.1, we discuss techniques for assisting users with articulating their information needs. These include query assistance services, such as query auto-completion and query suggestions, and specialized query building interfaces. Next, in Sect. 9.2, we turn to the question of result presentation. In conventional document retrieval, the standard way of serving results is to display a *snippet* for each document, consisting of its title and a short summary. This summary is automatically extracted from the document with the aim of explaining why that particular document is relevant to the query at hand. Moving from documents to entities as the unit of retrieval, the question we need to ask is: How can one generate dynamic summaries of entities when displaying them as search results? Finally, in Sect. 9.3, we describe entity recommendation methods that present users with contextual suggestions, encourage exploration, and allow for serendipitous discoveries. We study the related entity retrieval problem in different flavors, depending on what kind of input is available to the recommendation engine. Furthermore, we address the question of explaining the relationship in natural language between entities presented to the user. We refer to Table 9.1 for the notation used in this chapter.

9.1 Query Assistance

Chapter 7 dealt with query understanding from the machine's point of view. In this section, we bring the user's perspective to the forefront. How can a semantic search

© The Author(s) 2018

K. Balog, *Entity-Oriented Search*, The Information Retrieval Series 39,
https://doi.org/10.1007/978-3-319-93935-3_9

Table 9.1 Notation used in this chapter

Symbol	Meaning
e	Entity ($e \in \mathcal{E}$)
E	Graph edges
$\mathcal{E}_q$	Set of entities linked in query q ($\mathcal{E}_q \subset \mathcal{E}$)
$\mathcal{E}_s$	Set of entities clicked in session s ($\mathcal{E}_s \subset \mathcal{E}$)
$\mathcal{F}_e$	Knowledge base facts about entity e
q	Query
$\mathcal{Q}$	Query log (set of unique queries)
$\mathcal{Q}_s$	Set of queries issued within search session s
s	Search session ($s \in \mathcal{S}$)
$\mathcal{S}$	Set of search sessions
t	Term ($t \in \mathcal{V}$)
$\mathcal{T}$	Type taxonomy
$\mathcal{T}_e$	Types of entity e
u	Query template ($u \in \mathcal{U}$)
$\mathcal{U}$	Set of templates
$\mathcal{U}_q$	Set of templates for query q
V	Graph vertices
y	Entity type ($t \in \mathcal{T}$)

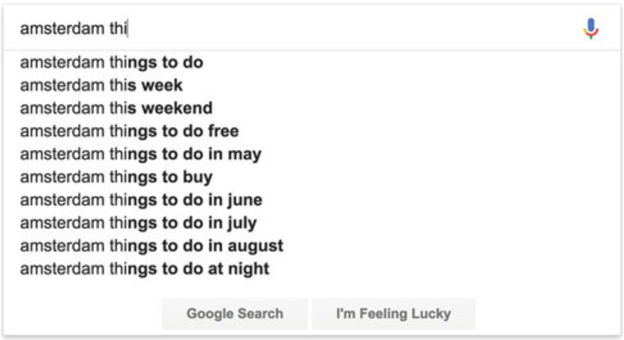
system assist the user in the process of articulating and expressing her information need? First, we discuss automatic methods that provide users with query suggestions while they are typing their query (Sect. 9.1.1) or after the query has been submitted (Sect. 9.1.2). Then, we present examples of specialized query building interfaces that enable users to formulate semantically enriched (keyword++) queries, by explicitly marking entities, types, or relationships (Sect. 9.1.3).

9.1.1 Query Auto-completion

Query auto-completion (QAC) provides users with query suggestions as they enter terms in the search box. Query auto completion is a common feature in modern search engines; see Fig. 9.1 for an illustration. It helps users to express their search intent as well as to avoid possible spelling mistakes [15].

Most systems rely on the wisdom of the crowds, i.e., suggest completions (matching the entered prefix) that have been most popular among users in the past, based on query logs [5]. Typically, QAC is viewed as a ranking problem, where the aim is “to return the user’s intended query at the top position of a list of [candidate] query completions” [15].

Formally, we let q_0 be the incomplete query that the user has typed in so far and q_s be a suggested candidate query suffix. Let $c(q_0 \oplus q_s)$ be the number of times


amsterdam thi

amsterdam things to do
amsterdam this week
amsterdam this weekend
amsterdam things to do free
amsterdam things to do in may
amsterdam things to buy
amsterdam things to do in june
amsterdam things to do in july
amsterdam things to do in august
amsterdam things to do at night

Fig. 9.1 Query auto-completion in the Google search engine

we observe the query $q_0 \oplus q_s$ issued in the query log $\mathcal{Q}$, where $\oplus$ is the string concatenation operator. The baseline approach to QAC, referred to as *most popular completion* in [5], is to rank suggestions according to:

$$score(q_s; q_0) = P(q_s|q_0) = \frac{c(q_0 \oplus q_s)}{\sum_{q_0 \oplus q_{s'} \in \mathcal{Q}} c(q_0 \oplus q_{s'})} .$$

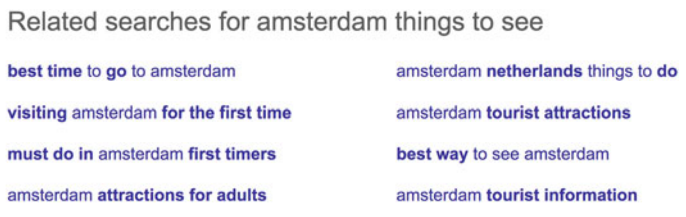
Given that many information needs revolve around entities, entities can be leveraged to improve query auto-completion.

9.1.1.1 Leveraging Entity Types

Meij et al. [32] focus on a specific subset of queries that can be decomposed into entity and refiner components, such as "ASPIRIN *side effects*" or "BRITNEY SPEARS *video*" (refiners typeset in italics). They show that exploiting the type of entities being sought can improve QAC for rare queries. Specifically, we let $q_0 = e$, where the *query entity* e is recognized using entity linking techniques (cf. Sect. 7.3). Further, let $\mathcal{T}_e$ denote the entity types assigned to e in the knowledge base. Their best performing model (called M1 in [32]) looks at the most likely completion for a given entity type $y \in \mathcal{T}_e$:

$$score(q_s; q_0) = P(q_s|y) = \frac{c(q_s, y)}{\sum_{q_0 \oplus q_{s'} \in \mathcal{Q}} c(q_{s'}, y)} ,$$

where $c(q_s, y)$ is the number of times we can observe completion q_s with an entity of type y in the query log. Note that entities commonly have multiple types assigned to them. Selecting a single "best" type y out of the types of the query entity is an



Related searches for amsterdam things to see

best time to **go** to amsterdam	amsterdam **netherlands** things to **do**
visiting amsterdam **for the first time**	amsterdam **tourist attractions**
must do in amsterdam **first timers**	**best way** to see amsterdam
amsterdam **attractions for adults**	amsterdam **tourist information**

Fig. 9.2 Query suggestions offered by the Bing search engine for the query "*amsterdam things to see*"

open issue that is not dealt with in [32]. Instead, they evaluate performance for all possible types and then choose the type that led to the best performance on the training set.

9.1.2 *Query Recommendations*

Unlike "as-you-type" query auto-completion, which assists users during the articulation of their information need, *query recommendations* (a.k.a. *query suggestions*) are presented on the SERP once an initial query has been issued. The idea is to help users formulate more effective queries by providing suggestions for the next query. These suggestions are semantically related to the submitted query and either dive deeper into the current search direction or move to a different aspect of the search task [37]. Query suggestions are an integral feature of major web search engines and an active area of research [10, 11, 18, 25, 37, 43, 46]. Figure 9.2 shows query recommendations in action in a major web search engine.

Generating query recommendations is commonly viewed as a ranking problem, where given an input query q, the task is to assign a score to each candidate suggestion q', $score(q';q)$. Like QAC, query recommendation also relies on the wisdom of the crowds by exploiting query co-occurrences and/or click-through information mined from search logs. While log-based methods work well for popular queries, it is a challenge for them to ensure *coverage* (i.e., provide meaningful suggestions) for rare queries. Most query assistance services perform poorly, or are not even triggered, on long-tail queries, simply because there is little to no historical data available for them. In this section, we will discuss methods that alleviate this problem by utilizing entities in a knowledge base.

We start by presenting the query-flow graph (QFG) method [10] in Sect. 9.1.2.1, which is a seminal approach for generating query recommendations. Then, in Sects. 9.1.2.2–9.1.2.4, we introduce various extensions to the QFG approach that tap into specific characteristics of entities, such as types and relationships. All these methods rely on the ability to recognize entities mentioned in queries. We refer back to Sect. 7.3 for the discussion of techniques for entity linking in queries. The set $\mathcal{E}_q$ denotes the entities identified in query q.

9.1.2.1 Query-Flow Graph

The query-flow graph (QFG), proposed by Boldi et al. [10], is a compact representation of information contained in a query log. It is a directed graph $G = (V, E, W)$, where the set V of vertices is the distinct set $\mathcal{Q}$ of queries contained in a query log, plus two special vertices, v_s and v_t, representing the start and terminal states of sessions: $V = \mathcal{Q} \cup \{v_s, v_t\}$. A *session* is defined as a "sequence of queries of one particular user within a specific time limit" [10]. Commonly, the session time limit is taken to be 30 min. Further, $E \subseteq V \times V$ is a set of directed edges and $W : E \rightarrow (0, 1]$ is a weighting function assigning a weight $w(q_i, q_j)$ to each edge $(q_i, q_j) \in E$. Two queries, q_i and q_j, are connected with a directed edge $(q_i \rightarrow q_j)$, if there is at least one session in the log in which q_i and q_j are consecutive.

The key aspect of the construction of the query-flow graph is how the weighing function connecting two queries, $w(q_i, q_j)$, is defined. A simple and effective solution is to base it on relative frequencies of the query pair appearing in the query log. Specifically, the weight of the edge connecting two queries is computed as:

$$w(q_i, q_j) = \begin{cases} \frac{1}{Z} c(q_i, q_j)\,, & (c(q_i, q_j) > \tau) \lor (q_i = v_s) \lor (q_j = v_t) \\ 0\,, & \text{otherwise}\,, \end{cases}$$

where $c(q_i, q_j)$ is the number of times query q_j follows immediately q_i in a session. The normalization coefficient Z is set to such that the sum of outgoing edge weights equals to 1 for each vertex, i.e., $\sum_j w(q_i, q_j) = 1$. Thus, weight $w(q_i, q_j)$ may be seen as the probability that q_i is followed by q_j in the same search session. This normalization may be viewed as the transition matrix of a Markov chain. Figure 9.3 shows an excerpt from a query-flow graph.

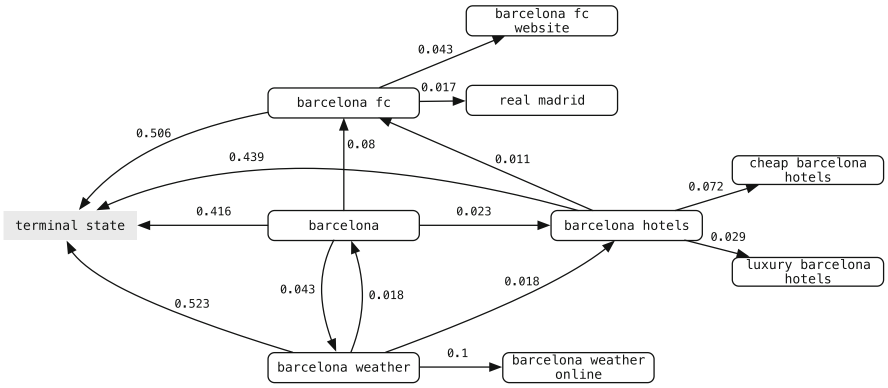


Fig. 9.3 Example of a query-flow graph, based on [10]. Note that not all outgoing edges are reported (thus, the sum of outgoing edges from each vertex does not reach 1). The terminal state vertex (v_t) is distinguished using a gray background

Based on this graph-based representation, query recommendations can be obtained by performing proximity-based top-k vertex retrieval, either neighborhood-based or path-based. A simple recommendation scheme is to pick, for an input query q, the top-k vertices connected with the largest edge weights. However, as observed by Boldi et al. [10], this method tends to drift off toward popular but unrelated queries. A better recommendation would be to pick the most important query q' relative to the initial query q. The recommendation algorithm proposed in [10] is a random walk with restart to a single vertex: a random surfer starts at the initial query q, and at each step either (1) follows one of the outlinks from the current vertex with probability α or (2) jumps back to q with probability $1-\alpha$. This process may also be viewed as applying "a form of personalized PageRank, where the preference vector is concentrated in a single node [vertex]" [10]. More formally, the process is described as computing the transition matrix $\mathbf{A}$ of a Markov chain:

$$\mathbf{A} = \alpha\mathbf{P} + (1-\alpha)\mathbf{1}\mathbf{i}_q^{\mathsf{T}} ,$$

where $\mathbf{P}$ is the row-normalized weight matrix of the query-flow graph, $\mathbf{1}$ is the identity matrix, and $\mathbf{i}_j$ is a "one-hot" vector whose entries are all zeroes, except for the jth vector whose value is 1. The parameter α is chosen to be 0.85 in [10]. $\mathbf{A}$ has a unique stationary distribution vector $\mathbf{v}$, such that $\mathbf{v}^{\mathsf{T}}\mathbf{A} = \mathbf{v}$. This distribution, called the *random-walk score relative to* q, can be computed using the power iteration method. Then, the highest scoring queries can be returned as the most relevant query suggestions for q. Notably, if the top-scoring query is the termination vertex v_t, then it means that the query chain is most likely to end at that point. In that case, it may be wise not to offer any query suggestions to the user.

Instead of using the raw random walk scores, Boldi et al. [10] propose to use a weighting scheme, so as to avoid returning very common queries as suggestions. The variant that yields the best recommendations in their experiments is given by the following formula:

$$score(q';q) = \frac{rw_q(q')}{\sqrt{rw(q')}} , \tag{9.1}$$

where $rw_q(q')$ is the random walk score of the query q' with respect to q (personalized PageRank [26]) and $rw(q')$ is the random walk score of q' computed using a uniform preference vector (no personalization, i.e., starting at random at any vertex).

Having introduced the QFG approach, we shall next look at a number of extensions that utilize entities in a knowledge base, in order to improve recommendations for long-tail queries.

9.1.2.2 Exploiting Entity Aspects

Reinanda et al. [39] define *entity aspects* as a set of refiners that represent the same search intent in the context of an entity. For example, for the entity BARCELONA F.C., the refiners "live," "live streaming," and "live stream" represent the same search intent, i.e., they amount to one particular aspect of that entity. For each entity e, Reinanda et al. [39] mine a set of aspects $\mathcal{A}_e = \{a_1, \ldots, a_m\}$ from a search log. First, the set of queries mentioning that entity, $\mathcal{Q}_e \subset \mathcal{Q}$, is identified using entity linking. Then, refiners r (referred to as *query contexts* in [39]) are extracted by removing the mention of the entity from the queries. Next, refiners that express the same intent are clustered together. Clustering relies on the pairwise similarity between two refiners, $sim(r_i, r_j)$, which is taken to be the maximum of the following three types of similarity:

- *Lexical similarity*, estimated by the Jaro-Winkler distance between r_i and r_j.
- *Semantic similarity*, using the cosine similarity of word embeddings of the refiners, $\cos(\mathbf{r}_i, \mathbf{r}_j)$. Specifically, the vector representation of a refiner, $\mathbf{r}_i$, is computed as the sum of Word2vec [33] vector of each term within.
- *Click similarity*, obtained using the cosine similarity between the vectors of clicked sites.

For clustering refiners, *hierarchical agglomerative clustering* with complete linkage is employed. Refiners are placed in the same cluster if their similarity is above a given threshold. By the end of this step, each cluster of refiners corresponds to an entity aspect.

The mined aspects are used for query recommendations as follows. During the construction of the query-flow graph, a distinction is made between queries that contain a mention of an entity (i.e., are *entity-bearing*) and those that are not. For an entity-bearing query, the mentioned entity e and the refiner r are extracted.[1] Then, r is matched against the appropriate entity aspect, by finding the aspect $a_i \in \mathcal{A}_e$ that contains r as its cluster member, $r \in a_i$. This way, semantically equivalent queries are collapsed into a single entity aspect vertex in the modified query-flow graph. Non-entity-bearing queries are handled as in the regular query-flow graph, i.e., each unique query corresponds to a vertex.

For an incoming new query, the process of generating recommendations works as follows. First, entity linking is performed on the query to decide if it is entity-bearing. Then, the query is matched against the appropriate graph vertex (using the same procedure that was used during the construction of the query-flow graph). Finally, recommendations are retrieved from the query-flow graph.[2] Note that entity aspect vertices may contain multiple semantically equivalent queries. In that case, a single one of these is to be selected (for each vertex), e.g., based on query popularity. This approach (referred to as QFG+A) is shown to achieve small but consistent and significant improvements over generic QFG query recommendations.

[1]In this work, a single "main" entity is selected from each query (due to the fact that the average query length is short, most queries mention just one entity or none).

[2]Reinanda et al. [39] use the simple recommendation scheme, based on raw edge weights. However, it is also possible to apply random walks using the weighting scheme proposed in [10], as is given in Eq. (9.1).

9.1.2.3 Entity Types

An obvious limitation of query-flow graphs is that they cannot make recommendations for long-tail or previously unseen queries. Szpektor et al. [46] alleviates this limitation by enhancing query-flow graphs with *query templates*.[3] Templates are defined by replacing the entity mention in the query by a placeholder, which is an entity type. For example, the queries "*New York hotels*," "*Los Angeles hotels*," and "*Paris hotels*" may be abstracted into a "$\langle$*city*$\rangle$ *hotels*" query template. Then, general recommendation rules, like "$\langle$*city*$\rangle$ *hotels*" $\rightarrow$ "$\langle$*city*$\rangle$ *restaurants*," may be extracted from query logs. Using such rules, it is possible to generate the recommendation "*Yancheng restaurants*" for the input query "*Yancheng hotels*," even if none of those queries have been observed before. Next, we detail the elements of this approach.

Generating Query Templates Each query q in the query log $\mathcal{Q}$ is considered for template construction. Given a query, every word n-gram (up to length 3 in [46]) is checked whether it refers to an entity. If yes, then that query segment of the query is replaced with the type(s) of the corresponding entity to produce the corresponding template(s). Here, entity types are not taken to be a flat set but are considered to exist in a hierarchical type taxonomy. We let $\tilde{\mathcal{T}}_e$ denote the most specific type(s) that entity e is assigned to in the knowledge base. For example, using the DBpedia Ontology as the type taxonomy, the entity ALBERT EINSTEIN has a single most-specific type $\tilde{\mathcal{T}}_e = \{Scientist\}$. By definition, the distance between an entity and (one of) its most specific type(s) is set to 1. Since the type taxonomy is a subsumption hierarchy (cf. Sect. 2.3.1), if entity e is an instance of type y, then it will also be an instance of all supertypes of y. We let $\hat{\mathcal{T}}_e$ denote the set of all supertypes of the types in $\tilde{\mathcal{T}}_e$. For the example entity ALBERT EINSTEIN, $\hat{\mathcal{T}}_e = \{Person, Agent\}$ (since *Scientist* is a subtype of *Person*, which is a subtype of *Agent*). Thus, the type assignments of an entity are partitioned into most specific types and their supertypes (i.e., $\mathcal{T}_e = \tilde{\mathcal{T}}_e \cup \hat{\mathcal{T}}_e$ and $\tilde{\mathcal{T}}_e \cap \hat{\mathcal{T}}_e = \emptyset$). Further, $d(y, y')$ is defined as the shortest path distance between types y and y' in the type taxonomy. The distance function $d(e, y)$ between entity e and type y is then defined as follows:

$$d(e,y) = \begin{cases} 1, & y \in \tilde{\mathcal{T}}_e \\ 1 + \min_{y' \in \tilde{\mathcal{T}}_e} d(y, y')\,, & y \in \hat{\mathcal{T}}_e \\ \infty, & \text{otherwise}\,. \end{cases}$$

For example, d(ALBERT EINSTEIN,*Scientist*) = 1, d(ALBERT EINSTEIN, *Person*) = 2, and d(ALBERT EINSTEIN,*Agent*) = 3.

[3]The notion of query templates is similar to that in [1] (cf. Sect. 7.4), with two main differences: (1) Templates here are defined over a taxonomy of entity types as opposed to attributes and (2) they are defined globally, i.e., are not restricted to any particular domain/vertical (such as travel or weather).

The set of templates constructed from a given query q is denoted as $\mathcal{U}_q$. Each template $u \in \mathcal{U}_q$ is associated with a confidence score $w(u,q)$, which expresses how well u generalizes q. Intuitively, the higher a type is located in the type hierarchy, the higher the risk of the corresponding template over-generalizing the query. Thus, the confidence score can be set to be exponentially decaying with the distance between entity e and type y:

$$w(q,u) = \alpha^{d(e,y)} , \tag{9.2}$$

where α is the decay rate (set to 0.9 in [46]). Note that template scoring in Eq. (9.2) does not take into account the uncertainty associated with the type-annotation of the query. The intuition is that by considering the transitions between templates (based on subsequent queries from which they were generated) in a sufficiently large query log, noise will be eliminated and meaningful transition patterns will surface. This will be explained next.

Extending the Query-Flow Graph In the extended query-flow graph, referred to as *query-template flow graph*, vertices represent not only queries but templates as well. We let the set $\mathcal{U}$ denote all templates that are generated by queries in the search log: $\mathcal{U} = \bigcup_{q \in \mathcal{Q}} \mathcal{U}_q$. In addition to query-to-query transition edges (E_{qq}) of the original query-flow graph, we now have two additional types of edges: (1) query-to-template edges (E_{qu}) and (2) template-to-template edges (E_{uu}).

- There is a directed edge between query q and template u iff $u \in \mathcal{U}_q$. The corresponding edge weight $w(q,u)$ is set proportional to the query-template confidence score, which is given in Eq. (9.2).
- There is a directed edge between templates u_i and u_j iff (i) they have the same placeholder type and (ii) there is at least one *supporting edge* $(q_i,q_j) \in E_{qq}$ such that $u_i \in \mathcal{U}_{q_i}$, $u_j \in \mathcal{U}_{q_j}$, and the substituted query segment is the same in q_i and q_j. The set of all support edges is denoted as $E_s(u_i,u_j)$. Then, the edge weight between u_i and u_j is set proportional to the sum of edge weights of all supporting query pairs:

$$w(u_i,u_j) \propto \sum_{(q_i,q_j) \in E_s(u_i,u_j)} w(q_i,q_j) .$$

  For example, for the template-to-template edge "$\langle$*city*$\rangle$ *hotels*" → "$\langle$*city*$\rangle$ *restaurants*," the set of support edges includes {"*Paris hotels*" → "*Paris restaurants*," *New York hotels*" → "*New York restaurants*," . . . }.

Normalization is performed in both cases to ensure that the outgoing edge weights of graph vertices sum up to 1.

Generating Query Recommendations Using the regular query-flow graph, candidate query recommendations q' for an input query q would be those for which there exists a directed edge $(q,q') \in E_{qq}$. With the extended query-template flow graph, candidate recommendations also include queries that have not been observed in the logs before but can be instantiated via a template. Specifically, there needs to be a mapping edge $(q,u) \in E_{qu}$ and a template-to-template transition edge $(u,u') \in E_{uu}$, such that by instantiating u' with the entity extracted from q, it yields q' as the result. Then, candidate recommendations are scored according to the following formula:

$$score(q';q) = w(q,q') + \sum_{\substack{u \in \mathcal{U}_q \\ (u,u') \in E_{uu} \\ ins(u',q,u)=q'}} w(q,u)\, w(u,u')\ ,$$

where $ins(u',q,u)$ denotes the query that is the result of instantiating template u' based on query q and template u. Given that edge weights are normalized, the resulting score will be in the range $[0,1]$ and "can be interpreted as the probability of going from q to q' by one of the feasible paths in the query-template flow graph" [46].

9.1.2.4 Entity Relationships

Bordino et al. [12] extend the query-flow graph with entity relationship information, and Huang et al. [25] capitalize on this idea for generating query recommendations. Suppose that two queries q_i and q_j appear in the same session and they mention entities e_i and e_j, respectively. Then, in addition to the flow from q_i to q_j in the query-flow graph, we can also utilize the relationships between e_i and e_j in the knowledge base. More formally, the enhanced graph, referred to as *EQGraph* in [12], has query vertices $V_{\mathcal{Q}}$ and entity vertices $V_{\mathcal{E}}$, with query-to-query edges E_{qq}, entity-to-query edges E_{eq}, and entity-to-entity edges E_{ee}. See Fig. 9.4 for an illustration.

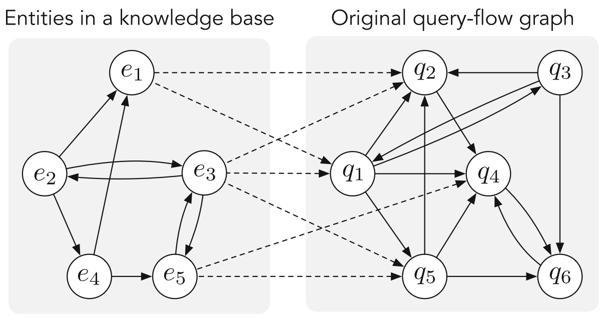


Fig. 9.4 Entity-query graph (EQGraph), extending the regular query-flow graph with entities from a knowledge base

Entity-to-Query Edges Each entity is connected to all the queries that mention it. The edge between entity e and query q is given a weight proportional to the relative frequency of that query in the log:

$$w(e,q) = \frac{c(q)}{\sum_{q':e\in\mathcal{E}_{q'}} c(q')} ,$$

where $c(q)$ is the number of times q appears in the query log, and $\mathcal{E}_{q'}$ denotes the set of entities that are extracted from q' via entity linking. Note that the outgoing edge weights sum up to one for each entity vertex.

Entity-to-Entity Edges Edge weights $w(e,e')$ represent the transition probability between a pair of entities e and e'. Bordino et al. [12] derive these weights based on the query log, by considering all query-to-query transitions $q \to q'$, where q mentions e and q' mentions e':

$$w(e,e') = 1 - \prod_{\substack{(q,q')\in E_{qq} \\ (e,q),(e',q')\in E_{eq}}} \Big(1 - \frac{w(q,q')}{|\mathcal{E}_q| \times |\mathcal{E}_{q'}|}\Big) .$$

This formulation distributes the probability mass uniformly among the possible $(|\mathcal{E}_q| \times |\mathcal{E}_{q'}|)$ entity-to-entity transitions derived from $q \to q'$.

Generating Query Recommendations Huang et al. [25] generate query recommendations by computing personalized PageRank [26] on the EQGraph, starting from entities. The core of their approach lies in the idea that instead of considering the entities that are mentioned in the query ($\mathcal{E}_q$), they consider related entities from the knowledge graph. This set of related entities, denoted as $\mathcal{E}_R$, is derived based on the notion of *meta-paths* [45]. A meta-path M in the knowledge graph is a sequence of entity types $y_1,\dots,y_n$ connected by predicates (i.e., relationship types) $p_1,\dots,p_{n-1}$, such that $M = y_1 \xrightarrow{p_1} y_2 \dots y_{n-1} \xrightarrow{p_{n-1}} y_n$. Each of these meta-paths represents a specific direct or composite ("multi-hop") relationship between two entities. Two entities may be connected by multiple meta-paths; a natural approach, followed in [25], is to select the shortest meta-path between them to represent their relationship. Let $\mathcal{M}$ be the set of meta-paths over the entity types in the KG, and $\mathcal{M}_y \subset \mathcal{M}$ be the set of outgoing meta-paths for type y. Related entities are collected by performing a path-constrained random walk [29] on knowledge graph predicates, with each meta-path $M \in \mathcal{M}_y$, for each of the types associated with the linked entities in the query ($y \in \mathcal{T}_e, e \in \mathcal{E}_q$). Each of these related entities $e \in \mathcal{E}_R$ accumulates weight, $w(e)$, based on the various meta-paths it can be reached on. See Algorithm 9.1 for the detailed procedure.

The most relevant queries, with respect to the related entities $\mathcal{E}_R$, are returned as recommendations. Specifically, for each of the related entities, $e \in \mathcal{E}_R$, personalized PageRank is performed on the EQGraph, starting from e with initial probability

Algorithm 9.1: Related entity finding for query recommendation [25]

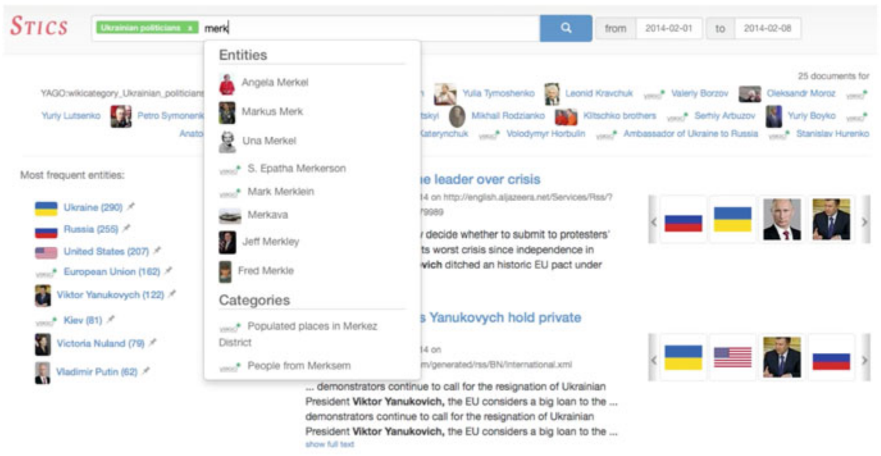
```
Input: query-flow graph, knowledge graph, query $q$
Output: related entities $\mathcal{E}_R$ with weights $w$
1  $\mathcal{E}_q \leftarrow$ perform entity linking on $q$
2  $\mathcal{E}_R \leftarrow \emptyset$
3  foreach $e \in \mathcal{E}_q$ do
4      foreach $y \in \mathcal{T}_e$ do
5          foreach $M \in \mathcal{M}_y$ do
6              $\mathcal{E}' \leftarrow pathConstrainedRandomWalk(e, M)$
7              foreach $e' \in \mathcal{E}'$ do
8                  $\mathcal{E}_R \leftarrow \mathcal{E}_R \cup \{e'\}$
9                  $w(e') \leftarrow w(e') + \frac{1}{|\mathcal{E}_q| \times |\mathcal{E}'|}$
10             end
11         end
12     end
13 end
14 return $\mathcal{E}_R$, $w$
```

$w(e)$. The resulting probability distributions are aggregated, and the top-k queries with the largest aggregated score are offered as recommendations.

Note that this recommendation method considers only the entities mentioned in the query but not the other contextual terms in the query. It means that if two queries q and q' mention the same entities ($\mathcal{E}_q = \mathcal{E}_{q'}$) then generated recommendations will be exactly the same for the two.

9.1.3 Query Building Interfaces

In Chap. 4, we have seen that leveraging semantically enriched queries, referred to as *keyword++ queries*, yields improved retrieval performance. Such keyword++ queries may contain annotations of specific entities, target types, or relationships. One way to obtain those annotations is via automated techniques aimed at query understanding—which we have discussed in Chap. 7. Alternatively, it may be delegated to the user to provide semantic annotations for queries, and thereby more explicitly express the underlying information need. This, however, can be challenging for ordinary users, due to their unfamiliarity with the underlying knowledge base. Furthermore, even those that are acquainted with the knowledge base will find it problematic to navigate the large space of entities, types, and relationships without some support. In order to aid users in the process of formulating complex queries, specialized query building interfaces have been proposed [6, 24, 42]. A common feature of these systems is that they provide context-sensitive suggestions. The STICS system offers suggestions for entities and categories, as users type query terms; a screenshot is shown in Fig. 9.5. Schmidt et al. [42] present a corpus-


STICS
Ukrainian politicians x merk
from 2014-02-01 to 2014-02-08

Entities
Angela Merkel
Markus Merk
Una Merkel
S. Epatha Merkerson
Mark Merklein
Merkava
Jeff Merkley
Fred Merkle

Categories
Populated places in Merkez District
People from Merksem

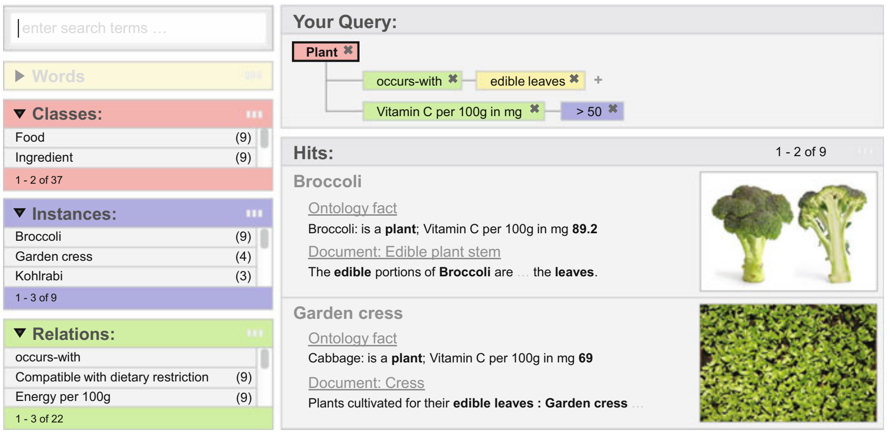
YAGO:wikicategory_Ukrainian_politicians
Yuriy Lutsenko Petro Symonenk Anato
25 documents for
Yulia Tymoshenko Leonid Kravchuk Valeriy Borzov Oleksandr Moroz
tskyi Mikhail Rodzianko Klitschko brothers Serhiy Arbuzov Yuriy Boyko
Katerynchuk Volodymyr Horbulin Ambassador of Ukraine to Russia Stanislav Hurenko

Most frequent entities:
Ukraine (290)
Russia (255)
United States (207)
European Union (162)
Viktor Yanukovych (122)
Kiev (81)
Victoria Nuland (79)
Vladimir Putin (62)

e leader over crisis
14 on http://english.aljazeera.net/Services/Rss/?
79989
decide whether to submit to protesters'
ts worst crisis since independence in
vich ditched an historic EU pact under

s Yanukovych hold private
14 on
m/generated/rss/BN/international.xml
... demonstrators continue to call for the resignation of Ukrainian President **Viktor Yanukovich,** the EU considers a big loan to the ... demonstrators continue to call for the resignation of Ukrainian President **Viktor Yanukovich,** the EU considers a big loan to the ...
show full text

Fig. 9.5 Screenshot of the STICS system [24], http://stics.mpi-inf.mpg.de/


enter search terms …

Words

Classes:

Food	(9)
Ingredient	(9)
1 - 2 of 37	

Instances:

Broccoli	(9)
Garden cress	(4)
Kohlrabi	(3)
1 - 3 of 9	

Relations:

occurs-with	
Compatible with dietary restriction	(9)
Energy per 100g	(9)
1 - 3 of 22	

Your Query:
Plant ✖
occurs-with ✖ — edible leaves ✖ +
Vitamin C per 100g in mg ✖ — > 50 ✖

Hits: 1 - 2 of 9

Broccoli
Ontology fact
Broccoli: is a **plant**; Vitamin C per 100g in mg **89.2**
Document: Edible plant stem
The **edible** portions of **Broccoli** are … the **leaves**.

Garden cress
Ontology fact
Cabbage: is a **plant**; Vitamin C per 100g in mg **69**
Document: Cress
Plants cultivated for their **edible leaves** : **Garden cress** …

Fig. 9.6 Screenshot of the Broccoli system [6], http://broccoli.informatik.uni-freiburg.de/

adaptive extension, where the ranking of candidate suggestions also takes into account the underlying document collection. That is, they only suggest entities and categories "that lead to non-empty results for the document collection being searched" [42]. They further introduce a data structure for storing pre-computed relatedness scores for all co-occurring entities, in order to keep response times below 100 ms. Another example is the Broccoli system, which targets expert users and allows them to incrementally construct tree-like SPARQL queries, using similar techniques (i.e., corpus-based statistics) for generating suggestions [6]. Figure 9.6 presents a screenshot of the system.

9.2 Entity Cards

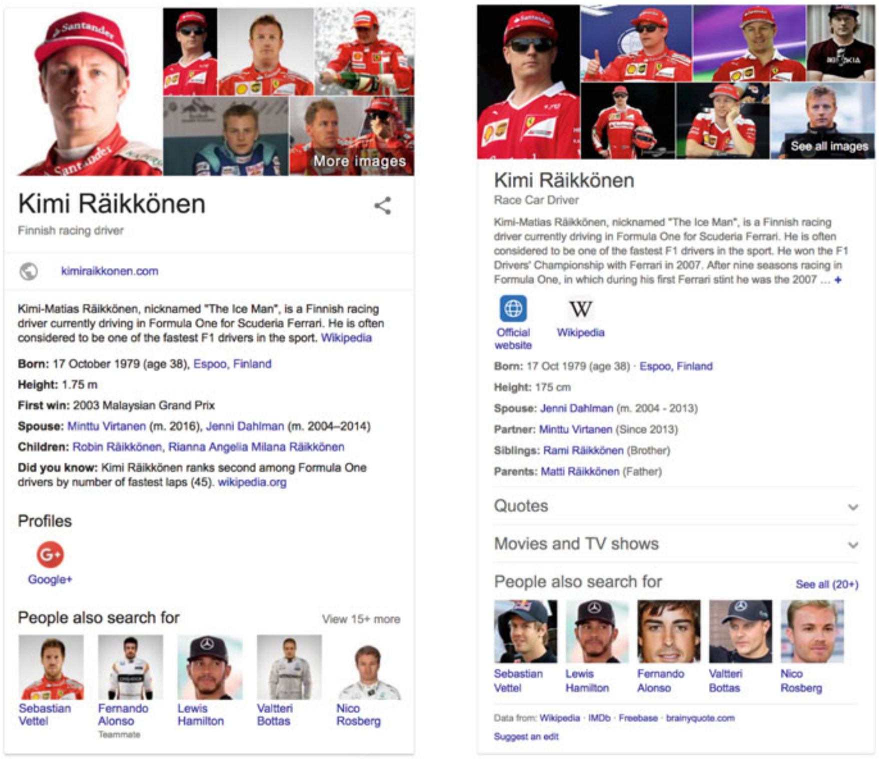
In recent years, there has been an increasing trend of surfacing structured results in web search in the form of various *knowledge panels*. Being served with rich and focused responses, users no longer need to engage with the organic (document-oriented) search results to find what they were looking for. This marks a paradigm shift in search engines evolving into *answer engines*. One group of knowledge panels, often referred to as *direct displays*, provide instant answers to a range of popular information needs, e.g., weather, flight information, definitions, or how-to questions. Some direct displays invite users to engage and interact with them (like currency conversion or finance answers), while others yield a clear inline answer (such as dictionary or reference answers) with no further interaction expected. Our focus in this section will be on another type of knowledge panel, called *entity card*, which summarizes information about a given entity of interest. Unlike direct displays, whose mere goal is to provide a succinct answer, entity cards intend to serve an additional purpose—to present the user with an overview of a particular entity for contextualization and further exploration.

An entity card portrays a summary of a selected entity, commonly including the entity's name, type, short description, a selection of key attributes and relationships, and links to other types of relevant content.

Entity cards are an integral component of modern search engine result pages, on both desktop and mobile devices [14, 28]. Triggered by an entity-bearing query, a rich informational panel is displayed (typically on the top-right of the SERP on a desktop device), offering a summary of the query entity, as shown in Fig. 9.7a, b.

It has been long known that providing query-biased snippets for documents in the result list positively influences the user experience [49]. Entity cards may be viewed as the counterpart of document snippets for entities, and, as we shall show in this section, may be generated in a query-dependent fashion. It has been shown that entity cards attract users' attention, enhance their search experience, and increase their engagement with organic search results [14, 36]. Furthermore, when cards are relevant to their search task, users issue significantly fewer queries [14] and can accomplish their tasks faster [36].

We shall begin with an overview of what is contained in an entity card. Then, we will focus on the problem of selecting a few properties from an underlying knowledge base that best describe the selected entity, with respect to a given query, which will serve as the *factual summary* of that entity.


More images

Kimi Räikkönen

Finnish racing driver

kimiraikkonen.com

Kimi-Matias Räikkönen, nicknamed "The Ice Man", is a Finnish racing driver currently driving in Formula One for Scuderia Ferrari. He is often considered to be one of the fastest F1 drivers in the sport. Wikipedia

Born: 17 October 1979 (age 38), Espoo, Finland

Height: 1.75 m

First win: 2003 Malaysian Grand Prix

Spouse: Minttu Virtanen (m. 2016), Jenni Dahlman (m. 2004–2014)

Children: Robin Räikkönen, Rianna Angelia Milana Räikkönen

Did you know: Kimi Räikkönen ranks second among Formula One drivers by number of fastest laps (45). wikipedia.org

Profiles

Google+

People also search for — View 15+ more

Sebastian Vettel · Fernando Alonso (Teammate) · Lewis Hamilton · Valtteri Bottas · Nico Rosberg

a

See all images

Kimi Räikkönen

Race Car Driver

Kimi-Matias Räikkönen, nicknamed "The Ice Man", is a Finnish racing driver currently driving in Formula One for Scuderia Ferrari. He is often considered to be one of the fastest F1 drivers in the sport. He won the F1 Drivers' Championship with Ferrari in 2007. After nine seasons racing in Formula One, in which during his first Ferrari stint he was the 2007 ... +

Official website · Wikipedia

Born: 17 Oct 1979 (age 38) · Espoo, Finland

Height: 175 cm

Spouse: Jenni Dahlman (m. 2004 - 2013)

Partner: Minttu Virtanen (Since 2013)

Siblings: Rami Räikkönen (Brother)

Parents: Matti Räikkönen (Father)

Quotes

Movies and TV shows

People also search for — See all (20+)

Sebastian Vettel · Lewis Hamilton · Fernando Alonso · Valtteri Bottas · Nico Rosberg

Data from: Wikipedia · IMDb · Freebase · brainyquote.com

Suggest an edit

b

Fig. 9.7 (**a**) Entity card in Google. (**b**) Entity card in Bing

9.2.1 The Anatomy of an Entity Card

Entity cards are complex information objects that are dynamically generated in response to an entity-oriented query, after determining the intended entity for that query (cf. Sect. 7.3). Figure 9.8 shows a card layout that is commonly used in contemporary web search engines, comprising (1) images, (2) the name and type of the entity, (3) a short textual description, (4) entity facts, and (5) related entities. Additionally, depending on the type of the entity and the intent of the search query, other components may also be included, such as maps, quotes, tables, or forms.

Most elements of entity cards have their own set of associated research challenges. An entity may be associated with multiple types in the KB. For example, the types of ARNOLD SCHWARZENEGGER in Freebase include, among others, `tv.tv_actor`, `sports.pro_athlete`, and `government.politician`. The problem of selecting a single "main" type to be displayed on the card has been addressed using both context-independent [7] and context-dependent [50] methods. For emerging entities, that already have some facts stored about them in the KB, but

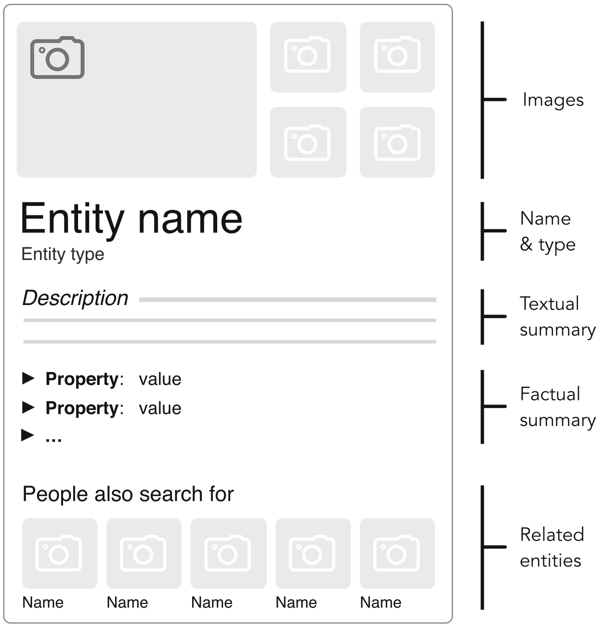


Fig. 9.8 Common entity card layout

lack a Wikipedia-style summary, natural language descriptions may be produced automatically [17, 41]. The factual summary, a truncated view of facts about the entity, is a central element of entity cards. We devote the remainder of this section to this very problem. Finally, related entity suggestions typically utilize search logs and entity co-occurrence information; we shall discuss specific methods in Sect. 9.3. Clicking on one of the related entities typically launches a new query with the related entity.

9.2.2 Factual Entity Summaries

The problem of generating informative entity summaries from RDF data has generated considerable interest over the recent years [16, 22, 23, 47, 48]. Since descriptions of entities in a knowledge base typically include hundreds of factual statements, "for human users it becomes difficult to comprehend the data unless a selection of the most relevant facts is provided" [47]. Below, we present the approach by Hasibi et al. [23] that is shown to be more effective than other relevance-oriented fact ranking methods, which employ variations of the PageRank algorithm [16, 47, 48]. Notably, it considers facts with both entity and literal object values.

Let $\mathcal{F}_e$ denote all the facts stored about a given entity e in the knowledge base $\mathcal{K}$. That is, $\mathcal{F}_e$ consists of all predicate-object pairs out of those SPO triples, where e appears as subject:

$$\mathcal{F}_e = \{(p,o) : (s,p,o) \in \mathcal{K}, s = e\} .$$

For notational convenience, we shall use the shorthand f to denote a single fact, which is essentially a property-value pair; further, we shall write p_f and o_f to denote the predicate and object elements of the fact, respectively. Since $\mathcal{F}_e$ is typically large (on the order of hundreds), the challenge is to select a small subset of them, to be displayed in the summary section of the entity card, that are deemed to have the highest utility for the user. Hasibi et al. [23] argue that factual entity summaries serve a dual purpose: "they offer a synopsis of the entity and, at the same time, can directly address users' information needs." Therefore, when selecting which facts to include in the summary, one should consider both their general importance, irrespective of any particular information need, as well as their relevance to the given user query. These two quantities are combined under the notion of *utility*. The utility of fact f is defined as a linear combination of its general importance and its relevance to the user query q:

$$utility(f,q) = \alpha \, importance(f) + (1 - \alpha) \, relevance(f,q) . \tag{9.3}$$

For simplicity, importance and relevance are given equal weights in [23]. The generation of factual summaries is addressed in two stages. First, facts are ranked according to their utility. Then, the top-ranked facts are visually arranged in order to be displayed on the entity card.

9.2.2.1 Fact Ranking

The ranking of facts is approached as a learning-to-rank problem, using two main groups of features, aimed at capturing either the *importance* or the *relevance* of a fact. To learn the ranking function, target labels are collected via crowdsourcing for each dimension separately on a 3-point scale (0..2). Then, the two are combined with equal weights (cf. Eq. (9.3)), resulting in a 5-point scale (0..4). Below, we shall introduce some of the most effective features developed for relevance and importance, respectively. Table 9.2 provides an overview. For a complete list of features, we refer to [23].

Importance features are mostly based on statistics derived from the knowledge base. We introduce the concepts of *fact frequency* and *entity frequency*, which are loosely analogous to collection frequency and document frequency in document retrieval. Specifically, the *fact frequency of object* is the number of SPO triples in the KB with a given object value:

$$FF_o(o_f) = \left|\{(s,p,o) : (s,p,o) \in \mathcal{K}, o = o_f\}\right| .$$

Table 9.2 Features for fact ranking

Group	Feature	Description
Importance		
	$NEF_p(f)$	Normalized entity frequency of the fact's predicate
	$typeImp(f,e)$	Type-based importance of the fact's predicate
	$predSpec(f)$	Predicate specificity
	$isEntity(f)$	Whether the fact's object is an entity
Relevance		
	$lexSim_o(f,q)$	Lexical similarity between the fact's object and the query
	$semSimAvg_o(f,q)$	Semantic similarity between the fact's object and the query
	$iRank(f,q)$	Inverse rank of the fact's object
	$conLen(q)$	Context length

Entity frequency of predicate is the number of entities in the KB that have at least one fact associated with them with a given predicate:

$$EF_p(p_f) = \left|\{e \in \mathcal{E} : \exists\, f' \in \mathcal{F}_e, p_{f'} = p_f\}\right| .$$

Another type-aware variant of the above statistic considers only those entities that, in addition to having the given predicate, also are of a given type y:

$$EF_p(p_f, y) = \left|\{e \in \mathcal{E} : y \in \mathcal{T}_e, \exists\, f' \in \mathcal{F}_e, p_{f'} = p_f\}\right| .$$

Furthermore, we define the *type frequency of a predicate* to be:

$$TF_p(p_f) = \left|\{y : \exists\, e \in \mathcal{E}, f' \in \mathcal{F}_e, p_{f'} = p_f, y \in \mathcal{T}_e\}\right| .$$

With the help of these statistics, we define the following features for measuring the importance of fact f for entity e:

- *Normalized entity frequency of predicate* is the relative frequency of the fact's predicate across all entities:

$$NEF_p(f) = \frac{EF_p(p_f)}{|\mathcal{E}|} ,$$

  where $|\mathcal{E}|$ is the total number of entities in the KB.
- *Type-based importance* also considers the frequency of the fact's predicate, but with respect to the types of the entity. Following [52], it is estimated using:

$$typeImp(f,e) = \sum_{y \in \mathcal{T}_e} EF_p(p_f, y) \log \frac{|\mathcal{T}|}{TF_p(p_f)} ,$$

  where $|\mathcal{T}|$ is the total number of types in the KB.

- *Predicate specificity* aims to identify facts with a common object value but a rare predicate:

$$predSpec(f) = FF_o(o_f) \log \frac{|\mathcal{E}|}{EF_p(p_f)} .$$

For example, the fact (*capital*, OTTAWA) would have relatively high predicate specificity, given that the object is frequent, while the predicate is relatively rare.

- *IsEntity* is a binary indicator that is true if the fact's object is an entity.

Relevance features capture the relevance of a fact with respect to the search query.

- *Lexical similarity* is measured by taking the maximum similarity between the terms of the fact's object and of the query using:

$$lexSim_o(f,q) = \max_{t \in o_f, t' \in q} \big(1 - dist(t,t')\big) ,$$

where *dist*() is a string distance function, taken to be the Jaro edit distance in [23].

- *Semantic similarity* aims to address the vocabulary mismatch problem, by computing similarity in a semantic embedding space. Specifically, we compute the average cosine similarity between terms of the fact's object and of the query:

$$semSimAvg_o(f,q) = \frac{\sum_{t \in o_f, t' \in q} \cos(\mathbf{t}, \mathbf{t}')}{\big|\{t : t \in o_f\}\big| \times |\{t' : t' \in q\}|} ,$$

where **t** denotes the embedding vector corresponding to term t. Hasibi et al. [23] use pre-trained Word2vec [33] vectors with 300 dimensions. The denominator is the multiplication of the number of unique terms in the fact's object and in the query, respectively.

- *Inverse rank* promotes facts with an entity that is highly relevant to the query as the object value. Entities in the KB are ranked with respect to the query. Then,

$$iRank(f,q) = \frac{1}{rank(o_f, \mathcal{E}_k(q))} ,$$

where $\mathcal{E}_k(q)$ is the set of top-k ranked entities returned in response to q, and *rank*() returns the position of an entity in the ranking (or ∞ if the entity cannot be found among the top-k results).

- *Context length* is the number of query terms that are not linked to any entity:

$$conLen(q) = |\{t : t \in q, t \notin linked(q)\}| ,$$

where $linked(q)$ denotes the set of query terms that are linked to an entity.

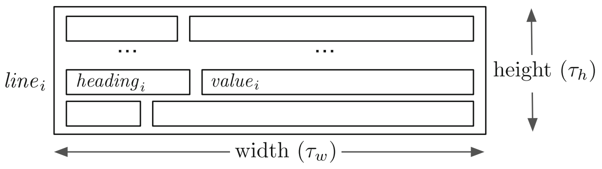


Fig. 9.9 Structure of an entity summary displayed on an entity card. Image is based on [23]

9.2.2.2 Summary Generation

The ranked list of facts we just obtained is yet to be arranged into a summary that can be presented to the user. Visually, a summary consists of a number of lines, each subdivided into heading and value parts. Additionally, it has a maximum display size, defined in terms of the maximum number of rows (τ_h) and the maximum number of characters within each row (τ_w); see Fig. 9.9. A straightforward approach is just to fill this structure with the top-ranked facts, by using the predicate label from the KB as the heading and the subject as the value part in each line. There are, however, some additional considerations that, if addressed, can yield higher quality summaries.

- There might be semantically identical predicates, even within a single KB, e.g., `<foaf:homepage>` and `<dbp:website>` in DBpedia. Such duplicates need to be identified and resolved, such that only one of them is included in the summary.
- There may be multiple facts with the same predicate, e.g., parents of a person or founders of a company. While these constitute separate facts, the object values of these so-called multi-valued predicates can be concatenated together into a single list for display purposes.

Hasibi et al. [23] address these issues with a summary generation algorithm, shown in Algorithm 9.2. Input to this algorithm is the list of top-k facts, generated by the fact ranking step, denoted as $\hat{\mathcal{F}}_e$. The first line of the summary generation algorithm creates a mapping from predicates in $\hat{\mathcal{F}}_e$ to their human-readable labels; these are commonly provided as part of the KB schema. Predicates that are mapped to the same label are then recognized as semantically identical. The mapping function may implement additional heuristics, specific to the underlying knowledge base. The summary is built in three stages. First (lines 2–8), up to τ_h unique line headings are selected. Second (lines 9–14), the values for each line are assembled. This is the part where multi-valued predicates are grouped. Third (lines 15–24), the heading and value parts are combined for each line.

Algorithm 9.2: Summary generation [23]

```
Input: ranked list of facts $\hat{\mathcal{F}}_e$, max height $\tau_h$, max width $\tau_w$
Output: entity summary $lines$

$M \leftarrow$ predicate-name mapping from $\hat{\mathcal{F}}_e$
$headings \leftarrow []$                                    /* Determine line headings */
foreach $f \in \hat{\mathcal{F}}_e$ do
    $label \leftarrow M[p_f]$
    if $(label \notin headings)$ and $(|headings| \leq \tau_h)$ then
        $headings$.append$\big((p_f, label)\big)$
    end
end
$values \leftarrow []$                                      /* Determine line values */
foreach $f \in \hat{\mathcal{F}}_e$ do
    if $p_f \in headings$ then
        $values[p_f]$.append$(o_f)$
    end
end
$lines \leftarrow []$                                       /* Construct lines */
foreach $(p_f, label) \in headings$ do
    $line \leftarrow label +$ ':'
    foreach $v \in values[p_f]$ do
        if len$(line)$ + len$(v) \leq \tau_w$ then
            $line \leftarrow line + v$                      /* Add comma if needed */
        end
    end
    $lines$.append$(line)$
end
```

9.3 Entity Recommendations

Earlier in this chapter, we have discussed tools that help users with expressing their information needs and with getting direct answers to those needs. There are also situations where users' information goals are less clearly defined, and they would just like to browse and explore, without looking for anything specific. Examples include learning about people in the news or exploring future travel destinations. Therefore, in addition to traditional search assistance tools, such as query suggestions and direct answers, exploration and discovery should also be regarded as central elements of next-generation search systems [55]. This section presents recommendation techniques that enable exploration, with the goal of increasing user engagement.

Specifically, our objective is to provide *related entity recommendations* (a.k.a. *related entity suggestions*) to users, based on their context. We shall consider multiple contexts that may serve as input data: (1) a particular entity (Sect. 9.3.1), (2) the user's current search session (Sect. 9.3.2), and (3) a given entity in a particular text context (Sect. 9.3.3).

The *entity recommendation* task is approached as a ranking problem: given some context (e.g., a particular entity or a search session) as input, return a ranked list of entities $\langle e_1, \ldots, e_k \rangle$ from an entity catalog $\mathcal{E}$ that are related to the user's context.

How does this problem relate to other tasks that have been discussed earlier in this book? A core component underlying all recommendation methods is a measure of *relatedness* between an input entity and a candidate entity. Pairwise entity relatedness has already been used in entity linking, for entity disambiguation (cf. Sect. 5.6.1.3), and those methods are applicable here as well. Another related task is that of finding similar entities (cf. Sect. 4.5), which also boils down to a pairwise entity measure. The similar entity finding task, however, has a different objective—it aims to complete an input set of entities, with similar entities. Consequently, it measures pairwise *entity similarity* as opposed to *entity relatedness*.

The degree of entity relatedness may be estimated using simple measures of statistical association, based on entity co-occurrence information (e.g., in search logs or in Wikipedia articles). Another family of methods makes use of entity relationship information stored in knowledge graphs and employs graph-theoretic measures (e.g., PageRank). Yet another group of approaches infers relatedness based on the content (i.e., attributes or descriptions) of entities.

In addition to receiving entity recommendations, users may also be interested in finding out *why* certain suggestions were shown to them. The problem of explaining recommendations boils down to the task of generating a human-readable description of the relationship between a pair of entities. This is discussed in Sect. 9.3.4.

9.3.1 Recommendations Given an Entity

We start by discussing the case of recommending entities related to a given input entity. A common application scenario is that of entity cards in web search, which are triggered by an entity-bearing query. These cards often include a "People also search for" section, displaying entities that are related to the query entity; see Fig. 9.8. This task may be formalized as the problem of estimating the probability of a candidate entity e', given an input entity e, $P(e'|e)$.

Blanco et al. [9] present the Spark system (with previous versions of the system described in [27, 58]), which had been powering related entity suggestions in Yahoo! Web Search. Spark extracts several signals (over 100 features) from a variety of proprietary and public data sources (including Yahoo!'s knowledge graph and web search logs, and social media platforms Flickr and Twitter) and combines them in a learning-to-rank framework. The training data consists of 47K entity pairs, labeled by professional editors on a five-point scale. Aggarwal et al. [2] show that comparable accuracy may be achieved by utilizing only publicly available data, in particular, Wikipedia, and using only 16 features.

Table 9.3 Features for related entity recommendation, given an input entity

Group	Feature	Description
Co-occurrence		
	$P(e,e')$	Joint probability $(c(e,e';\mathcal{C})/\lvert\mathcal{C}\rvert)$
	$P(e'\lvert e)$	Conditional probability $(c(e,e';\mathcal{C})/c(e;\mathcal{C}))$
	$P(e\lvert e')$	Reverse conditional probability $(c(e,e';\mathcal{C})/c(e';\mathcal{C}))$
	$PMI(e,e')$	Pointwise mutual information
	$WLM(e,e')$	Wikipedia link-based measure (cf. Eq. (5.4))
Graph-theoretic		
	$PR(e)$	PageRank score of the entity (cf. Eq. (4.4))
Content-based features		
	$\cos(\mathbf{e},\mathbf{e}')$	Cosine similarity between vector representations of entities
Popularity		
	$c(e;\mathcal{C})$	Frequency of the entity
	$P(e)$	Relative frequency of the entity $(c(e;\mathcal{C})/\lvert\mathcal{C}\rvert)$

Unary features are computed for both e and e'. All statistics are computed over some data collection $\mathcal{C}$, where $|\mathcal{C}|$ denotes the total number of items (documents, queries, etc.); $c(e;\mathcal{C})$ is the frequency of entity e, i.e., the number of items in which e occurs; $c(e,e';\mathcal{C})$ denote the number of items in which e and e' co-occur

We distinguish between four main groups of features: *co-occurrence features*, *graph-theoretic features*, *content-based features*, and *popularity features*. Popularity features and graph-theoretic features are unary, expressing the importance of an entity on its own. The remaining features are binary, capturing the strength of association between two entities. Table 9.3 presents a non-exhaustive selection of features.

Co-occurrence features Intuitively, entities that are observed to occur frequently together are likely to be related to each other. One question here is what data collection $\mathcal{C}$ to use for extracting co-occurrence information. Another question is what co-occurrence statistic to compute based on those observations. Prior work has considered a wide variety of data sources, including search logs [9], web pages [3], Wikipedia [2, 35, 44], Twitter [9], and Flickr [9]. Co-occurrence measures include joint, conditional, and reverse conditional probabilities, pointwise mutual information, KL divergence, entropy, and the Wikipedia link-based measure (WLM) [35].

Graph-theoretic features The most commonly used feature in this group is the PageRank score of an entity in the knowledge graph. PageRank may also be computed on a hyperlink graph obtained from the Web [9]. For details on PageRank and for additional centrality measures, we refer back to Sect. 4.6.

Content-based features This set of features aims to capture the similarity between a pair of entities based on their content. A standard approach is to represent entities either as term vectors or embedding vectors, and then compute the cosine similarity of those vectors. See Sect. 4.5.1 for alternative ways of measuring pairwise entity similarity.

Popularity features Popularity is based on the frequency of an entity in a given data source, e.g., search queries and sessions, or number of views or clicks in web search. Additional popularity features were discussed in Sects. 4.6.1 and 5.6.1.1.

9.3.2 Personalized Recommendations

Rather than suggesting entities related to a given input entity, in this section we discuss methods that provide personalized entity recommendations based on the user's current search session. A number of approaches have been proposed for learning models for specific domains, such as movies or celebrities [8, 57]. Such *model-based methods* rely on manually designed domain-dependent features, related to specific properties of entities (e.g., the genre of a movie or how many pop singers were viewed by a specific user). There is an obvious connection to make here to traditional item-based recommender systems (e.g., the ones used in e-commerce systems), such as collaborative filtering [19]. One main difference is that in collaborative filtering approaches the user-item matrix is given. For entity recommender systems, user preferences of entities are more difficult to observe. Another difference is the sheer scale of data (i.e., millions of entities vs. thousands of products in an e-commerce scenario) coupled with extreme data sparsity. Fernández-Tobías and Blanco [20] perform personalized entity recommendations using a purely collaborative filtering approach. Inspired by nearest neighbor techniques, these *memory-based methods* exploit user interactions that are available in search logs. Since they do not depend on descriptions or properties of entities, memory-based methods generalize to arbitrary types of entities and scale better to large amounts of training data than model-based methods.

Next, we present three probabilistic methods for estimating $P(e'|s)$, the probability of recommending entity e' to a user based on her *current search session* s.[4] These methods are named after how item-to-item similarity aggregation is performed: entity-based, query-based, or session-based. In their paper, Fernández-Tobías and Blanco [20] define multiple alternatives for each component of these models. Here, we will discuss a single option for each, the one that performed best experimentally. According to the results reported in [20], the entity-based method performs best, followed by the query-based and then the session-based approaches.

We shall use the following notation below. Let $\mathcal{Q}$ be the set of unique queries in the search log and $\mathcal{S}$ be the set of all user sessions. For a given session $s \in \mathcal{S}$, $\mathcal{Q}_s \subset \mathcal{Q}$ denotes the set of queries issued and $\mathcal{E}_s \subset \mathcal{E}$ denotes the set of entities clicked by the user within that search session.

[4]For notational consistency, we shall continue to denote the candidate entity recommendation, which is being scored, by e'.

9.3.2.1 Entity-Based Method

The intuition behind the first method is that an entity e' is more likely to be relevant to the user's current session s if it is similar to other entities that have previously been clicked by the user in the same session. Formally, this is expressed as:

$$P_{EB}(e'|s) = \sum_{e \in \mathcal{E}_s} P(e'|e)P(e|s) \,,$$

where $P(e'|e)$ captures the similarity between a pair of entities and $P(e|s)$ expresses the relevance of e given the search session s. Pairwise entity similarities are estimated in a collaborative fashion, by measuring the co-occurrence of entities within all user sessions using the Jaccard coefficient. To estimate the relevance of a clicked entity e within a session s, we aggregate the importance of e for each query q in that session, weighted by the query likelihood in that session:

$$P(e|s) = \sum_{q \in \mathcal{Q}_s} P(e|q,s)P(q|s) \,. \tag{9.4}$$

A given entity's relevance may be measured based on *dwell time*, i.e., how much time the user spent on examining that result, relative to all other entities that were returned for the same query:

$$P(e|q,s) = \frac{dwell(e,q,s)}{\sum_{e' \in \mathcal{E}_s} dwell(e',q,s)} \,, \tag{9.5}$$

where $dwell(e,q,s)$ is the time spent on examining entity e for query q in session s. In case the user did not click on e as a result to q, $P(e|q,s)$ is taken to be 0.

The probability $P(q|s)$ in Eq. (9.4) captures how important that query is within its session. It tends to reason that more recent queries should be considered more important, as they represent more accurately the user's current interests (which may have shifted over time). This notion of temporal decay is formally expressed as:

$$P(q|s) \propto \mathsf{e}^{-(t_s - t_q)} \,, \tag{9.6}$$

where e is the mathematical constant (the base of the natural logarithm), t_q is the timestamp of query q, and t_s is the timestamp of the last query in the session.

9.3.2.2 Query-Based Method

The second approach works by first identifying queries from other sessions in the search log that are potentially relevant to the current session. Then, it retrieves entities from those sessions. Formally:

$$P_{QB}(e'|s) = \sum_{\substack{q \in \mathcal{Q} \\ q \notin \mathcal{Q}_s}} P(e'|q)P(q|s) \,, \tag{9.7}$$

where $P(q|s)$ is the probability that query q is relevant to the current session s and $P(e'|q)$ measures how relevant e is for query q (across all sessions). Note that, in contrast to the entity-based method, the queries q we aggregate over are not present in the current session. Rather, these are chosen from the queries submitted by other users, performing similar tasks. The relevance of an entity given a query is estimated by considering all sessions in the search log that contain that query:

$$P(e'|q) \propto \sum_{s' \in \mathcal{S}} P(e'|q,s)P(q|s)P(s) \,, \tag{9.8}$$

where as before, $P(e'|q,s)$ is measured using dwell time (cf. Eq. (9.5)) and $P(q|s)$ is estimated based on temporal decay (cf. Eq. (9.6)). Note that $P(q|s)$ here expresses the probability of choosing the query q from session s containing that query ($q \in \mathcal{Q}_s$) and is not to be confused with $P(q|s)$ in Eq. (9.7), where it is used to capture the relevance of a query that is not observed in the given session ($q \notin \mathcal{Q}_s$). Finally, $P(s)$ is assumed to be uniform for simplicity.

The query relevance probability, $P(q|s)$ in Eq. (9.7), is defined to select queries from the search log ($q \in \mathcal{Q} \setminus \mathcal{Q}_s$) that are similar to the ones in the current session:

$$P(q|s) = \sum_{q' \in \mathcal{Q}_s} P(q|q')P(q'|s) \,,$$

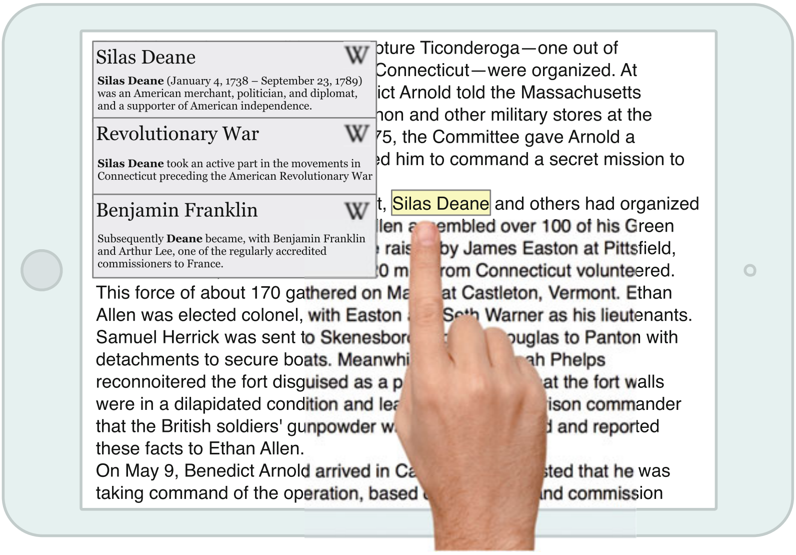
where $P(q|q')$ expresses the similarity between a pair of queries and is computed based on the co-occurrence of q and q' within all sessions in the search log using the Jaccard coefficient. As before, $P(q'|s)$ uses the temporal decay estimator (cf. Eq. (9.6)).

9.3.2.3 Session-Based Method

The last approach works by finding sessions similar to the current session, then recommending entities from those sessions:

$$P_{SB}(e'|s) = \sum_{s' \in \mathcal{S}} P(e'|s')P(s'|s) \,,$$

where $P(e'|s')$ is the importance of an entity given a session, computed as given by Eq. (9.4). The pairwise session similarity, $P(s'|s)$, is estimated based on entity embeddings. Specifically, Fernández-Tobías and Blanco [20] use Word2vec [33] (where sessions correspond to documents and entities within sessions correspond to words within documents) and extract embedding vectors of dimension 100. The



Silas Deane

Silas Deane (January 4, 1738 – September 23, 1789) was an American merchant, politician, and diplomat, and a supporter of American independence.

Revolutionary War

Silas Deane took an active part in the movements in Connecticut preceding the American Revolutionary War

Benjamin Franklin

Subsequently **Deane** became, with Benjamin Franklin and Arthur Lee, one of the regularly accredited commissioners to France.

pture Ticonderoga—one out of
Connecticut—were organized. At
ict Arnold told the Massachusetts
non and other military stores at the
75, the Committee gave Arnold a
ed him to command a secret mission to

t, Silas Deane and others had organized
llen a embled over 100 of his Green
rais by James Easton at Pittsfield,
0 m rom Connecticut volunteered.
This force of about 170 gathered on Ma at Castleton, Vermont. Ethan
Allen was elected colonel, with Easton Seth Warner as his lieutenants.
Samuel Herrick was sent to Skenesboro ouglas to Panton with
detachments to secure boats. Meanwhi ah Phelps
reconnoitered the fort disguised as a p at the fort walls
were in a dilapidated condition and lea ison commander
that the British soldiers' gunpowder w d and reported
these facts to Ethan Allen.
On May 9, Benedict Arnold arrived in Ca sted that he was
taking command of the operation, based nd commission

Fig. 9.10 An example of contextual entity recommendations. Image is based on [31]

similarity between two sessions is then computed based on the distance between the centroids of entities within them:

$$P(s'|s) \propto \left(\left\| \frac{1}{|\mathcal{E}_s|} \sum_{e \in \mathcal{E}_s} \mathbf{e} - \frac{1}{|\mathcal{E}_{s'}|} \sum_{e' \in \mathcal{E}_{s'}} \mathbf{e}' \right\| \right)^{-1} ,$$

where **e** is the embedding vector corresponding to entity e.

9.3.3 Contextual Recommendations

Web search is a prominent application area for entity recommendations but is certainly not the only one. Entity recommendation may also be offered directly within the application where content is consumed. As one such example, Lee et al. [31] present the scenario of a user reading a document on a tablet or e-reader device. At some point, the user might stumble upon an entity that she wishes to learn more about. Instead of leaving the application and switching to a web search engine to query for that entity, the user might just highlight and tap on an entity of interest. She will then be presented with a list of contextually relevant entities, as it is shown in Fig. 9.10.

According to the outlined scenario, the input to the *contextual entity recommendation* problem consists of an input entity e and some context c. Specifically, the

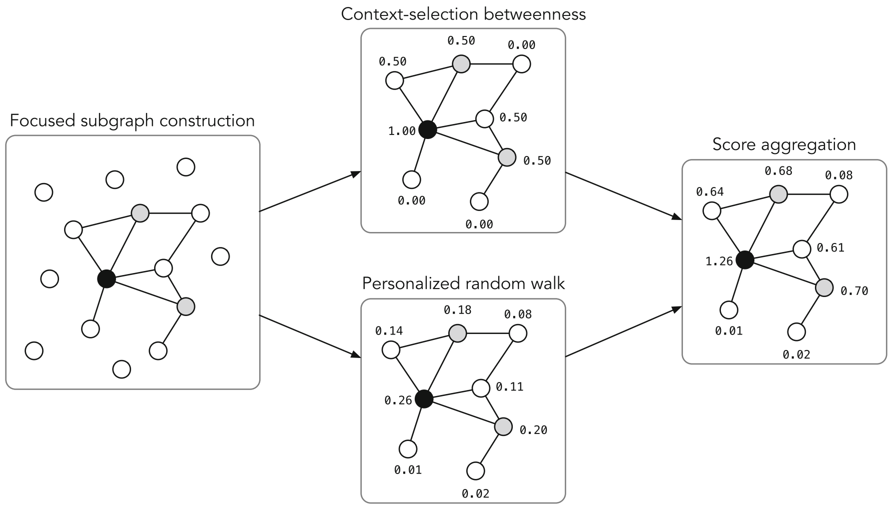


Fig. 9.11 Overview of the approach by Lee et al. [31]. The input entity (e) vertex is marked black, contextual entity vertices (E_c) are marked gray

context is a window of text around the selected entity mention (100 terms before and after in [31]). The approach proposed by Lee et al. [31] consists of three main steps, which are illustrated in Fig. 9.11.

1. A *focused subgraph* is extracted from the underlying knowledge graph G. The vertices of this focused subgraph are $V = \{e\} \cup \mathcal{E}_c \cup \mathcal{E}'_c$, where e is the input entity, $\mathcal{E}_c$ is the set of context entities, recognized in c by performing entity linking, and $\mathcal{E}'_c$ is the set of entities reachable from $\mathcal{E}_c$ via paths of length one in the knowledge graph. The edges between these vertices are induced from G.
2. Each candidate entity e' in the focused subgraph is scored using two different methods: context-selection betweenness and personalized random walk. *Context-selection betweenness* (CSB) captures the extent to which the candidate entity e' serves as a bridge between the input entity e and the context entities $\mathcal{E}_c$. Intuitively, a higher CSB score means that the candidate entity plays a more important role in connecting the input and context entities. Formally, CSB considers all shortest paths between the input and context entities, which go through the candidate entity:

$$CSB(e') = \frac{1}{Z} \sum_{e'':e' \in SP(e,e'')} \frac{w(e,e'')}{|SP(e,e'')| \times l(e,e'')} ,$$

where $SP(e,e'')$ is the set of all shortest paths between the e and e'', and $l(e,e'')$ is the length of that path. Each path between the input and a context entity is weighted by their semantic distance, based on the Wikipedia link-based measure

(WLM, cf. Eq. (5.4)):[5]

$$w(e,e'') = \max\big(WLM(e,e'') - \gamma, 0\big) \ .$$

The threshold γ is used for emphasizing context entities that are semantically related to the input entity ($\gamma = 0.5$ in [31]). The normalization factor is set to:

$$Z = \sum_{e'' \in \mathcal{E}_c} \frac{w(e,e'')}{l(e,e'')} \ .$$

The other scoring method is *personalized random walk* (PRW, a.k.a. personalized PageRank [26]), which aims to measure the relevance of entities given the user's selection. To compute these scores, the random jump probabilities are initialized as follows. The input entity vertex is assigned probability $0 < x_e < 1$, the context entity vertices are assigned probability $x_c/|\mathcal{E}_c|$, where $0 \leq x_c \leq x_e$, and all other entity vertices are assigned probability 0. Lee et al. [31] use $x_e = 0.05$ and $x_c = 0$ in their experiments, and report that $x_c > 0$ does not lead to significant improvements.

3. The final score for each entity vertex is computed by taking a weighted combination of the context-selection betweenness and personalized random walk scores:

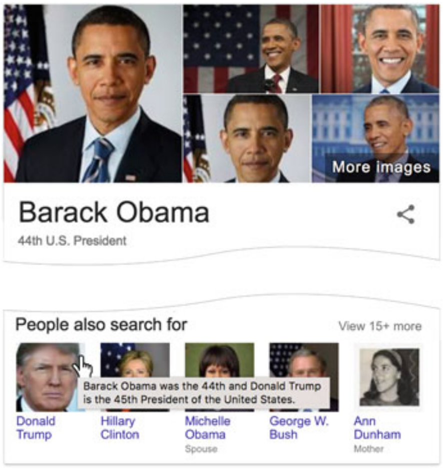
$$score(e'; e, \mathcal{E}_c) = \alpha \, \frac{|\mathcal{E}_c|}{|V|} \, CSB(e') + |V| \times PRW(e') \ ,$$

where $|V|$ is the number of vertices in the focused subgraph and α is a scaling factor. The multipliers serve normalization purposes, making the two scores compatible.

9.3.4 Explaining Recommendations

Thus far in this section, we have presented both non-personalized and personalized methods for recommending related entities (e') given an input entity (e). In addition to the recommendations themselves, users might also be interested in finding out *why* certain entities were shown to them. This brings us to answering the question: How are the input entity and the recommended entity related? Such explanations are offered, e.g., on entity cards in modern web search engines by hovering the mouse over a recommended entity; see Fig. 9.12. Another typical application scenario for explaining entity relationships is timeline generation [3].

[5] Mind that we define WLM as a similarity measure, as opposed to a distance measure, hence the equation differs from the one in [31].



Barack Obama was the 44th and Donald Trump is the 45th President of the United States.

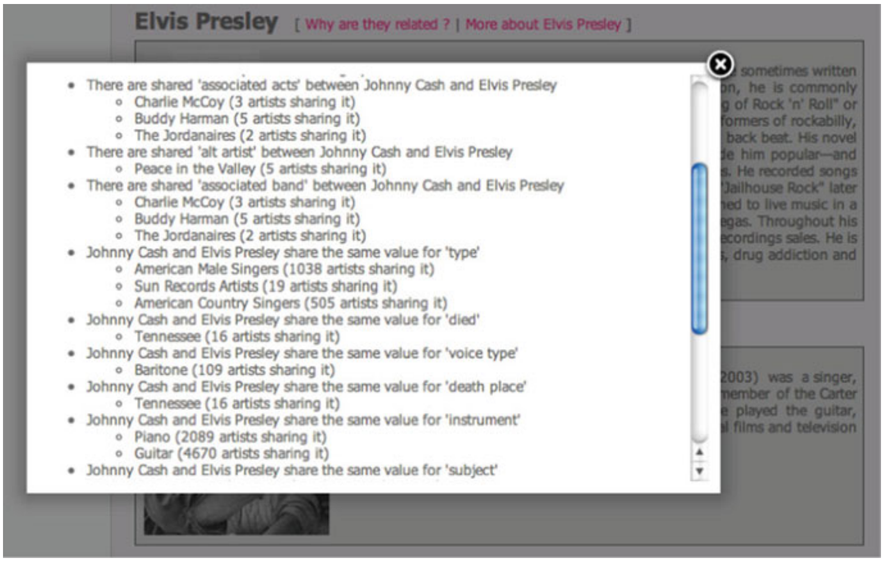
Fig. 9.12 Excerpt from a Google entity card displayed for BARACK OBAMA. When hovering over a related entity, an explanation of the relationship is shown

One of the earliest works addressing the problem of explaining the relationship between two entities is the *dbrec* music recommendation system [38]. It offers explanations in the form of a list of shared property-value pairs between the input and recommended entities, as shown in Fig. 9.13. This form of presentation, however, was considered as "too geeky" by 6 out of the 10 test subjects participating in the user evaluation [38]. Instead, human-readable descriptions that verbalize the relationship are more natural to use. The use of natural language has also shown to improve confidence in decision-making under uncertainty [21]. Three main lines of approaches have been proposed in prior work for generating natural language descriptions of entity relationships: (1) by manually defining templates [3], (2) by retrieving sentences from a text corpus [54], and (3) by automatically creating templates for a specific relationship and then filling the template slots for a new relationship instance [53]. Below, we briefly elaborate on the latter two.

All existing approaches solve a simplified version of the task of explaining entity relationships, by focusing on a specific relationship between a pair of entities. This corresponds to generating a textual description for an SPO triple, where the subject is e, the predicate is p, and the object is e'. We shall refer to the triple (e, p, e') as *relationship instance*. When referring to predicate p, we shall use the terms predicate and relationship interchangeably. As it is illustrated in Fig. 9.13, entities may be connected via multiple relationships. Selecting p from the set of possible predicates that connect two entities remains an open research challenge to date.

9.3.4.1 Explaining Relationships via Sentence Ranking

Voskarides et al. [54] approach the task as a sentence ranking problem: automatically extracting sentences from a text corpus and ranking them according to how well they describe a given relationship instance (e, p, e').



Elvis Presley [Why are they related ? | More about Elvis Presley]

- There are shared 'associated acts' between Johnny Cash and Elvis Presley
  - Charlie McCoy (3 artists sharing it)
  - Buddy Harman (5 artists sharing it)
  - The Jordanaires (2 artists sharing it)
- There are shared 'alt artist' between Johnny Cash and Elvis Presley
  - Peace in the Valley (5 artists sharing it)
- There are shared 'associated band' between Johnny Cash and Elvis Presley
  - Charlie McCoy (3 artists sharing it)
  - Buddy Harman (5 artists sharing it)
  - The Jordanaires (2 artists sharing it)
- Johnny Cash and Elvis Presley share the same value for 'type'
  - American Male Singers (1038 artists sharing it)
  - Sun Records Artists (19 artists sharing it)
  - American Country Singers (505 artists sharing it)
- Johnny Cash and Elvis Presley share the same value for 'died'
  - Tennessee (16 artists sharing it)
- Johnny Cash and Elvis Presley share the same value for 'voice type'
  - Baritone (109 artists sharing it)
- Johnny Cash and Elvis Presley share the same value for 'death place'
  - Tennessee (16 artists sharing it)
- Johnny Cash and Elvis Presley share the same value for 'instrument'
  - Piano (2089 artists sharing it)
  - Guitar (4670 artists sharing it)
- Johnny Cash and Elvis Presley share the same value for 'subject'

Fig. 9.13 Explanation for recommending ELVIS PRESLEY for the input entity JONNY CASH from the dbrec music recommendation system. Figure taken from Passant [38] (C) Springer 2010, reprinted with permission

First, a set $\mathcal{X}$ of candidate sentences is extracted from a corpus of documents. In [54], this corpus is Wikipedia. Other document collections may also be used, as long as documents are pertinent to the entities of interest. A sentence is considered as a candidate if (1) it originates from the Wikipedia page of e or e' and contains a mention to the other entity or (2) it mentions both e and e'. In order to make sentences self-contained outside the context of the source document, pronouns "she" and "he" are replaced with the name of the respective entity. Further, sentences are annotated with entities by performing entity linking. As an illustration, consider the sentence "He gave critically acclaimed performances in the crime thriller Seven...," which, after these enrichment steps, becomes "BRAD PITT gave critically acclaimed performances in the crime thriller SEVEN..."

Next, candidate sentences $x \in \mathcal{X}$ are ranked using supervised learning. Four groups of features are employed:

- *Textual features* consider the importance of the sentence on the term level. These include sentence length, aggregated IDF scores, sentence density [30], and fractions of verbs, nouns, and adjectives.
- *Entity features* characterize the sentence based on the mentioned entities. These include, among others, whether e and e' are linked in x, and the distance between the positions of their mentions. Another group of features focuses on other entities mentioned in the sentence and whether those are related to e and e'.

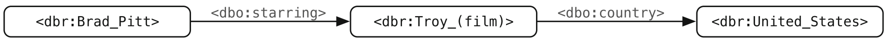


Fig. 9.14 Dependency graph for the sentence "Brad Pitt appeared in the American epic adventure film Troy," using entities and predicates from DBpedia

- *Relationship features* indicate whether the relationship p occurs in x. Exact term-based matching has low coverage (e.g., "spouse" vs. "husband" or "married"), therefore both synonym-based matches (using Wordnet) and word embeddings (using Word2vec [33]) are considered.
- *Source features* describe the position of x and the number of occurrences of e and e' in the document from which x originates.

Voskarides et al. [54] train their models on a set of manually annotated sentences, using a five-level graded relevance scale. They show that learning relationship-specific models, as opposed to a single global model, can yield additional improvements.

9.3.4.2 Generating Descriptions of Relationships

The previous approach is limited by the underlying text corpus, which may not contain descriptions for certain relationship instances. Voskarides et al. [53] propose to overcome this by automatically generating descriptions. The idea is to learn how a given relationship p is typically expressed (in the document corpus), and create sentence templates for that relationship. Then, for a new relationship instance, the appropriate template can be instantiated.

As before, it is assumed that a given relationship between two entities can be expressed as a single sentence. This sentence should mention both e and e', and possibly other entities that may provide additional contextual information. The following example sentence is given as an illustration in [53] for the (BRAD PITT, stars in, TROY) relationship instance: "Brad Pitt appeared in the American epic adventure film Troy." It not only verbalizes the "stars in" predicate but also mentions other entities and attributes (the film's genre and origin) to offer additional context. To be able to provide such contextual information, each sentence is augmented with an entity dependency graph. In this graph, vertices represent entities and edges represent relationships (predicates). The graph is created by traversing paths in the knowledge base between each pair of entities that are mentioned in the sentence. See Fig. 9.14 for an illustration.[6]

[6]In their paper, Voskarides et al. [53] use Freebase as the underlying knowledge base and pay special attention to compound value type (CVT) entities. CVT entities are specific to Freebase, and are used for modeling attributes of relationships (e.g., date of a marriage). For ease of presentation, we will not deal with CVT entities in our discussion.

The template creation process then takes as input, for each predicate, a set of relationship instances, sentences describing those relationship instances, and entity dependency graphs corresponding to those sentences. The following sequence of steps are performed:

1. Entities sharing the same predicates are clustered together across the dependency graphs. This will group entities of the same type, such as persons and films.
2. A compression graph G_C is created from sentences where vertices are either words or entity clusters.
3. G_C is traversed for finding valid paths between all pairs of entity cluster vertices. A path is considered valid if (i) it contains a verb and (ii) it can be observed as a complete sentence at least once in the corpus.
4. A template is constructed from each path, which is supported by a minimum number of sentences in the corpus.

With a set of templates at hand, generating a description for a new relationship instance (e, p, e') goes as follows. First, the available templates for predicate p are ranked. Two template scoring functions are presented in [53], one based on cosine similarity of TF-IDF term vectors and another using feature-based supervised learning. The top-ranked template is then instantiated by filling its slots with entities from the knowledge base. If multiple instantiations exist, then one of those is chosen randomly. If the template cannot be instantiated, then we proceed to the next template in the ranking.

9.4 Summary

This chapter has introduced search assistance tools that help users (1) express their information needs, (2) get direct answers, and (3) explore related content with the help of entities. We have started with query assistance features, such as query auto-completion and query recommendation, which users would expect today as standard functionality from a modern search engine. An issue with traditional methods, which rely solely on query logs, is that of coverage. That is, they fail to provide meaningful suggestions for long-tail queries. We have discussed how knowledge bases may be utilized to alleviate this problem, yielding small but significant improvements over traditional methods. Next, we have looked at entity cards, a new type of search result presentation that has been adopted by major web search engines and intelligent personal assistants. Each card presents a concise summary of a specific entity, and can satisfy the user's information need directly, while also encouraging further engagement with search results and exploration of related content. We have addressed, in detail, the question of which facts to highlight about an entity in the limited screen space that is available on the card. Finally, we have presented techniques for promoting exploratory search by recommending related entities to users. We have further discussed how to generate a natural language explanation for the relationship between an input entity and a recommended related entity.

9.5 Further Reading

Web search result pages are becoming increasingly complex, featuring direct displays, entity cards, query suggestions, advertisements, etc., arranged in a nonlinear page layout. With these novel interfaces, determining the user's satisfaction with the search results is becoming more difficult. Research in this area includes the topics of evaluating whole page relevance [4], understanding how users examine and interact with nonlinear page layouts [13, 36], and detecting search satisfaction without clicks [28, 56].

Entity cards are the most widely used and universally applicable tools for summarizing entity information, but there are other possibilities that could serve users better in certain application scenarios. One such alternative that has garnered research interest is *entity timelines*, which organize information associated with an entity, arranged along a horizontal time axis. Timeline visualizations are often coupled with interactive features to enable further exploration. For example, Rybak et al. [40] visualize how a person's expertise changes over the course of time. Tuan et al. [51] and Althoff et al. [3] generate a timeline of events and relations for entities in a knowledge base.

In this chapter, we have focused on the algorithmic aspects of generating entity recommendations. For a study on how people interact with such recommendations, see, e.g., [34].

References

1. Agarwal, G., Kabra, G., Chang, K.C.C.: Towards rich query interpretation: Walking back and forth for mining query templates. In: Proceedings of the 19th international conference on World wide web, WWW '10, pp. 1–10. ACM (2010). doi: 10.1145/1772690.1772692
2. Aggarwal, N., Mika, P., Blanco, R., Buitelaar, P.: Leveraging Wikipedia knowledge for entity recommendations. In: Proceedings of the ISWC 2015 Posters & Demonstrations Track co-located with the 14th International Semantic Web Conference, ISWC '15. Springer (2015)
3. Althoff, T., Dong, X.L., Murphy, K., Alai, S., Dang, V., Zhang, W.: TimeMachine: Timeline generation for knowledge-base entities. In: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '15, pp. 19–28. ACM (2015). doi: 10.1145/2783258.2783325
4. Bailey, P., Craswell, N., White, R.W., Chen, L., Satyanarayana, A., Tahaghoghi, S.M.M.: Evaluating whole-page relevance. In: Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '10, pp. 767–768. ACM (2010). doi: 10.1145/1835449.1835606
5. Bar-Yossef, Z., Kraus, N.: Context-sensitive query auto-completion. In: Proceedings of the 20th International Conference on World Wide Web, WWW '11, pp. 107–116. ACM (2011). doi: 10.1145/1963405.1963424
6. Bast, H., Bäurle, F., Buchhold, B., Haussmann, E.: Semantic full-text search with broccoli. In: Proceedings of the 37th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '14, pp. 1265–1266. ACM (2014). doi: 10.1145/2600428.2611186

7. Bast, H., Buchhold, B., Haussmann, E.: Relevance scores for triples from type-like relations. In: Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '15, pp. 243–252. ACM (2015). doi: 10.1145/2766462.2767734
8. Bi, B., Ma, H., Hsu, B.J.P., Chu, W., Wang, K., Cho, J.: Learning to recommend related entities to search users. In: Proceedings of the Eighth ACM International Conference on Web Search and Data Mining, WSDM '15, pp. 139–148. ACM (2015). doi: 10.1145/2684822.2685304
9. Blanco, R., Cambazoglu, B.B., Mika, P., Torzec, N.: Entity recommendations in web search. In: Proceedings of the 12th International Semantic Web Conference, ISWC '13, pp. 33–48. Springer (2013). doi: 10.1007/978-3-642-41338-4_3
10. Boldi, P., Bonchi, F., Castillo, C., Donato, D., Gionis, A., Vigna, S.: The query-flow graph: Model and applications. In: Proceedings of the 17th ACM Conference on Information and Knowledge Management, CIKM '08, pp. 609–618. ACM (2008). doi: 10.1145/1458082.1458163
11. Bonchi, F., Perego, R., Silvestri, F., Vahabi, H., Venturini, R.: Efficient query recommendations in the long tail via center-piece subgraphs. In: Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '12, pp. 345–354. ACM (2012). doi: 10.1145/2348283.2348332
12. Bordino, I., De Francisci Morales, G., Weber, I., Bonchi, F.: From Machu_Picchu to "rafting the urubamba river": Anticipating information needs via the entity-query graph. In: Proceedings of the Sixth ACM International Conference on Web Search and Data Mining, WSDM '13, pp. 275–284. ACM (2013). doi: 10.1145/2433396.2433433
13. Bota, H.: Nonlinear composite search results. In: Proceedings of the 2016 ACM on Conference on Human Information Interaction and Retrieval, CHIIR '16, pp. 345–347. ACM (2016). doi: 10.1145/2854946.2854956
14. Bota, H., Zhou, K., Jose, J.M.: Playing your cards right: The effect of entity cards on search behaviour and workload. In: Proceedings of the 2016 ACM on Conference on Human Information Interaction and Retrieval, CHIIR '16, pp. 131–140. ACM (2016). doi: 10.1145/2854946.2854967
15. Cai, F., de Rijke, M.: A Survey of Query Auto Completion in Information Retrieval, vol. 10. Now Publishers Inc. (2016)
16. Cheng, G., Tran, T., Qu, Y.: RELIN: Relatedness and informativeness-based centrality for entity summarization. In: Proceedings of the 10th International Conference on The Semantic Web - Volume Part I, ISWC'11, pp. 114–129. Springer (2011). doi: 10.1007/978-3-642-25073-6_8
17. Dalvi, B., Minkov, E., Talukdar, P.P., Cohen, W.W.: Automatic gloss finding for a knowledge base using ontological constraints. In: Proceedings of the Eighth ACM International Conference on Web Search and Data Mining, WSDM '15, pp. 369–378. ACM (2015). doi: 10.1145/2684822.2685288
18. Dehghani, M., Rothe, S., Alfonseca, E., Fleury, P.: Learning to attend, copy, and generate for session-based query suggestion. In: Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, CIKM '17, pp. 1747–1756. ACM (2017). doi: 10.1145/3132847.3133010
19. Desrosiers, C., Karypis, G.: A Comprehensive Survey of Neighborhood-based Recommendation Methods, pp. 107–144. Springer (2011)
20. Fernández-Tobías, I., Blanco, R.: Memory-based recommendations of entities for web search users. In: Proceedings of the 25th ACM International on Conference on Information and Knowledge Management, CIKM '16, pp. 35–44. ACM (2016). doi: 10.1145/2983323.2983823
21. Gkatzia, D., Lemon, O., Rieser, V.: Natural language generation enhances human decision-making with uncertain information. In: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL' 16. The Association for Computer Linguistics (2016)

22. Gunaratna, K., Thirunarayan, K., Sheth, A.: FACES: Diversity-aware entity summarization using incremental hierarchical conceptual clustering. In: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, AAAI'15, pp. 116–122. AAAI Press (2015)
23. Hasibi, F., Balog, K., Bratsberg, S.E.: Dynamic factual summaries for entity cards. In: Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '17. ACM (2017). doi: 10.1145/3077136.3080810
24. Hoffart, J., Milchevski, D., Weikum, G.: STICS: Searching with strings, things, and cats. In: Proceedings of the 37th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '14, pp. 1247–1248. ACM (2014). doi: 10.1145/2600428.2611177
25. Huang, Z., Cautis, B., Cheng, R., Zheng, Y.: KB-enabled query recommendation for long-tail queries. In: Proceedings of the 25th ACM International on Conference on Information and Knowledge Management, CIKM '16, pp. 2107–2112. ACM (2016). doi: 10.1145/2983323.2983650
26. Jeh, G., Widom, J.: Scaling personalized web search. In: Proceedings of the 12th International Conference on World Wide Web, WWW '03, pp. 271–279. ACM (2003). doi: 10.1145/775152.775191
27. Kang, C., Vadrevu, S., Zhang, R., van Zwol, R., Pueyo, L.G., Torzec, N., He, J., Chang, Y.: Ranking related entities for web search queries. In: Proceedings of the 20th International Conference Companion on World Wide Web, WWW '11, pp. 67–68. ACM (2011). doi: 10.1145/1963192.1963227
28. Lagun, D., Hsieh, C.H., Webster, D., Navalpakkam, V.: Towards better measurement of attention and satisfaction in mobile search. In: Proceedings of the 37th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '14, pp. 113–122. ACM (2014). doi: 10.1145/2600428.2609631
29. Lao, N., Cohen, W.W.: Relational retrieval using a combination of path-constrained random walks. Mach. Learn. **81**(1), 53–67 (2010). doi: 10.1007/s10994-010-5205-8
30. Lee, H., Peirsman, Y., Chang, A., Chambers, N., Surdeanu, M., Jurafsky, D.: Stanford's multi-pass sieve coreference resolution system at the CoNLL-2011 Shared task. In: Proceedings of the Fifteenth Conference on Computational Natural Language Learning: Shared Task, CONLL Shared Task '11, pp. 28–34. Association for Computational Linguistics (2011)
31. Lee, J., Fuxman, A., Zhao, B., Lv, Y.: Leveraging knowledge bases for contextual entity exploration. In: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '15, pp. 1949–1958. ACM (2015). doi: 10.1145/2783258.2788564
32. Meij, E., Mika, P., Zaragoza, H.: An evaluation of entity and frequency based query completion methods. In: Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '09, pp. 678–679. ACM (2009). doi: 10.1145/1571941.1572074
33. Mikolov, T., Sutskever, I., Chen, K., Corrado, G., Dean, J.: Distributed representations of words and phrases and their compositionality. In: Proceedings of the 26th International Conference on Neural Information Processing Systems, NIPS'13, pp. 3111–3119. Curran Associates Inc. (2013)
34. Miliaraki, I., Blanco, R., Lalmas, M.: From "Selena Gomez" to "Marlon Brando": Understanding explorative entity search. In: Proceedings of the 24th International Conference on World Wide Web, WWW '15, pp. 765–775. International World Wide Web Conferences Steering Committee (2015). doi: 10.1145/2736277.2741284
35. Milne, D., Witten, I.H.: An effective, low-cost measure of semantic relatedness obtained from Wikipedia links. In: Proceeding of AAAI Workshop on Wikipedia and Artificial Intelligence: An Evolving Synergy, pp. 25–30. AAAI Press (2008)
36. Navalpakkam, V., Jentzsch, L., Sayres, R., Ravi, S., Ahmed, A., Smola, A.: Measurement and modeling of eye-mouse behavior in the presence of nonlinear page layouts. In: Proceedings of the 22nd International Conference on World Wide Web, WWW '13, pp. 953–964. ACM (2013). doi: 10.1145/2488388.2488471

37. Ozertem, U., Chapelle, O., Donmez, P., Velipasaoglu, E.: Learning to suggest: A machine learning framework for ranking query suggestions. In: Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '12, pp. 25–34. ACM (2012). doi: 10.1145/2348283.2348290
38. Passant, A.: Dbrec: Music recommendations using DBpedia. In: Proceedings of the 9th International Semantic Web Conference on The Semantic Web - Volume Part II, ISWC'10, pp. 209–224. Springer (2010)
39. Reinanda, R., Meij, E., de Rijke, M.: Mining, ranking and recommending entity aspects. In: Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '15, pp. 263–272. ACM (2015). doi: 10.1145/2766462.2767724
40. Rybak, J., Balog, K., Nørvåg, K.: ExperTime: Tracking expertise over time. In: Proceedings of the 37th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '14 (2014). doi: 10.1145/2600428.2611190
41. Saldanha, G., Biran, O., McKeown, K., Gliozzo, A.: An entity-focused approach to generating company descriptions. In: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL '16. The Association for Computer Linguistics (2016)
42. Schmidt, A., Hoffart, J., Milchevski, D., Weikum, G.: Context-sensitive auto-completion for searching with entities and categories. In: Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '16, pp. 1097–1100. ACM (2016). doi: 10.1145/2911451.2911461
43. Sordoni, A., Bengio, Y., Vahabi, H., Lioma, C., Grue Simonsen, J., Nie, J.Y.: A hierarchical recurrent encoder-decoder for generative context-aware query suggestion. In: Proceedings of the 24th ACM International on Conference on Information and Knowledge Management, CIKM '15, pp. 553–562. ACM (2015). doi: 10.1145/2806416.2806493
44. Strube, M., Ponzetto, S.P.: WikiRelate! - Computing semantic relatedness using Wikipedia. In: Proceedings of the 21st National Conference on Artificial Intelligence - Volume 2, AAAI'06, pp. 1419–1424. AAAI Press (2006)
45. Sun, Y., Han, J., Yan, X., Yu, P.S., Wu, T.: Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. Proceedings of the VLDB Endowment **4**(11), 992–1003 (2011)
46. Szpektor, I., Gionis, A., Maarek, Y.: Improving recommendation for long-tail queries via templates. In: Proceedings of the 20th International Conference on World Wide Web, WWW' 11, pp. 47–56. ACM (2011). doi: 10.1145/1963405.1963416
47. Thalhammer, A., Lasierra, N., Rettinger, A.: LinkSUM: Using link analysis to summarize entity data. In: Proc. of 16th International Web Engineering Conference, ICWE '16, pp. 244–261. Springer (2016). doi: 10.1007/978-3-319-38791-8_14
48. Thalhammer, A., Rettinger, A.: Browsing DBpedia entities with summaries. In: The Semantic Web: ESWC 2014 Satellite Events, pp. 511–515 (2014)
49. Tombros, A., Sanderson, M.: Advantages of query biased summaries in information retrieval. In: Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '98, pp. 2–10. ACM (1998). doi: 10.1145/290941.290947
50. Tonon, A., Catasta, M., Prokofyev, R., Demartini, G., Aberer, K., Cudré-Mauroux, P.: Contextualized ranking of entity types based on knowledge graphs. Web Semant. **37–38**, 170–183 (2016). doi: 10.1016/j.websem.2015.12.005
51. Tuan, T.A., Elbassuoni, S., Preda, N., Weikum, G.: CATE: Context-aware timeline for entity illustration. In: Proceedings of the 20th International Conference Companion on World Wide Web, WWW '11, pp. 269–272. ACM (2011). doi: 10.1145/1963192.1963306
52. Vadrevu, S., Tu, Y., Salvetti, F.: Ranking relevant attributes of entity in structured knowledge base (2016)
53. Voskarides, N., Meij, E., de Rijke, M.: Generating descriptions of entity relationships. In: Proceedings of the 39th European Conference on Information Retrieval, ECIR '17. Springer (2017). doi: 10.1007/978-3-319-56608-5_25

54. Voskarides, N., Meij, E., Tsagkias, M., de Rijke, M., Weerkamp, W.: Learning to explain entity relationships in knowledge graphs. In: Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), pp. 564–574. Association for Computational Linguistics (2015)
55. White, R.W.: Interactions with Search Systems. Cambridge University Press (2016)
56. Williams, K., Kiseleva, J., Crook, A.C., Zitouni, I., Awadallah, A.H., Khabsa, M.: Is this your final answer?: Evaluating the effect of answers on good abandonment in mobile search. In: Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '16, pp. 889–892. ACM (2016). doi: 10.1145/2911451.2914736
57. Yu, X., Ma, H., Hsu, B.J.P., Han, J.: On building entity recommender systems using user click log and Freebase knowledge. In: Proceedings of the 7th ACM International Conference on Web Search and Data Mining, WSDM '14, pp. 263–272. ACM (2014). doi: 10.1145/2556195.2556233
58. van Zwol, R., Pueyo, L.G., Muralidharan, M., Sigurbjornsson, B.: Ranking entity facets based on user click feedback. In: Proceedings of the 4th IEEE International Conference on Semantic Computing, pp. 192–199 (2010)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.