Chapter 4
Semantically Enriched Models for Entity Ranking

Most of our efforts in the previous chapter have revolved around constructing term-based representations of entities. These representations can then be ranked using direct adaptations of existing document retrieval models. On the one hand, the resulting approaches are robust and effective across a broad range of application scenarios. On the other hand, these term-based models have little awareness of what it takes to be an entity. Perhaps the most exciting challenge and opportunity in entity retrieval is how to leverage entity-specific properties—attributes, types, and relationships—to improve retrieval performance. This requires a departure from purely term-based approaches toward more semantically informed representations. This change of direction is supported by the emergence of knowledge bases over the past decade (cf. Sect. 2.3). Knowledge bases organize information about entities in a structured and semantically meaningful way. For us, semantics is taken to be synonymous with structure (more precisely, with references to meaningful structure). Our efforts in this chapter are driven by the following question: How can one leverage structured knowledge repositories in entity retrieval?

At its core, the entity ranking task (and most IR tasks for that matter) boils down to the problem of matching representations. That is, computing similarities between representations of queries (information needs) and those of entities (information objects). The question then becomes: How to preserve and represent structure associated with entities? Importantly, to be able to make use of richer (i.e., semantic) entity representations during matching, queries also need to have correspondingly enriched representations. For example, if types of entities are represented as semantic units, as opposed to sequences of words, then we also need to know the target types of the query. For now, we shall assume that we are provided with such enriched queries, which will be referred to as keyword++ queries.

In this chapter, we will look at semantically enriched entity retrieval in several flavors. Table 4.1 presents an overview of the different tasks, and serves as a roadmap to this chapter. Many of these tasks have been proposed and studied in the context of some benchmarking campaign. As we shall see, some of the most
effective approaches are tailor-made and highly specialized for the particular task. This chapter is mainly organized around the various aspects of entities that are utilized: properties generally (Sect. 4.2), then more specifically types (Sect. 4.3) and relationships (Sect. 4.4). In Sect. 4.5, we consider the task of similar entity search, which revolves around comparing representations of entities. Finally, in Sect. 4.6, we show that structure can also be exploited in a static (query-independent) fashion. Table 4.2 summarizes the notation used throughout this chapter.
4.1 Semantics Means Structure

Our objective in this chapter is to build semantically enriched entity retrieval models. We introduce the following working definition of semantics: “references to meaningful structures.” What we mean by that is that specific entities, types, or relationships are recognized and identified uniquely, with references to an underlying knowledge repository, as opposed to being treated as mere strings. This makes it possible to search by meaning rather than just literal matches.

Semantically enriched entity retrieval models extend the representation of entities from mere sequences of terms to include information about specific attributes, types, and relationships, and leverage this structured information when matching entities against information needs (queries).

This semantic enrichment needs to be woven through all elements of the retrieval process. In order to make use of rich entity representations, the retrieval model has to utilize these additional structures during matching. To be able to do that, queries also need to be enriched. See Fig. 4.1a vs. b for the illustration of the difference between term-based and semantic entity retrieval models.

The enrichment of queries may happen on the users’ side or may be performed automatically. The former is typically facilitated through various query assistance services, such as facets or query auto-completion. For expressing complex information needs, custom user interfaces may also be built (see, e.g., [9, 36]). While such interactive query-builders offer the same expressivity as structured query languages (like SQL and SPARQL), they share the same disadvantages as well: users need to receive some training on how to use the tool. The other alternative is to rely on machine understanding of queries, i.e., to obtain semantic enrichments by automatic means. We will look at this direction in detail in Chap. 7. Finally, hybrid approaches that combine human and machine annotations are also possible.

Our focus in this chapter is not on the mechanics of query enrichment. We assume an enriched query as our input; how exactly it was obtained is presently immaterial for us. We make the following distinction for notational convenience.

![Diagram](image)

**Fig. 4.1** (a) Ranking entities using term-based representations. (b) Semantically enriched entity ranking by incorporating structure
We use $\tilde{q}$ for a semantically enriched, i.e., keyword++, query. When referring to the keyword component of the query, we shall write $q$. In many cases, our input query will be a tuple $\tilde{q} = (q, X_q, Y_q, \ldots)$, where $X$ and $Y$ are the additional query components or “enrichments” (e.g., target types or example entities). A typical approach, which we shall encounter several times throughout this chapter, is to build multiple representations of both entities and queries, in addition to the term-based one. Each of these additional “parallel representations” is designated to preserve the semantics associated with a specific entity property (e.g., types or relationships). Then, a given candidate entity is scored against the query based on each of these representations. Finally, these scores are combined, e.g., linearly:

$$
score(e; \tilde{q}) = \lambda_t score_t(e; q) + \lambda_X score_X(e; X_q) + \lambda_Y score_Y(e; Y_q) + \ldots,
$$

where $score_t(e; q)$ is the term-based retrieval score (using methods from the previous chapter), and the other score components correspond to the various other representations ($X, Y, \ldots$).

One important detail that needs attention when using the above formulation is that the different similarity scores need to be “compatible,” i.e., must have the same magnitude. A simple technique to ensure compatibility is to normalize the top-$k$ results (using the same $k$ value across the different score components) by the sum of the scores for that query, and assign a zero score to results below rank $k$:

$$
score'_x(e; q) = \begin{cases} 
\frac{1}{Z} score_x(e; q), & e \in E_q(k) \\
0, & \text{otherwise}
\end{cases},
$$

where $E_q(k)$ denotes the set of top-$k$ results (entities) for query $q$ and the normalization coefficient is set to $Z = \sum_{e \in E_q(k)} score_x(e; q)$.

### 4.2 Preserving Structure

In this section, we look at how to preserve (and exploit) the rich structure associated with entities in a knowledge base. We will assume that each entity is stored as a set of SPO triples (cf. Sect. 2.3.1.2). Recall that the subject and predicate are always URIs, while the object can be a URI or a literal value. Previously, in Chap. 3, we have built fielded entity descriptions by grouping multiple predicates together into each field. The corresponding object values have been concatenated and set as that field’s value. Further, we have replaced object URIs with the primary name (or label) of the given object (cf. Table 3.4). The resulting term-based entity representations are well suited for use with existing retrieval methods. On the other hand, most of the rich structure that is associated with an entity has been lost in the process.

We will now modify our approach in order to preserve the semantic information encapsulated in SPO triples. Two specific issues will be covered. The first is the case of multi-valued predicates (i.e., triples with the same subject and predicate, but with
multiple object values). While these are implicitly handled to some extent when using proximity-aware retrieval models (cf. Sect. 3.3.2.4), we will address “multi-valuedness” more explicitly in Sect. 4.2.1. The second is the case of URI-valued objects, which are references to other entities. Instead of simply replacing these with the corresponding entity names and treating them as terms, we will consider them as first-class citizens and distinguish them from regular terms when constructing entity representations (Sect. 4.2.2).

### 4.2.1 Multi-Valued Predicates

For the issue of multi-valued predicates, we will continue to use a term-based entity representation. Accordingly, all entity properties may be seen as attributes (since types and related entities are also “just strings” in this case). The one adjustment we make is that we keep each predicate as a separate field, instead of folding predicates together. Thus, each field corresponds to a single predicate. Some of the predicates are multi-valued, i.e., have more than a single value. For example, a person may have multiple email addresses and most movies have multiple actors. Campinas et al. [14] present an extension to BM25F, called BM25MF, for dealing with multi-valued fields. Even though we focus exclusively on BM25F here, we note that similar extensions may be developed for other fielded retrieval models as well.

For convenience, we repeat how term frequencies are aggregated across different fields according to the original BM25F formula (cf. Sect. 3.3.2.3):

\[
\tilde{c}(t; e) = \sum_{f \in F} \alpha_f \frac{c(t; f_e)}{1 - b_f + b_f \frac{\overline{l_f}}{\overline{lf}}}. 
\]

Recall that for a given field \(f\), \(\alpha_f\) is the field’s weight, \(b_f\) is a field length normalization parameter, and \(\overline{l_f}\) is the average length of the field; these values are the same for all entities in the catalog. For a specific entity, \(e\), \(c(t; f_e)\) is the frequency of term \(t\) in field \(f\) that entity and \(l_{fe}\) is the length of the field (number of terms).

According to the multi-valued extension BM25MF, the entity term frequency becomes:

\[
\tilde{c}(t; e) = \sum_{f \in F} \alpha_f \frac{\tilde{c}(t; f_e)}{1 - b_f + b_f \frac{|f_e|}{|f|}}. 
\]

Here, \(|f_e|\) is the cardinality of field \(f\) of \(e\), while \(|f|\) denotes the average cardinality of field \(f\) across all entities in the catalog. Field cardinality refers to the number of distinct values in a field. Term frequencies are computed with respect to a given entity field according to:

\[
\tilde{c}(t; f_e) = \sum_{v \in f_e} \alpha_v \frac{c(t; f_{e,v})}{1 - b_v + b_v \frac{l_v}{l_f}}, 
\]
where \( c(t; f, v) \) is the term frequency within the specific field value \( v \). The length of the value, \( l_v \), is measured in the number of terms. Finally, \( \alpha_v \) and \( b_v \) are value-specific weights and normalization parameters, respectively.

### 4.2.1.1 Parameter Settings

For setting the field and value weight parameters (\( \alpha_f \) and \( \alpha_v \)), Campinas et al. [14] introduce a number of options, both query-independent and query-dependent. It is also possible to combine multiple weighting methods (by multiplying the parameters produced by each of the methods).

Recall that each field in the entity description corresponds to a unique predicate (URI). We write \( p_f \) to denote the predicate value that is assigned to field \( f \). Field weights can be defined heuristically based on \( p_f \), using a small set of regular expressions:

\[
\alpha_f = \begin{cases} 
2.0, & p_f \text{ matches } .*[\text{label}|\text{name}|\text{title}|\text{sameas}]$
\vspace{.2em}
0.5, & p_f \text{ matches } .*[\text{seealso}|\text{wikilinks}]$
\vspace{.2em}
0.1, & p_f \text{ matches } \text{<rdf:}\_\_\_\_[0-9]+$
1.0, & \text{otherwise .}
\end{cases}
\]

Alternatively, the field and value weight parameters may be estimated based on the portion of query terms covered. Note that this way \( \alpha_f \) and \( \alpha_v \) are set in a query-dependent manner. Query coverage measures the portion of query terms that are contained in the field or value. Additionally, it also considers the importance of terms, based on their IEF value. Formally:

\[
\alpha_x(q) = \frac{\sum_{t \in x \cap q} \text{IEF}(t)^2}{\sum_{t \in q} \text{IEF}(t)^2},
\]

where \( x \) stands for either \( f \) or \( v \).

Another way of setting the value weight parameter is based on the notion of value coverage, which reflects the portion of terms for a given field value that match the query. To compensate for the differences in value lengths, the following formula is used:

\[
\alpha_v(q) = \frac{\alpha}{1 + (\alpha - 1)\left[ \frac{\sum_{t \in v \cap q} c(t; f, v)}{l_v} \right]^B},
\]

where \( \alpha \in (0, 1) \) is a parameter that imposes a fixed lower bound, to prevent short values gaining a benefit over long values, and \( B \) is a parameter that controls the

---

1We use the prefixed version for the rdf namespace; see Table 2.4 for the full URL.
4.2 Preserving Structure

effect of coverage. “The higher \( B \) is, the higher the coverage needs to be for the value node [field value] to receive a weight higher than \( \alpha \)” [14].

On top of field length normalization \( (b_f) \), the BM25MF ranking function offers an additional normalization on the field’s cardinality \( (b_v) \). Based on the experiments in [14], \( b_f \in [0.4,0.7] \) and \( b_v \in [0.5,0.8] \) generally provide good overall performance (although there can be considerable differences across datasets).

4.2.2 References to Entities

Until this point, we have replaced the references to related entities (i.e., URI object values of SPO triples) with terms; specifically, with the primary names of the corresponding entities. This enhances the “findability” of entities by means of keyword queries. At the same time, a large part of the underlying structure (and hence semantics) gets lost in this translation. Consider in particular the issue of entity ambiguity. This has already been resolved in the underlying knowledge base, thanks to the presence of unique identifiers. Replacing those identifiers with the associated names re-introduces ambiguity.

Hasibi et al. [25] propose to preserve these entity references by employing a dual entity representation: On top of the traditional term-based representation, each entity is additionally represented by means of its associated entities. We will refer to the latter as entity-based representation. The idea is illustrated in Fig. 4.2. We will come back to the issue of field selection for building these fielded representations.

Crucially, to be able to make use of the proposed entity-based representation, we need to likewise enrich the term-based query with entity annotations. We shall assume that there is some query annotation mechanism in place that recognizes entities in the query and assigns unique identifiers to them (the dashed arrow in

![Fig. 4.2 Illustration of dual term-based and entity-based entity representations. On the left, predicate-object pairs are shown for a given subject entity (ANN DUNHAM). In the middle are the corresponding term-based and entity-based representations. On the right is the query, which also has a dual representation. Multi-valued fields are indicated using square brackets ([…]). URIs are typeset in monospace font. Matching query terms/entities are highlighted. Figure is based on [25].](image-url)
These query entities may be provided by the user (e.g., by using query auto-completion and clicking on a specific entity in the list of suggestions) or by some automatic entity linking method (see Sect. 7.3). The details of the query annotation process are not our focus here.

Formally, our input keyword++ query is a tuple \( \tilde{q} = (q, E_q) \), where \( q = \langle q_1, \ldots, q_n \rangle \) is the keyword query (sequence of terms) and \( E_q = \{e_1, \ldots, e_m\} \) is a set of entities recognized in the query, referred to as query entities (possibly an empty set). Further, we assume that each of the query entities \( e \in E_q \) has an associated weight \( w(e, q) \), reflecting the confidence in that annotation. These weights play a role when annotations are obtained automatically; if query entities are provided by the user then they would all be assigned \( w(e, q) = 1 \).

There are several possibilities to combine term-based and entity-based representations during matching. Hasibi et al. [25] propose a theoretically sound solution, referred to as ELR (which stands for entity linking incorporated retrieval). It is based on the Markov random field (MRF) model [40], and thus may be applied on top of any term-based retrieval model that can be instantiated in the MRF framework. In our discussion, we will focus on the sequential dependence model (SDM) variant (we refer back to Sect. 3.3.1.3 for details). ELR extends the underlying graph representation of SDM with query entity nodes; see the shaded circles in Fig. 4.3. Notice that query entities are independent of each other. This introduces a new type of clique: 2-cliques between the given entity (that is being scored) and the query entities. Denoting the corresponding feature function as \( f_E(e_i; e) \) and the associated weight as \( \lambda_E \), the MRF ranking function is defined as:

\[
P_A(e|\tilde{q}) \overset{\text{rank}}{=} \lambda_T \sum_{i=1}^{n} f_T(q_i; e) + \lambda_O \sum_{i=1}^{n-1} f_O(q_i, q_{i+1}; e) + \lambda_U \sum_{i=1}^{n-1} f_U(q_i, q_{i+1}; e) + \lambda_E \sum_{i=1}^{m} f_E(e_i; e).
\]

**Fig. 4.3** Graphical representation of the ELR model [25] for a query with three terms and two query entities.
There is, however, a crucial difference between term-based and entity-based matches, which we need to take account of. As explained in Hasibi et al. [25], “the number of cliques for term-based matches is proportional to the length of the query (\(n\) for unigrams and \(n - 1\) for ordered and unordered bigrams), which makes them compatible (directly comparable) with each other, irrespective of the length of the query.” This is why the SDM parameters are outside the summations and can be trained without having to deal with query length normalization. The same cannot be done with \(\lambda_E\), for two reasons. First, the number of query entities varies and is independent of the length of the query (e.g., a short but ambiguous keyword query may have several query entities, while a long natural language query might have only a single one). Second, query entities have different weights (confidence scores) associated with them, and these should be taken into consideration (this is of particular importance when automatic query annotation is used). To overcome the above issues, the \(\lambda\) parameters are rewritten as parameterized functions over each clique:

\[
\begin{align*}
\lambda_T(q_i) &= \lambda_T \frac{1}{n}, \\
\lambda_O(q_i, q_{i+1}) &= \lambda_O \frac{1}{n - 1}, \\
\lambda_U(q_i, q_{i+1}) &= \lambda_U \frac{1}{n - 1}, \\
\lambda_E(e_i) &= \lambda_E \frac{w(e_i, q)}{\sum_{j=1}^{m} w(e_j, q)}.
\end{align*}
\]

Using these parametrized \(\lambda\) functions, and factoring constants out of the summations, the final ranking function becomes:

\[
P_{\Lambda}(e|\hat{q}) \overset{\text{rank}}{=} \frac{\lambda_T}{n} \sum_{i=1}^{n} f_T(q_i; e) \\
+ \frac{\lambda_O}{n - 1} \sum_{i=1}^{n-1} f_O(q_i, q_{i+1}; e) \\
+ \frac{\lambda_U}{n - 1} \sum_{i=1}^{n-1} f_U(q_i, q_{i+1}; e) \\
+ \frac{\lambda_E}{\sum_{j=1}^{m} w(e_j, q)} \sum_{i=1}^{m} w(e_i, q) f_E(e_i; e).
\]

We subject the free parameters to the constraint \(\lambda_T + \lambda_O + \lambda_U + \lambda_E = 1\). Notice that by setting \(\lambda_O\) and \(\lambda_U\) to zero, the above model is an extension of unigram
language models, LM, MLM, and PRMS. Otherwise, it extends SDM and FSDM. The default parameter values in [25] are (1) $\lambda_T = 0.9$ and $\lambda_E = 0.1$ for unigram language models and (2) $\lambda_T = 0.8$, $\lambda_O = 0.05$, $\lambda_U = 0.05$, and $\lambda_E = 0.1$ for sequential dependence models (SDM and FSDM).

The last component of the model that remains to be defined is the feature function $f_E(e_i; e)$. Before that, let us briefly discuss the construction of entity-based representations, as the choices we make there will have an influence on the feature function. Let $\tilde{e}$ denote the entity-based representation of entity $e$. Each unique predicate from the set of SPO triples belonging to $e$ corresponds to a separate field in $\tilde{e}$, and $\tilde{F}$ denotes the set of fields across the entity catalog. Unlike for term-based representations, we do not fold predicates together here. When computing the degree of match between a candidate entity $e$ that is being scored and a query entity $e_i$, we take a weighted combination of field-level scores (for each field in $\tilde{e}$). There are two important differences compared to a traditional term-based representation. Firstly, each entity appears at most once in each field (because of how we mapped SPO triples to fields). Secondly, if a query entity appears in a field, then it shall be regarded as a “perfect match,” independent of what other entities are present in the same field. Using our example from Fig. 4.2, if the query entity $\langle \text{dbp:Barack_Obama} \rangle$ is contained in the $\langle \text{dbo:child} \rangle$ field of the entity ANN DUNHAM, then it is treated as a perfect match (since it should not matter how many other children she has). Driven by the above considerations the feature function is defined as:

$$f_E(e_i; e) = \log \sum_{f \in \tilde{F}} w_f \left( (1 - \lambda) \mathbb{1}(e_i, f_{\tilde{e}}) + \lambda \frac{\sum_{e' \in \tilde{E}} \mathbb{1}(e_i, f_{\tilde{e}'})}{|\{ e' \in \tilde{E} : f_{\tilde{e}'} \neq \emptyset \}|} \right),$$

where the linear interpolation implements the Jelinek-Mercer smoothing method (using $\lambda = 0.1$ in [25]), and $\mathbb{1}(e, f_{\tilde{e}})$ is a binary indicator function, which is 1 if $e_i$ is present in the entity field $f_{\tilde{e}}$ and otherwise 0. The background model part of the interpolation employs a notion of (fielded) entity frequency; the number of entities in the catalog that contain $e_i$ in field $f$ is divided by the total number of entities for which that field is non-empty.

Finally, for setting the field weights $w_f$, we employ dynamic mapping using PRMS, exactly the same way as we did for terms (cf. Eq. (3.13)), but using entity identifier (URI) tokens instead of terms. Setting the field weights in this manner has a number of advantages: (1) there are no additional free parameters to be trained and (2) the importance of fields is chosen dynamically for each query entity (depending on which fields it typically occurs in).

We note that the ELR model we discussed here may be applied to entity types as well (as those are also represented with a URI value as object in SPO triples). However, since types are conceptually different from related entities, they shall receive special treatment in the next section.
4.3 Entity Types

One distinctive characteristic of entities is that they are typed. Each entity in a knowledge base in principle has a type, or multiple types, assigned to it. These types are often organized hierarchically in a type taxonomy. Types may also be referred to as categories, e.g., in an e-commerce context.

In this section, we will assume a keyword++ input query that takes the form $\tilde{q} = (q, T_q)$, where $q$ is the keyword query and $T_q$ is a set of target types (also referred to as query types). Like with query entities in the previous section, these types may be provided by the user (e.g., through the use of faceted search interfaces, see Fig. 4.4) or identified automatically (cf. Sect. 7.2). An abstraction of this very scenario has been studied at the INEX Entity Ranking track [19, 20, 54], where a keyword query is complemented with a small number of target types. There, Wikipedia categories were used as the type taxonomy. An example INEX topic is shown in Listing 4.1.

Having knowledge of the target types of the query, retrieval results may be filtered or re-ranked based on how well they match these target types. There are, however, a number of complicating factors, which we shall elaborate upon below.

4.3.1 Type Taxonomies and Challenges

When the type taxonomy is flat and comprises of only a handful of types (such as in the Airbnb example in Fig. 4.4 (left)), the usage of type information is rather straightforward: A strict type filter may be employed to return only entities of the desired type(s). Many type taxonomies, however, are not like that. They consist of a large number of types that are organized in a multi-layered hierarchy. The type hierarchy is transitive, i.e., if entity $e$ is of type $y$ and $y$ is a subtype of $z$, then $e$ is

![Fig. 4.4 Entity type filters on the SERP in AirBnb (Left) and eBay (Right)]
Table 4.3 Overview of type taxonomies corresponding to different knowledge repositories

|                  | DBpedia\(^a\) | Freebase\(^b\) | Wikipedia\(^c\) | YAGO       |
|------------------|----------------|---------------|----------------|------------|
| #types           | 713            | 1719          | 1,333,352      | 568,672    |
| Depth            | 7              | 2             | N/A            | 19         |
| #top-level types | 22             | 92            | N/A            | 61         |
| #leaf-level types| 561            | 1626          | 662,609        | 549,775    |
| Avg. #types/entity| 2.8           | 4.4           | 4.5            | 12.2       |

\(^a\)Version 2015-10
\(^b\)Latest available dump (2015-03-31)
\(^c\)Corresponding to DBpedia version 2015-10

Listing 4.1 Example topic definition from the INEX 2008 Entity Ranking track. Systems only use the contents of the \(<\text{title}>\) and \(<\text{categories}>\) tags, \(<\text{description}>\) and \(<\text{narrative}>\) are meant for human relevance assessors to clarify the query intent.

also of type \(z\). The root node of the hierarchy is often a generic concept, such as “object” or “thing.” There might be less “well-behaved” type systems (Wikipedia categories being a prime example), where types do not form a well-defined “is-a” hierarchy (i.e., the type taxonomy is a graph, not a tree). Table 4.3 shows the type taxonomies corresponding to four popular large-scale knowledge repositories. Dealing with hierarchical type taxonomies brings a set of challenges, related to the modeling and usage of type information.

Concerning the user’s side, in many cases the user has no extensive knowledge of the underlying type taxonomy. One real-life example is provided in Fig. 4.4 (right), displaying the category filters shown on eBay in response to the query “gps mount.” It is not necessarily obvious which of these categories should be picked (especially since this interface only allows for selecting a single one). The INEX topic example in Listing 4.1 also identifies a very specific (albeit incomplete) set of categories; for other topics, however, the target type(s) may be more wide-ranging. For example, for another INEX query, “Nordic authors who are known for children’s literature,” a single target category “writers” is provided. Thus, the target types provided by the user might be very broad or very narrow.
There are also issues on the data side. Type assignments of entities, particularly in large entity repositories, are imperfect. Types associated with an entity may be incomplete or missing altogether, wrong types may be attributed, and type assignments may be done inconsistently across entities. The type taxonomy may suffer from various coverage and quality problems too. It is often the case that certain branches of the taxonomy are very detailed, while for specific entities there is no matching category other than the root node of the hierarchy, which is overly generic to be useful. The upshot is as follows:

In many application scenarios, which involve a type system (taxonomy) with more than a handful of possible categories, target type information should be treated as “hints” rather than as strict filters.

### 4.3.2 Type-Aware Entity Ranking

We model type-based similarity as a separate retrieval component. The type-aware scoring formula can be written in the form of a linear mixture [3, 32, 45, 47]:

$$
\text{score}(e; \tilde{q}) = \lambda \text{score}_t(e; q) + (1 - \lambda) \text{score}_T(e; T_q),
$$

where the first component, \( \text{score}_t(e; q) \), is the term-based similarity between entity \( e \) and the keyword part of the query. This score may be computed using any of the methods from the previous chapter. The second component, \( \text{score}_T(e; T_q) \), expresses the type-based similarity between the entity and the set of target types \( T_q \). The interpolation parameter \( \lambda \) is chosen empirically.

Alternatively, the type-aware score may also be written as a multiplication of the components:

$$
\text{score}(e; \tilde{q}) = \text{score}_t(e; q) \times \text{score}_T(e; T_q).
$$

Since the relative influence of the individual components cannot be adjusted, this formulation is primarily used for type-based filtering of search results. We shall see an example for this sort of usage in Sect. 4.4.3.

### 4.3.3 Estimating Type-Based Similarity

Let us next consider a number of different ways of establishing the similarity between an entity and a set of target types, that is, estimating \( \text{score}_T(e; T_q) \). Note that relevant entities may not be associated with the provided target types. This is alleviated by leveraging hierarchical relationships of types in the taxonomy (either
explicitly in the scoring formula or by expanding the types of the entity and/or the query). The choice of method to use depends on the particular application scenario and type taxonomy.

**Term-Based Similarity** One simple solution is to join the labels of the corresponding entity types in a separate field and measure the similarity between the labels of target types and this field using any term-based retrieval model; see, e.g., [18]. Considering the example in Listing 4.1, the bag-of-words type query becomes “21st(1) century(1) classical(2) composers(3) finnish(1) living(1),” where the numbers in the superscript denote the query term frequency. The advantage of this method is that a separate type field is often distinguished anyway in the fielded entity description (cf. Table 3.5), thus no extra implementation effort is involved on the entity side. Further, the hierarchical nature of the type taxonomy can be easily exploited by expanding the type query with labels of subcategories, siblings, etc. The disadvantage is that we are limiting ourselves to surface-level matches.

Another variant of this idea is to represent types in terms of their “contents” (not just by their labels). This content-based representation can be obtained by concatenating the descriptions of entities that belong to that type; see, e.g., [32]. A term-based type query is formulated the same way as before, and is scored against the content-based representation.

**Set-Based Similarity** Since both the target types and the types of an entity are sets (of type identifiers), it is natural to consider the similarity of those sets. Pehcevski et al. [45] measure the *ratio of common types* between the set of types associated with an entity ($T_e$) and the set of target types ($T_q$):

$$score_T(e; T_q) = \frac{|T_e \cap T_q|}{|T_q|} .$$

Both the entity and target types may be expanded using ancestor and descendant categories in the type taxonomy.

**Graph-Based Distance** Raviv et al. [47] propose to base the type score on the distance of the entity and query types in the taxonomy:

$$score_T(e; T_q) = e^{-\alpha d(T_q, T_e)} ,$$

where $\theta$ stands for the mathematical constant that is the base of the natural logarithm (not to be confused with entity $e$), $\alpha$ is a decay coefficient (set to 3 in [47]), and $d(T_q, T_e)$ is the distance between the types of the query and the types of entity $e$, computed as follows:

$$d(T_q, T_e) = \begin{cases} 0 , & T_q \cap T_e \neq \emptyset \\ \min \{d_{\max}, \min_{y \in T_q, z \in T_e} d(y, z) \} , & \text{otherwise} \end{cases} .$$

In words, if the query and entity share any types then their distance is taken to be zero; otherwise, their distance is defined to be the minimal path length between all
pairs of query and entity types (denoted as \(d(y, z)\) for types \(y\) and \(z\)). Additionally, there is a threshold on the maximum distance allowed, in case the query and entity types are too far apart in the taxonomy (\(d_{\text{max}}\), set to 5 in [47]).

**Probability Distributions** Balog et al. [3] model type information as probability distributions over types, analogously to the idea of language modeling (which is about representing documents/entities as probability distributions over terms). Let \(\theta_{\mathcal{T}_q}\) and \(\theta_{\mathcal{T}_e}\) be the type models (i.e., probability distributions) corresponding to the query and entity, respectively. The type-based similarity then is measured in terms of the distance between the two distributions:

\[
\text{score}_\tau(e; \mathcal{T}_q) = \max_{e' \in \mathcal{E}} KL(\theta_{\mathcal{T}_q} || \theta_{\mathcal{T}_{e'}}) - KL(\theta_{\mathcal{T}_e} || \theta_{\mathcal{T}_{e'}}).
\]

The distance function employed is the Kullback–Leibler (KL) divergence; the maximum distance is used for turning this into a similarity function.

Type models are represented as multinomial distributions. The probability of a type \(y\) given an entity is estimated analogously to unigram language models employing Dirichlet prior smoothing. Mind that we denote individual types as \(y\) (so as not to be confused with terms):

\[
P(y | \theta_{\mathcal{T}_e}) = \frac{1(y \in \mathcal{T}_e)}{|\mathcal{T}_e| + \mu_{\mathcal{T}}},
\]

where \(1(y \in \mathcal{T}_e)\) takes the value 1 if \(y\) is one of the types assigned to entity \(e\), and otherwise equals to 0. The total number of types assigned to \(e\) is denoted as \(|\mathcal{T}_e|\).

The smoothing parameter \(\mu_{\mathcal{T}}\) is set to the average number of types assigned to an entity across the catalog. Finally, the background (catalog-level) type model is the relative frequency of the type across all entities:

\[
P(y | \mathcal{E}) = \frac{\sum_{e \in \mathcal{E}} 1(y \in \mathcal{T}_e)}{|\mathcal{E}|}.
\]

The types of the query are also modeled as a probability distribution. In the simplest case, we can set it according to the relative type frequency in the query:

\[
P(y | \theta_{\mathcal{T}_q}) = \frac{1(y \in \mathcal{T}_q)}{|\mathcal{T}_q|}.
\]

---

2Since types are binary assignments, the attentive reader might wonder why not use a multivariate Bernoulli distribution instead. That is certainly a possibility, which the reader is invited to explore. General reasons for using a multinomial distribution include that it is simpler to understand, easier to implement, and appears to be more efficient [38]. Our specific reason is that the same statistical distribution is used for modeling the query; when employing query expansion techniques, we need to be able to capture the importance of a given type (as opposed to its mere presence/absence).
Since each type appears at most once in the query, this basically means distributing the probability mass uniformly across the query types. Considering that input type information may be very sparse, it makes sense to enrich the type query model \( \theta T_q \) by (1) considering other types that are relevant to the keyword query, or (2) applying (pseudo) relevance feedback techniques (analogously to the term-based case) [3].

### 4.4 Entity Relationships

Relationships, or “typed links,” are another unique characteristic of entities. Many information needs involve searching for entities based on the relationships between them. Consider, e.g., the queries “teammates of Michael Schumacher,” “wives of Tom Cruise,” or “astronauts who landed on the Moon.” In this section, we discuss approaches for utilizing relationship information for entity retrieval. We look at three particular variants of the entity ranking task. We start with classical ad hoc search using keyword queries (Sect. 4.4.1). Next, we consider list search, where we still use a keyword query, but we have an additional piece of information, namely, that the query seeks a list of entities (Sect. 4.4.2). Finally, we discuss related entity finding, where the input is a keyword++ query, which includes an input entity and a target type (Sect. 4.4.3).

Depending on the task, we may view the knowledge repository as a knowledge graph. There, each entity is a node that is connected to other resources via labeled directed edges (i.e., predicates).

#### 4.4.1 Ad Hoc Entity Retrieval

To begin with, we consider the standard ad hoc entity retrieval task (using conventional keyword queries). However, instead of relying only on term-based ranking, we will additionally exploit the structure of the knowledge graph. Specifically, we present the approach proposed by Tonon et al. [52], where (1) a set of seed entities are identified using term-based entity retrieval, and then (2) edges of these seed entities are traversed in the graph in order to identify potential additional results.

Let \( \mathcal{E}_q(k) \) denote the set of top-\( k \) entities identified using the term-based retrieval method; these are used as seed entities. Let \( \hat{\mathcal{E}}_q \) denote the set of candidate entities that may be reached from the seed entities. Tonon et al. [52] define a number of graph patterns, which are expressed as SPARQL queries. Scope one structured queries look for candidate entities that have direct links to the seed entities, i.e., follow the pattern \( e' \xrightarrow{p} e \), where \( e' \) is a seed entity, \( e \) is a candidate entity, and the

---

3We say resources because in addition to entities and types, the graph may contain other types of URI-nodes as well (e.g., general concepts or disambiguations).
two are connected by predicate $p$. Based on the predicate, four different variations are considered, such that each extends the set of predicates from the one above:

- **Same-as links.** The `<owl:sameAs>` predicate connects identifiers that refer to the same real-world entity.
- **Disambiguations and redirects.** Disambiguations (`<dbo:wikiPageDisambiguates>`) and redirects (`<dbo:wikiPageRedirects>`) to other entities are also incorporated.
- **Properties specific to user queries.** An additional set of predicates that connect seed and candidate entities is selected empirically using a training dataset. These include generic properties, such as Wikipedia links (`<dbp:wikilink>`) and categories (`<dct:subject>`), as well as some more specific predicates, like `<dbo:artist>` or `<dbp:region>`.
- **More general concepts.** On top of the predicates considered by the previous methods, links to more general concepts (predicate `<skos:broader>`) are also included.

An extension of the above approach is to look for candidate entities multiple hops away in the graph. Mind that the number of candidate entities reached potentially grows exponentially with distance from the seed entity. Therefore, only **scope two** queries are used in [52]. These queries search for patterns in the form of $e' \xleftarrow{p_1} x \xrightarrow{p_2} e$, where $x$ can be an entity or a type standing in between the seed entity $e'$ and the candidate entity $e$. The connecting graph edges (predicates $p_1$ and $p_2$) are selected from the most frequent predicate pairs.

According to the results in [52], using the second type of scope one queries (i.e., same-as links plus Wikipedia redirects and disambiguations) and retrieving $k = 3$ seed entities performs best. The resulting set of candidate entities may be further filtered based on a pre-defined set of predicates [52]. For each of the candidate entities $e \in \hat{E}_q$, it is kept track of which seed entity it was reached from. Noting that there may be multiple such seed entities, we let $\mathcal{E}_{e',e}$ denote the set of seed entities that led to $e$.

Finally, retrieval scores are computed according to the following formula:

$$\text{score}(e; q) = \lambda \text{score}_t(e; q) + (1 - \lambda) \sum_{e' \in \mathcal{E}_{e',e}} \text{score}_t(e; e', q).$$

The first component is the term-based retrieval score of the entity. The second component is the sum of the retrieval scores of the seed entities $e'$ that led to entity $e$. The interpolation parameter $\lambda$ is set to 0.5 in [52].
4.4.2 List Search

Next, we consider a specific flavor of ad hoc entity retrieval, where the user is seeking a list of entities. That is, we are supplied with an extra bit of information concerning the intent of the query. We are not concerned with how this knowledge is obtained; it could be an automatic query intent classifier or it could be the user indicating it somehow. This scenario was addressed by the list search task of the Semantic Search Challenge in 2011 [10] (cf. Sect. 3.5.2.4). Examples of such queries include “Apollo astronauts who walked on the Moon” and “Arab states of the Persian Gulf.” Notice that queries are still regular keyword queries, which means that existing term-based entity ranking approaches are applicable (which is indeed what most challenge participants did). Below, we discuss a tailored solution that performs substantially better than conventional entity ranking methods.

The SemSets model by Ciglan et al. [15] is a three-component retrieval model, where entities are ranked according to:

\[
score(e; q) = \text{score}_C(e; q) \times \text{score}_S(e; q) \times \text{score}_P(e; q).
\]

(4.1)

We detail each of the three score components below.

**Candidate Entity Score** To identify candidate entities that are possible answers to the query, the process starts with a standard (term-based) entity ranking step (using any model of choice from Sect. 3.3). Let \( E_q(k) \) denote the set of top-\( k \) ranked entities and let \( \text{rank}(e, q) \in [0..k-1] \) indicate the rank position of these entities (lower rank means higher relevance). The “base” entity score is set inversely proportional to the rank position:

\[
score_B(e; q) = \begin{cases} 1 - \frac{\text{rank}(e, q)}{k}, & e \in E_q(k) \\ 0, & \text{otherwise} \end{cases}
\]

The base scores are then propagated in the knowledge graph, following the principle of the activation spreading. Ciglan et al. [15] restrict the spreading to only one hop from the given vertices (and use \( k = 12 \) base entities). Accordingly, the candidate score becomes:

\[
score_C(e; q) = score_B(e; q) + \sum_{(s, p, o) \in K \atop s=e', o=e} score_B(e'; q) .
\]

Thus, each entity receives, in addition to its base score, the sum of base scores of all entities that link to it. Optionally, spreading may be restricted to a selected subset of predicates. The candidate set comprises of entities with a non-zero candidate score: \( E_C = \{ e : score_C(e; q) > 0 \} \). Entities outside this set would receive a final score of zero because of the multiplication of score components in Eq. (4.1), and therefore are not considered further.
**Semantic Set Score** A key idea in this model is the concept of *semantic sets* (SemSets): “sets of semantically related entities from the underlying knowledge base” [15]. Intuitively, members of a music band or companies headquartered in the same city would constitute SemSets. We shall first explain how SemSets are used for scoring entities, and then detail how they can be obtained.

The SemSets score of an entity is calculated as a sum of the relevance scores of all SemSets it belongs to:

$$score_S(e; q) = 1 + b \left( \sum_{S \in S_q} \sum_{e \in S} score(S; q) \right),$$

where $S_q$ are the candidate semantic sets for the query and $b$ is a boost parameter (set to 100 in [15]). For a given SemSet $S$, the relevance score $score(S; q)$ is established based on its similarity to the query. A term-based representation of the set is built by concatenating the descriptions of all entities that belong to that set: $S_t = \bigcup_{e \in S} et$, where $et$ is the term-based representation of $e$. This representation can then be scored against the query using any standard text retrieval model.

The selection of candidate semantic sets is based on their overlap with the set of candidate entities. That is, a certain fraction of the SemSet’s member entities must also be in the candidate entity set identified for the query. Denoting the set of all possible SemSets as $S$, the set $S_q$ of candidate SemSets for the query is given by:

$$S_q = \{ S \in S : \frac{|S \cap E_C|}{|S|} \geq \gamma \},$$

where $\gamma$ is a threshold parameter (set to 0.7 in [15]).

The construction of possible SemSets $S$ is governed by two graph patterns, illustrated in Fig. 4.5:

(a) Vertices (entities) with outgoing edges, labeled with the same predicate, to the same object (or, in terms of SPO triples: $(\star, p, o)$). For instance, Wikipedia categories are examples of sets of this type, i.e., the predicate is `<dct:subject>` and the object is a given category (e.g., `<dbc:People_who_have_walked_on_the_Moon>`).

(b) Vertices (entities) with incoming edges, labeled with the same predicate, to the same object. For instance, pages with the same label in a Wikipedia category are examples of sets of this type, i.e., the predicate is `<dct:subject>` and the object is a given category (e.g., `<dbc:People_who_have_walked_on_the_Moon>`).
(b) Vertices (entities) with incoming edges, labeled with the same predicate, from
the same subject (SPO pattern \((s, p, \star)\)). For example, members of some music
band (predicate \(<\text{dbo:bandMember}>\)) constitute a semantic set of this type.

A problem with the above constructions is that the number of possible SemSets is
huge and becomes impractical to handle. Arguably, not all types of edges (predi-
cates) are equally useful. Thus, set formation may be restricted to specific predi-
cates. It is shown in [15] that using only two types of predicates, Wikipedia category
memberships (\(<\text{dct:subject}>\)) and Wikipedia templates (\(<\text{dbp:wikiPageUses-
Template}>\)), provides solid performance. This is not surprising, since Wikipedia
categories and templates are complementary, manually curated lists of semantically
related entities (cf. Sect. 2.2). Alternatively, sets may be filtered automatically, using
graph structural measures [15].

**Principal Entity Relatedness** The third score component considers the distance
of a given entity from the principal entity of the query. Entities in the query are
recognized using a dictionary-based approach (cf. Sect. 7.3). Then, the entity with
the highest confidence score is selected as the principal entity of the query. (It might
also happen that no entities are recognized in the query, in which case this score
component is ignored.) The principal entity relatedness score is defined as:

\[
score_P(e; q) = 1 + c \times \text{sim}(e, e_q),
\]

where \(e_q\) denotes the principal entity of the query, \(c\) is a boost parameter (set to 100
in [15]), and \(\text{sim}(e, e')\) is a graph structural similarity measure. Specifically, entities
are represented in terms of their neighbors in the knowledge graph and the cosine
similarity of the corresponding vectors is computed. Other possible measures of
pairwise entity similarity will be discussed in Sect. 4.5.1.

### 4.4.3 Related Entity Finding

Taking the previous task one step further, one may explicitly target a class of queries
that mention a focal entity and seek for entities that are related to that entity.
The TREC Entity track in 2009 introduced the related entity finding (REF) task
as follows: “Given an input entity, by its name and homepage, the type of the
target entity, as well as the nature of their relation, described in free text, find
related entities that are of target type, standing in the required relation to the input
entity” [7].

Formally, the input keyword++ query is \(\tilde{q} = (q, e_q, y_q)\), where \(q\) is the keyword
query (describing the relation), \(e_q\) is the input entity, and \(y_q\) is the target type.
An example REF query is shown in Listing 4.2. There are several possibilities for
defining the input entity and target type. At the first edition of the TREC Entity
track, homepages were used as entity identifiers and the target type could be either
person, organization, or product. Later editions of the track also experimented,
among other things, with using a Linked Data crawl for entity identification and lifting the restrictions on target types \cite{5,6}. Here, we will consider a simplified version of the task assuming that (1) entities are equipped with unique identifiers and come from a given catalog (knowledge repository) and (2) the target type is from a handful of possible coarse-grained categories, such as person, organization, product, or location.

Even though we use a knowledge repository for identifying entities, that repository, as a single source of data, is often insufficient for answering entity relationship queries. For the ease of argument, let us assume that the said repository is a general-purpose knowledge base (like DBpedia or Freebase). First, the number of distinct predicates in a KB is very small compared to the wide range of possible relationships between entities. For instance, the “[airline] uses [aircraft]” relationship from our topic example is not recognized in DBpedia. Second, even if the given relationship is recognized in the KB, the KB may be incomplete with regards to a specific entity. Therefore, for the REF task, we will complement the knowledge base with a large unstructured data collection: a web corpus. We shall assume that this web corpus has been annotated with entity identifiers from the underlying entity catalog.

Commonly, the REF task is tackled using a pipeline of three steps: (1) identifying candidate entities, (2) filtering entities that are of incorrect type, and (3) computing the relevance of the (remaining) candidates with respect to the input entity and relation. This pipeline is shown in Fig. 4.6.

Bron et al. \cite{12} address the REF task using a generative probabilistic model. Entities are ranked according to the probability $P(e|q,e_q,y_q)$ of entity $e$ being relevant. Using probability algebra (Bayes’ theorem) and making certain independence assumptions, the following ranking formula is derived:

$$P(e|q,e_q,y_q) \overset{\text{rank}}{=} P(e|e_q)P(y_q|e)P(q|e,e_q), \quad (4.2)$$
where $P(e|e_q)$, $P(y_q|e)$, and $P(q|e, e_q)$ correspond to the candidate selection, type filtering, and entity relevance steps of the pipeline, respectively. The graphical representation of the model is shown in Fig. 4.7. We note that this is only one of the many possible ways to go about modeling this task. Nevertheless, the components that make up the scoring formula in Eq. (4.2) are rather typical.

### 4.4.3.1 Candidate Selection

The first step of the pipeline is concerned with the identification of candidate entities. At this stage the focus is on achieving high recall, in order to capture all entities that are possible answers to the query. In Bron et al. [12], this is done through a so-called co-occurrence model, $P(e|e_q)$, which reflects the degree of association between a candidate entity $e$ and the input entity $e_q$. Let $a(e, e_q)$ be a function (to be defined) that expresses the strength of association between a pair of entities. The co-occurrence probability is then estimated according to:

$$P(e|e_q) = \frac{a(e, e_q)}{\sum_{e' \in E} a(e', e_q)}.$$  

In order to arrive at a reliable estimate, the function $a(e, e_q)$ is based on co-occurrence statistics of the two entities in a large web corpus. Let $c(e)$ denote the number of documents in which $e$ occurs and let $c(e, e_q)$ denote the number of documents in which $e$ and $e_q$ co-occur.

There are many possible ways to set the association function. One of the simplest options is to compute the maximum likelihood estimate:

$$a_{MLE}(e, e_q) = \frac{c(e, e_q)}{c(e)}.$$  

Another alternative, which was reported to perform best empirically in [12], is using the $\chi^2$ hypothesis test (to determine “if the co-occurrence of two entities is more likely than just by chance” [12]):

$$a_{\chi^2}(e, e_q) = \frac{|\mathcal{D}|(c(e, e_q)c(\bar{e}, \bar{q}_e) - c(e, \bar{q}_e)c(\bar{e}, e_q))^2}{c(e)c(e_q)(|\mathcal{D}| - c(e))(|\mathcal{D}| - c(e_q))}.$$
4.4 Entity Relationships

where $|\mathcal{D}|$ is the total number of documents in the collection, and $e$ and $e_q$ do not occur.

Rather than relying on the mere co-occurrence of two entities in documents, one might want to consider “stronger evidence.” Requiring that the two entities cross-link to each other constitutes one particular solution. Specifically, the anchor-based co-occurrence method in [12] takes account of how many times one entity appears in the anchor text of the other entity’s description (e.g., the Wikipedia page of that entity). The co-occurrence probability in this case is estimated as:

$$P(e|e_q) = \frac{1}{2} \frac{c(e,e_q)}{\sum_{e' \in \mathcal{E}} c(e',e_q)} + \frac{1}{2} \frac{c(e_q,e)}{\sum_{e' \in \mathcal{E}} c(e',e)} ,$$

where $c(e,e_q)$ is the number of times entity $e$ occurs in the anchor text in the description of $e_q$. For the sake of simplicity, both linking directions are taken into consideration with the same weight.

At the end of the candidate selection step, entities with a non-zero $P(e|e_q)$ value are considered for downstream processing. Commonly, this set is further restricted to the top-$k$ entities with the highest probability.

4.4.3.2 Type Filtering

Earlier, in Sect. 4.3, we have discussed type-aware entity retrieval and have presented various ways of comparing target types of the query with types of entities. The very same methods can be used here as well, for estimating the type component, $P(y_q|e)$, which expresses the probability that entity $e$ is of type $y_q$. In our earlier scenario, however, it was assumed that target types from the corresponding taxonomy are provided explicitly. Here, the target types are only given as coarse-grained categories (such as person, organization, product, or location). Selecting the appropriate types from the type taxonomy is part of the task.

One strategy for dealing with this is to establish a mapping from each possible coarse-grained input type to multiple categories in the type taxonomy. Such mapping may be constructed using a handful of simple rules. For example, using Wikipedia categories as the underlying type taxonomy, Kaptein et al. [33] map the person target type to categories that (1) start with “People,” (2) end with “births” or “deaths,” and (3) the category “Living People.” This initial mapping may be further expanded by adding descendant types until a certain depth according to the type taxonomy [12].

Another strategy is to infer target types from the keyword query directly [34]. We will detail these methods in Sect. 7.2.
4.4.3.3 Entity Relevance

The last part of the pipeline is responsible for determining the relevance of entities. In Bron et al. [12], it is expressed as \( P(q|e,e_q) \), the likelihood that the relation contained in the keyword query is “observable” in the context of an input and candidate entity pair. This context is represented as the entity co-occurrence language model, \( \theta_{e,e_q} \). The query is scored against this model by taking the product over individual query terms:

\[
P(q|\theta_{e,e_q}) = \prod_{t \in q} P(t|\theta_{e,e_q})^{c(t;q)} .
\]

The probability of a term given the entity co-occurrence language model is estimated using the following aggregation:

\[
P(t|\theta_{e,e_q}) = \frac{1}{|D_{e,e_q}|} \sum_{d \in D_{e,e_q}} P(t|\theta_d) ,
\]

where \( D_{e,e_q} \) is the set of documents, or document snippets, in which \( e \) and \( e_q \) co-occur, and \( \theta_d \) is the (smoothed) language model corresponding to document \( d \).

As a matter of choice, the conditional dependence of the query on the input entity may be dropped when computing entity relevance, like it is done in [22]. Thus, \( P(q|e,e_q) \cong P(q|e) \). Then, the query likelihood \( P(q|e) \) may be estimated using entity language models, which we have already discussed in Chap. 3 (cf. Sect. 3.3.1.1).

4.5 Similar Entity Search

Similar entity search is the task of ranking entities given a small set of example entities. One might imagine a specialized search interface that allows the user to explicitly provide a set of example entities. A more practical scenario is to construct the set of entities via implicit user interactions, by taking the entities that the user viewed or “dwelled on” during the course of her search session. These examples may complement the keyword query, which we shall call example-augmented search, or serve on their own as the expression of the user’s information need, which will be referred to as example-based search. The keyword++ query may be written as \( \tilde{q} = (q, E_q) \), where \( E_q \) is a set of example entities. Note that \( q \) may be empty.

Example-augmented search has been studied at the INEX 2007–2009 Entity Ranking track [19, 20, 54] under the name list completion. Listing 4.3 shows an example INEX list completion topic. Example-based search (also known as set expansion) is considerably harder than example-augmented search, because of the inherent ambiguity. In the lack of a keyword query, there are typically multiple
4.5 Similar Entity Search

Listing 4.3 Example topic definition from the INEX 2009 Entity Ranking track. Systems only use the contents of the <title> and <entities> tags, <description> and <narrative> are meant for human relevance assessors to clarify the query intent.

possible interpretations. Take, for instance, the following three example entities: CANON, SONY, and NIKON. The underlying information need may be “camera brands” or “multinational corporations headquartered in Japan” or something else. Domain-specific applications of example-based search include, for instance, finding people with expertise similar to that of others within an organization [4, 28]. A more generic application area is concept expansion for knowledge base population (cf. Chap. 6).

As we have already grown accustomed to it in this chapter, we will employ a two-component mixture model:

$$\text{score}(e; \tilde{q}) = \lambda \text{score}_t(e; q) + (1 - \lambda) \text{score}_E(e; \mathcal{E}_q),$$

where $\text{score}_t(e; q)$ is the standard text-based retrieval score and $\text{score}_E(e; \mathcal{E}_q)$ is the example-based similarity. This general formula can encompass both flavors of similar entity search. In the case of example-based search, where the input comprises only of $\mathcal{E}_q$, the interpolation parameter $\lambda$ is set to 0. For example-augmented search, $0 \leq \lambda \leq 1$, with the exact value typically set empirically. Bron et al. [13] adjust $\lambda$ on a per-query basis, depending on which of the text-based and example-based score components is more effective in retrieving the example entities.

Our main focus in this section is on estimating the example-based similarity component, $\text{score}_E(e; \mathcal{E}_q)$. Before we delve in, a quick note on terminology. To make the distinction clear, we will refer to the entity $e$ that is being scored as the candidate entity, and the set $\mathcal{E}_q$ of examples complementing the keyword query as example entities (or seed entities).

Next, we discuss two families of approaches. Pairwise methods consider the similarity between the candidate entity and each of the example entities (Sect. 4.5.1).
Collective methods, on the other hand, treat the entire set of examples as a whole (Sect. 4.5.2).

### 4.5.1 Pairwise Entity Similarity

A simple and intuitive method is to take the average pairwise similarity between the candidate entity \( e \) and each of the example entities \( e' \in \mathcal{E}_q \):

\[
\text{score}_{\mathcal{E}}(e; \mathcal{E}_q) = \frac{1}{|\mathcal{E}_q|} \sum_{e' \in \mathcal{E}_q} \text{sim}(e, e').
\]

This approach is rather universal, as it boils down to the pairwise entity similarity function, \( \text{sim}() \). This similarity is of fundamental importance, which extends well beyond this specific task. The choice of the entity similarity measure is closely tied to, and constrained by, the entity representation used. We discuss a range of options below, organized by the type of entity representation employed. Note that the combination of multiple similarity measures is also possible.

#### 4.5.1.1 Term-Based Similarity

Perhaps the most conventional method is to compare term-based representations of entities (i.e., entity descriptions, cf. Chap. 3). Sometimes, this is referred to as topical similarity. Let \( e \) denote the term vector corresponding to entity \( e \):

\[
e = \langle w(t_1, e), \ldots, w(t_m, e) \rangle,
\]

where \( m \) is the size of the vocabulary, \( t_1 \ldots t_m \) are the distinct terms in the vocabulary, and \( w(t_j, e) \) is the weighted term frequency of \( t_j \). Typically, a TF-IDF weighting scheme is used. A standard way of comparing two term vectors is using the cosine similarity:

\[
\text{sim}_{\text{cos}}(e, e') = \frac{e \cdot e'}{\|e\| \|e'\|} = \frac{\sum_{i=1}^{m} w(t_i, e) w(t_i, e')}{\sqrt{\sum_{i=1}^{m} w(t_i, e)^2} \sqrt{\sum_{i=1}^{m} w(t_i, e')^2}}.
\]

Instead of using individual terms (unigrams), the vector representation may also be made up of n-grams or keyphrases [27, 28]. Specifically, Hoffart et al. [27] introduce the keyphrase overlap relatedness (KORE) measure, and present approximation techniques, based on min-hash sketches and locality-sensitive hashing, for efficient computation.
4.5 Similar Entity Search

4.5.1.2 Corpus-Based Similarity

Entity similarity may be established based on co-occurrence statistics in some corpus of data. This corpus may be a collection of documents, in which case entities are represented by the set of documents mentioning them. Letting $D_e$ and $D_{e'}$ denote the set of documents in which entities $e$ and $e'$ occur, respectively, the similarity of the two document sets can be measured, e.g., using the Jaccard coefficient:

$$\text{sim}_{\text{Jac}}(e, e') = \frac{|D_e \cap D_{e'}|}{|D_e \cup D_{e'}|}.$$ 

Other co-occurrence-based similarity functions include the maximum likelihood estimate and the $\chi^2$ hypothesis test, which we have already discussed in Sect. 4.4.3.1.

It is also possible to consider co-occurrences on the sub-document level, e.g., in lists [26, 48] or tables [55]. As another option, a corpus of query log data may also be utilized for the same purpose [26].

4.5.1.3 Distributional Similarity

The conventional term-based representation (also called one-hot representation) allows only for exact word matches. Consider the following oversimplified example, for the sake of illustration. Let us assume that entity $A$ has a single term in its representations, “apple,” and entity $B$ also has a single term in its representation, “orange.” These two entities would have a similarity of 0 according to any standard term-based similarity measure (like the Jaccard coefficient or cosine similarity). Yet, arguably, the similarity of “apple” to “orange” should be higher than, say, that of “apple” to “chess.” With the traditional term-based representation, it is not possible to make this distinction. This, in fact, is one of the fundamental challenges in IR, known as the vocabulary mismatch problem. The overall idea behind distributed representations (or distributional semantics) is to represent each word as a “pattern.” Words are embedded into a continuous vector space such that semantically related words are close to each other.

Word vector representations of terms, a.k.a. word embeddings, are obtained from large text corpora using neural networks. The embedding (or latent factor) vector space has low dimensionality, typically in the order of 200–500. At the time of writing, the two dominating methods for computing word vector representations are

---

4Distributional representations and distributed representations are both used, and in this case, both are correct. The former has to do with the linguistic aspect, “meaning is context,” i.e., items with similar distributions (i.e., context words) have similar meanings. The latter refers to the idea of having a compact, dense, and low dimensional representation; a single component of a vector representation does not have a meaning on its own, i.e., it is distributed among multiple vector components. Word embeddings have both these properties.
Word2vec [41] and Glove [46]. Both these implementations are publicly available, and may be run on any text corpus. A variety of pre-trained word vectors are also available for download; these may be used off-the-shelf.\footnote{https://code.google.com/archive/p/word2vec/.} \footnote{http://nlp.stanford.edu/projects/glove/.

Let us now see how these word embeddings can be used for entities. One way to compute the distributed representation of an entity is to take the weighted sum of distributed word vectors for each term in the entity’s term-based representation. Variants of this idea may be found in recent literature, see, e.g., [49, 53, 56]. Formally, let \( n \) be the dimensionality of the embedding vector space. We write \( \vec{e} = (\vec{e}_1, \ldots, \vec{e}_n) \) to denote the distributed representation of entity \( e \)—this is what we wish to compute. Let \( T \) be an \( m \times n \) dimensional matrix, where \( m \) is the size of the vocabulary; row \( j \) of the matrix corresponds to the distributed vector representation of term \( t_j \) (i.e., is an \( n \)-dimensional vector). The distributed representation of entity \( e \) can then be computed simply as:

\[
\vec{e} = e^T,
\]

where \( e \) denotes the term-based entity representation, cf. Eq. (4.3). The above equation may be expressed on the element level as:

\[
\vec{e}_i = \sum_{j=1}^{m} e_j T_{ji}.
\]

One way to look at this transformation is that an \( m \)-dimensional representation is compressed into an \( n \)-dimensional one (where \( m \gg n \)), in such a way that entities that are more similar become closer in this lower dimensional space. The similarity of two entities in the embedding vector space is computed using \( \text{sim}_{\text{cos}}(\vec{e}, \vec{e}') \).

### 4.5.1.4 Graph-Based Similarity

Viewing entities as nodes in a knowledge graph gives rise to another family of similarity functions. One way to establish similarity between two entities is based on the set of other nodes that they are connected to, the idea being that “two objects are similar if they are related to similar objects” [30]. Let \( \mathcal{L}_e \) denote the set of nodes connected to \( e \), where connectedness may be interpreted as (1) incoming links (nodes that link to \( e \)), (2) outgoing links (nodes linked by \( e \)), or (3) the union of incoming and outgoing links of \( e \). For entity \( e' \), \( \mathcal{L}_{e'} \) is defined analogously. Then, it is possible to measure the similarity of the two link sets, \( \mathcal{L}_e \) and \( \mathcal{L}_{e'} \), e.g., using the Jaccard coefficient [26]. The Wikipedia link-based measure (WLM) [42] gives
another quantification of semantic relatedness of two entities, based on the same idea of the overlap between links; see Eq. (5.4) in Sect. 5.6.1.3 for details.

Instead of using only the direct neighbors of entities, like the above measures do, one might consider their distance in the knowledge graph. One way of establishing similarity is by setting it inversely proportional to the minimum (weighted) distance of the two entities [50]. Alternatively, the problem may be approached as that of propagating similarity from an entity node through graph edges, using some variant of graph walk [30, 43]. Graph edge weights can be set uniformly, manually, or automatically (using some weighting function [50] or a learning procedure [43]).

### 4.5.1.5 Property-Specific Similarity

In addition to the above methods, which are generally applicable, similarity measures may be tailored to individual properties. This may be imagined as having a distinct entity representation corresponding to each particular property, such as entity name or type. These property-level similarities are then combined, e.g., as:

\[
sim(e, e') = \sum_{i} \lambda_i \cdot \sim_i(e, e')
\]

where \( \sim_i(e, e') \) is a similarity function for property \( i \) and \( \lambda_i \) is the corresponding weight (importance of that property). Weights may be set manually (based on domain knowledge) or learned from training data. We note that any of the similarity functions from above may be used as \( \sim_i(e, e') \).

A distinguished property is entity type (or category), which naturally provides a grouping of similar entities; here, similarity is understood in an ontological sense. We refer back to Sect. 4.3.3 for various ways of establishing type similarity (where \( T_q \) is to be replaced with \( T_{e'} \)). Note that type-based similarity is only effective in telling apart entities that are of a different kind, e.g., people vs. products. It needs to be combined with other similarity measures when the two entities belong to the same semantic category (e.g., racing drivers). Effectiveness further depends on the granularity of type information, i.e., how detailed the type taxonomy is.

A range of options exists for particular domains or applications. For example, Balog [2] introduces similarity measures for specific product attributes. Product names are compared using various string distance measures, both character-based (e.g., Levenshtein or Jaro-Winkler distance) and term-based (e.g., Jaccard or Dice coefficient). For product prices, the relative value difference is used. Another example, for geospatial entities, is given in [51], for matching the location coordinates of entities.
4.5.2 Collective Entity Similarity

We now switch to collective methods, which consider the set of example entities as a whole. One simple solution, which works for some but not all kinds of entity representations, is to take the centroid of example entities, and compare it against the candidate entity (using a similarity function corresponding to that kind of representation). Here, we introduce two more advanced methods, which make explicit use of the structure associated with entities.

4.5.2.1 Structure-Based Method

Bron et al. [13] employ a structured entity representation, which is comprised of the set of properties of the entity. Given an entity \( e \), and the set of SPO triples that contain \( e \), each triple yields a property \( r \) by removing the entity in question from that triple. (If the data is viewed as a knowledge graph, these properties correspond to the nodes adjacent to the entity, along with the connecting edges.) Formally, the structured representation of a given entity \( e \) is defined as:

\[
\tilde{e} = \{ (r = (p,o) : (e,p,o) \in K) \cup \{ r = (s,p) : (s,p,e) \in K \} \}.
\]

For example, given the entity Michael Schumacher, with a set of RDF triples from Sect. 2.3.1.2, the structured representation of this entity becomes:

\[
\tilde{e} = \{ (<\text{foaf:name}>, "Michael Schumacher"),
\quad (<\text{dbo:birthPlace}>, <\text{dbr:West_Germany}>)
\quad (<\text{dbr:1996_Spanish_Grand_Prix}>, <\text{dbp:firstDriver}>)
\quad \ldots \}.
\]

Under this representation, the set of example entities \( \mathcal{E}_q \) becomes a set of set of properties: \( \tilde{\mathcal{E}}_q = \{ \tilde{e}_1, \ldots, \tilde{e}_{|\mathcal{E}_q|} \} \). The similarity between the entity \( e \) and the set of examples is estimated by marginalizing over all properties of the entity and considering whether they are observed with the examples:

\[
\text{score}_e(e; \mathcal{E}_q) = P(\tilde{e} | \tilde{\mathcal{E}}_q) = \sum_{r \in \tilde{e}} P(r | \theta_{\mathcal{E}_q}) = \sum_{r \in \tilde{e}} \frac{\sum_{i=1}^{|\mathcal{E}_q|} \mathbb{1}(r, \tilde{e}_i)}{\sum_{i=1}^{|\mathcal{E}_q|} \sum_{r \in \tilde{e}_i} \mathbb{1}(r, \tilde{e}_i)},
\]

where \( \mathbb{1}(r, \tilde{e}_i) \) is a binary indicator function which is 1 if \( r \) occurs in the representation of \( \tilde{e}_i \) and is 0 otherwise. The denominator is the representation length of the seed entities, i.e., the total number of relations of all seed entities.
4.5.2.2 Aspect-Based Method

In their approach called QBEES, Metzger et al. [39] aim to explicitly capture the different potential user interests behind the provided examples. Each entity $e$ is characterized based on three kinds of aspects:

- **Type aspects**, $A_T(e)$, include the set of types that the entity is an instance of, e.g.,
  $\{<\text{dbo:Person}>, <\text{dbo:RacingDriver}>, <\text{dbo:FormulaOneRacer}>, \ldots\}$.

- **Relational aspects**, $A_R(e)$, are the predicates (edge labels) connecting the entity to other non-type nodes (irrespective of the direction of the edge), e.g.,
  $\{<\text{dbo:birthPlace}>, <\text{dbp:firstDriver}>, <\text{dbo:championships}>, \ldots\}$.

- **Factual aspects**, $A_F(e)$, are relationships to other entities, i.e., other entity nodes incident to the entity, along with the connecting edges, e.g.,
  $\{(<\text{dbo:birthPlace}>, <\text{dbr:West_Germany}>>, (<\text{dbr:1996_Spanish_Grand_Prix}>, <\text{dbp:firstDriver}>>), \ldots\}$.

The **basic aspects** of an entity is the union of the above three kinds of aspects:

$A(e) = A_T(e) \cup A_R(e) \cup A_F(e)$. A **compound aspect** of an entity is any subset of its basic aspects: $A_e \subseteq A(e)$.

For each basic aspect $a$, the **entity set of an aspect**, $E_a$, is a set of entities that have this aspect: $E_a = \{e \in \mathcal{E} : a \in A(e)\}$. This concept can easily be extended to compound aspects. Let $E_A$ be the set of entities that share **all basic aspects in $A$**:

$E_A = \{e \in \mathcal{E} : a \in A(e) \forall a \in A\}$.

A compound aspect $A$ is a **maximal aspect of an entity $e$** iff

1. $E_A$ contains at least one entity other than $e$;
2. $A$ is maximal w.r.t. inclusion, i.e., extending this set with any other basic aspect of $e$ would violate the first condition.

We write $\mathcal{M}(e)$ to denote the family (i.e., collection) of all maximal aspects of $e$.

Next, we extend all the above concepts from a single entity to the set of example entities $\mathcal{E}_q$. Let $A(\mathcal{E}_q)$ denote the set of basic aspects that are shared by all example entities: $A(\mathcal{E}_q) = \bigcap_{e \in \mathcal{E}_q} A(e)$. A compound aspect $A$ is said to be a **maximal aspect of the set of example entities $\mathcal{E}_q$** iff

1. $E_A$ contains at least one entity that is not in $\mathcal{E}_q$;
2. $A$ is maximal w.r.t. inclusion.

The family of all maximal aspects of $\mathcal{E}_q$ is denoted as $\mathcal{M}(\mathcal{E}_q)$.

The ranking of entities is based on the fundamental observation that for a given set of example entities $\mathcal{E}_q$, the most similar entities will be found in entity sets of maximal aspects of $\mathcal{E}_q$. One of the main challenges in similar entity search, i.e., in computing $\text{score}_e(e; \mathcal{E}_q)$, is the inherent ambiguity because of the lack of an explicit user query. Such situations are typically addressed using diversity-aware...
Algorithm 4.1: QBEES [39]

| Line | Description |
|------|-------------|
| 1    | $E_R \leftarrow \emptyset$ |
| 2    | $\mathcal{M}(E_q) \leftarrow$ maximal aspects of $E_q$ |
| 3    | Filter $\mathcal{M}(E_q)$ by type constraints |
| 4    | Rank $\mathcal{M}(E_q)$ |
| 5    | while $|E_R| < k$ and $|\mathcal{M}(E_q)| > 0$ do |
| 6    | $A^* \leftarrow$ top ranked aspect ($A^* \in \mathcal{M}(E_q)$) |
| 7    | $E^* \leftarrow \{e \in E : e \in E_{A^*}, e \not\in E_q \cup E_R\}$ |
| 8    | $e^* \leftarrow \arg \max_{e \in E^*} \text{score}(e)$ |
| 9    | $E_R \leftarrow E_R \cup \{e^*\}$ |
| 10   | update ranking of $\mathcal{M}(E_q)$ |
| 11   | end |

approaches. That is, constructing the result set in a way that it covers multiple possible interpretations of the user’s underlying information need.

In QBEES, these possible interpretations are covered by the compound aspects. Further, the concept of maximal aspect is designed such that it naturally provides diversity-awareness. This is articulated in the following theorem.

**Theorem 4.1** Let $A_1(E_q)$ and $A_2(E_q)$ be two different maximal aspects of $E_q$. In this case, $A_1(E_q)$ and $A_2(E_q)$ do not share any entities, except those in $E_q$: $A_1(E_q) \cap A_2(E_q) \setminus E_q = \emptyset$.

We omit the proof here and refer to [39] for details. According to the above theorem, the partitioning provided by the maximal aspects of $E_q$ can guide the process of selecting a diverse set of result entities $E_R$ that have the highest similarity to $E_q$.

We detail the key steps of the procedure used for selecting top-$k$ similar entities, shown in Algorithm 4.1.

- **Finding the maximal aspects of the example entities.** From the aspects shared by all example entities, $\mathcal{A}(E_q)$, the corresponding family of maximal aspects, $\mathcal{M}(E_q)$, is computed. For that, all aspects that are subsets of $\mathcal{A}(E_q)$ need to be considered and checked for the two criteria defining a maximal aspect. For further details on how to efficiently perform this, we refer to [39].

- **Constraining maximal aspects on typical types.** It can reasonably be assumed that the set of example entities is to be completed with entities of similar types. Therefore, the set of typical types of the example entities, $\mathcal{T}_{E_q}$. To determine $\mathcal{T}_{E_q}$, first, the types that are shared by all example entities are identified (according to a predefined type granularity). Then, only the most specific types in this set are kept (by filtering out all types that are super-types of another type in the set). Maximal aspects that do not contain at least one of the typical types (or their subtypes) as a basic aspect are removed.

- **Ranking maximal aspects.** “The resulting maximal aspects are of different specificity and thus quality” [39]. Therefore, a ranking needs to be established,
and maximal aspects will be considered in that order. Metzger et al. [39] discuss various ranking functions. We present one particular ranker \((cost)\), which combines the (normalized) “worth” of aspects with their specificity:

\[
cost(A) = \frac{1}{E_A} \times \frac{\text{val}(A)}{\sum_{A' \in \mathcal{MA}(E_q)} \text{val}(A')}.
\]

Recall that \(E_A\) is the set of entities that share all basic aspects in \(A\). Thus, the first term expresses the specificity of an aspect (the larger the set of entities \(E_A\), the less specific aspect \(A\) is). The value of a compound aspect is estimated as a sum of the value of its basic aspects. The value of a basic aspect is its “inverse selectivity,” i.e., aspects with many entities are preferred:

\[
\text{val}(A) = \sum_{a \in A} \left(1 - \frac{1}{E_a}\right).
\]

• **Selecting an entity.** An entity is picked from the top ranked aspect’s entity set. That is, if \(A^*\) is the top ranked aspect, then an entity is selected from \(E_{A^*}\). Entities in \(E_{A^*}\) are ranked according to their (static) importance, \(\text{score}(e)\), which can be based on, e.g., popularity. (We discuss various query-independent entity ranking measures in Sect. 4.6.) Aspects that cannot contribute more entities to the result list (i.e., \(E_A \setminus E_q \cup E_R = \emptyset\)) are removed. If the objective is to provide diversification, the aspects are re-ranked after each entity selection. Metzger et al. [39] further suggest a relaxation heuristic; instead of removing a now empty maximal aspect, it may be relaxed to cover more entities from “the same branch of similar entities.” A simple relaxation strategy is to consider each of the basic aspects \(a\) and either dropping it or replacing it with the parent type, if \(a\) is a type aspect.

## 4.6 Query-Independent Ranking

Up until this point, we have focused on ranking entities with respect to a query. In a sense, all entities in the catalog have had an equal chance when competing for the top positions in the ranked result list. There are, however, additional factors playing a role, which are independent of the actual query.

For the sake of illustration, let us consider the task of searching on an e-commerce site. Let us imagine that for some search query there are two or more products that have the same retrieval score, i.e., they are equally good matches. This could happen, for instance, because those products have very similar names and descriptions. It might also happen that the user is not issuing a keyword query, but is browsing the product catalog by product category or brand instead. In this case, the selected category or brand is the query, for which a complete and perfect set of
results may be returned (i.e., all entities that belong to the selected category/brand are equally relevant). In both the above scenarios, the following question arises: How can we determine the relative ranking of entities that have the same relevance to the query?

Intuitively, the ranking of products in the above example could be based on which (1) has been clicked on more, (2) has been purchased more, (3) has a better consumer rating, (4) has a better price, (5) is a newer model, etc., or some combination of these. Notice that all these signals are independent of the actual query.

Query-independent (or “static”) entity scores capture the notion of entity importance. That is, they reflect the extent with which a given entity is relevant to any query.

Let us refer back to the language modeling approach from the previous chapter (Sect. 3.3.1.1). There, we rewrote the probability $P(e|q)$ using Bayes’ theorem. We repeat that equation (Eq. (3.7)) here for convenience:

$$P(e|q) = \frac{P(q|e)P(e)}{P(q)} = P(q|e)P(e).$$

Back then, we ignored the second component, $P(e)$, which is the prior probability of the entity. As we can see, the generative formulation of the language modeling framework offers a theoretically principled way of incorporating the query-independent entity score.

Generally, the query-independent entity score may be incorporated into any retrieval method by way of multiplication of the (original) query relevance score:

$$score'(e; q) = score(e) \times score(e; q).$$

Since $score(e)$ is independent of the actual query, it can be computed offline and stored in the entity index (hence then name static score).

In this section, we present two main groups of query-independent measures, which revolve around the notions of popularity and centrality. Multiple query-independent features may be combined using a learning-to-rank approach [16].

### 4.6.1 Popularity

Intuitively, the popularity of an entity is an indicator of its importance. It may be expressed as a probability:

$$P(e) = \frac{c(e)}{\sum_{e' \in E} c(e')} ,$$
where \(c(e)\) is some measure of popularity. Options for estimating \(c(e)\) include the following:

- Aggregated *click or view statistics* over a certain period of time [39]. One useful resource is the Wikipedia page view statistics, which is made publicly available.\(^8\)
- The *frequency of the entity* in a large corpus, like the Web. This may be approximated by using the entity’s name as a query and obtaining the number of hits from a web search engine API [16]. Alternatively, the name of the entity can be checked against a large and representative n-gram corpus to see how often it was used [16]. One such resource is the Google Books Ngram Corpus [37].\(^9\) When the web corpus is annotated with specific entities, it is possible to search using entity IDs, instead of entity names; see Sect. 5.9.2 for web corpora annotated with entities from Freebase.

In addition to the above “universal” signals, there are many possibilities in specific domains, e.g., in an e-commerce setting, the number of purchases of a product; in academic search, the number of citations of an article; on a social media platform, the number of likes/shares, etc.

### 4.6.2 Centrality

Entity *centrality* can be derived from the underlying graph structure using (adaptations of) link analysis algorithms, like PageRank [11] and HITS [35]. In this section, we focus on the PageRank algorithm, due to its popularity, and discuss its application to entities. For similar work using the HITS algorithm, see, e.g., [8, 23].

#### 4.6.2.1 PageRank

Web pages can be represented as a directed graph, where nodes are pages and edges are hyperlinks connecting these pages to each other. *PageRank* assigns a numerical score to each page that reflects its importance. The main idea is that it is not only the number of incoming links that matters but also the quality of those links. Links that originate from more important sources (i.e., pages with high PageRank score) weigh more than unimportant pages. Another way to understand PageRank is that it measures the likelihood of a random surfer landing on a given web page. The random surfer is assumed to navigate on the Web as follows. In each step, the user either (1) moves to one of the pages linked from the current page or (2) jumps to a random web page. The random jump also ensures that the user does not get stuck on a page that has no outgoing links.

\(^8\)http://dumps.wikimedia.org/other/pagecounts-raw/

\(^9\)http://books.google.com/ngrams.
The PageRank score of an entity is computed as follows:

\[ \text{PR}(e) = \frac{\alpha}{|E|} + (1 - \alpha) \sum_{i=1}^{n} \frac{\text{PR}(e_i)}{|L_{e_i}|}, \]

(4.4)

where \( \alpha \) is the probability of a random jump (typically set to 0.15 [11]), \( e_1 \ldots e_n \) are the entities linking to \( e \), and \( |L_{e_i}| \) is the number of outgoing links of \( e_i \). Notice that PageRank is defined recursively, thus needs to be computed iteratively. In each successive iteration, the PageRank score is determined using the PageRank values from the previous iteration. Traditionally, all nodes are initialized with an equal score (i.e., \( 1/|E| \)). The final values are approximated fairly accurately even after a few iterations. Notice that the PageRank scores form a probability distribution (\( \sum_{e \in E} \text{PR}(e) = 1 \)). Therefore, the iterative computation process may also be interpreted as propagating a probability mass across the graph. It is common to use quadratic extrapolation (e.g., between every fifth iteration) to speed up convergence [31].

4.6.2.2 PageRank for Entities

A number of variations and extensions of PageRank have been proposed for entities. We shall take a look at a selection of them below. The first important question that needs to be addressed is: How to construct the entity graph?

- Using unstructured entity descriptions, references to other entities need to be recognized and disambiguated (see Chap. 5). Directed edges are added from each entity to all the other entities that are mentioned in its description.
- In a semi-structured setting, e.g., Wikipedia, links to other entities might be explicitly provided.
- When working with structured data, RDF triples define a graph (i.e., the knowledge graph). Specifically, subject and object resources (URIs) are nodes and predicates are edges.

In the remainder of this section, we shall concentrate on the structured data setting. Recall that nodes in an RDF graph include not only entities but other kinds of resources as well (in particular entity types, cf. Fig. 2.3). Since it is not pages but resources that are being scored, Hogan et al. [29] refer to the computed quantity as ResourceRank. Instead of computing static scores on the entire graph, ResourceRank computes PageRank scores over topical subgraphs (resources matching the keyword query and their surrounding resources).

It can be argued that the traditional PageRank model is unsuitable for entity popularity calculation because of the heterogeneity of entity relationships [44]. ObjectRank [1] extends PageRank to weighted link analysis, applied to the problem of keyword search in databases. ObjectRank, however, relies on manually assigned link weights. This makes the approach applicable only in restricted domains (e.g.,
Nie et al. [44] introduce PopRank, to rank entities within a specific domain. A separate popularity propagation factor (i.e., weight) is assigned to each link depending on the type of entity relationship. For example, in their case study on academic search, the types of entities involved are authors, papers, and conferences/journals. The types of relationships between entities include cited-by, authored-by, and published-by. It is problematic to manually decide the propagation factors for each type of relationship. Instead, partial entity rankings are collected from domain experts for some subsets of entities. The setting of propagation factor then becomes a parameter estimation problem using the partial rankings as training data. This method is applicable in any vertical search domain, where the number of entity types and relationships is sufficiently small (product search, music search, people search, etc.).

The quality of ranking may be improved by taking the context of data into account. When working with RDF, this context is the provenance or source of data. Hogan et al. [29] extract a context graph from the RDF graph and compute PageRank scores on this graph, referred to as ContextRank. Further, by inferring links between contexts and resources (as well as between contexts), a combined graph can be created. This graph contains both resource and context nodes. ReConRank refers to the PageRank scores computed on this unified graph. ReConRank captures the “symbiotic relationship between context authority and resource authority” [29].

### 4.6.2.3 A Two-Layered Extension of PageRank for the Web of Data

The heterogeneity of data is further increased when moving from a single knowledge base to the Web of Data, which is comprised of multiple datasets. Instead of considering only the ranks of entities, the dataset where that entity originates from should be also taken into account. Additionally, computing popularity scores on a graph of that scale brings challenges.

Delbru et al. [17] propose a two-layered extension of PageRank. Their method, called DING, operates in a hierarchical fashion between the dataset and entity layers. The top layer is comprised of a collection of inter-connected datasets, whereas the bottom layer is composed of (independent) graphs of entities; see Fig. 4.8 for an illustration. The hierarchical random surfer model works as follows:

1. The user randomly selects a dataset.
2. Then, the user may choose and perform one of these actions:
   (a) Select randomly an entity in the current dataset.
   (b) Jump to another dataset by following a link from the current dataset.
   (c) End the browsing.

According to the above model, the computation is performed in two stages. First, the importance of the top level dataset nodes (DatasetRank) are calculated. The rank score of a dataset is composed of two parts: (1) the contribution from the other datasets via incoming links, and (2) the probability of selecting the dataset
Fig. 4.8 A two-layer model for computing entity importance in the Web of Data. Dashed edges on the entity layer represent inter-dataset links; these are aggregated to form weighted links on the dataset layer. Figure is based on [17]

during a random jump (which is set proportional to its size). Then, in a second stage, the importance of entities within a given dataset (local entity rank) are computed. Local entity rank is PageRank applied on the intra-links of the dataset. Delbru et al. [17] further introduce an unsupervised link weighting scheme (LF-IDF, motivated by TF-IDF), which assigns high importance to links with high frequency within a given dataset and low dataset frequency across the collection of datasets. Mind that the computation of the local entity ranks can be performed independently for each dataset (and thus can easily be parallelized). The two scores are combined using the following equation:

$$\text{score}_{DING}(e) = r(D) \cdot r(e) \frac{|D|}{\sum_{D'} |D'|}$$

where $r(D)$ is DatasetRank and $r(e)$ is the local entity rank. The last term in the equation serves as a normalization factor. Since the magnitude of local entity ranks depends on the size of the dataset (recall that $\sum_{e \in E} r(e) = 1$), entities in small datasets will receive higher scores than in large datasets. This is compensated by taking the dataset size into account, where $|D|$ denotes the size of dataset $D$ (measured in the number of entities contained).

### 4.6.3 Other Methods

Without detailed elaboration, we mention a number of other query-independent methods that have been proposed in the literature. Instead of centrality, more simple frequency-based measures may be used, which can be extracted from the knowledge
graph. These include the number of subjects, objects, distinct predicates, etc., for at most $k$ steps away from the entity node [16]. Alternatively, one may not even need to consider the graph structure but can obtain statistics from the set of RDF triples that contain a given entity. For example, how many times does the entity occur as a subject in an RDF triple where the object is a literal [16]?

### 4.7 Summary

This chapter has introduced semantically enriched models for entity retrieval. In the various entity retrieval tasks we have looked at, the common theme has been to exploit specific properties of entities: attributes, types, and relationships. Typically, this semantic enrichment is operationalized by combining the term-based retrieval score with one or multiple property-specific score components. Utilizing entity attributes and relationships is shown to yield relative improvements in the 5–20% range [14, 25, 52]. Exploiting entity type information is even more rewarding, with relative improvements ranging from 25% to well over 100% [3, 15, 47].

While these results are certainly encouraging, a lot remains to be done. First, most of the approaches we have presented in this chapter were tailored to a rather specific entity retrieval task, making assumptions about the format/characteristics of the query and of the underlying user intent. It is an open question whether a “one size fits all” model can be developed, or if effort might be better spent by designing task-specific approaches and automatically deciding which one of these should be used for answering a given input query. Second, we have assumed a semantically enriched query as input, referred to as keyword++ query, which explicitly provides query entities, types, etc. Obtaining these “enrichments” automatically is an area of active research (which we will be discussing in Chap. 7). Third, the potential of combining structured and unstructured data has not been fully explored and realized. Instead of performing this combination inside the retrieval model, an alternative is to enrich the underlying collections. Annotating documents with entities can bring structure to unstructured documents, which in turn can help populate knowledge bases with new information about entities. In Part II, we will discuss exactly how this strategy of annotating documents, and consequently populating knowledge bases, can be accomplished.

### 4.8 Further Reading

It is worth pointing out that there are many additional factors beyond topical relevance to consider in real-life applications. One particular example is e-commerce search, where the quality of images has shown to have an impact on which results get clicked [21, 24]. Another example scenario is that of expert search, where it has been
shown that contextual factors (e.g., availability, experience, physical proximity) also play a role [28].

References

1. Balmin, A., Hristidis, V., Papakonstantinou, Y.: Objectrank: Authority-based keyword search in databases. In: Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30, VLDB ’04, pp. 564–575. VLDB Endowment (2004)
2. Balog, K.: On the investigation of similarity measures for product resolution. In: Proceedings of the Workshop on Discovering Meaning On the Go in Large Heterogeneous Data, LHD-11 (2011)
3. Balog, K., Bron, M., De Rijke, M.: Query modeling for entity search based on terms, categories, and examples. ACM Trans. Inf. Syst. 29(4), 22:1–22:31 (2011a)
4. Balog, K., de Rijke, M.: Finding similar experts. In: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval, SIGIR ’07, pp. 821–822. ACM (2007). doi: 10.1145/1277741.1277926
5. Balog, K., Serdyukov, P., de Vries, A.P.: Overview of the TREC 2010 Entity track. In: Proceedings of the Nineteenth Text REtrieval Conference, TREC ’10. NIST (2011b)
6. Balog, K., Serdyukov, P., de Vries, A.P.: Overview of the TREC 2011 Entity track. In: The Twentieth Text REtrieval Conference Proceedings, TREC ’11. NIST (2012)
7. Balog, K., de Vries, A.P., Serdyukov, P., Thomas, P., Westerveld, T.: Overview of the TREC 2009 Entity track. In: Proceedings of the Eighteenth Text REtrieval Conference, TREC ’09. NIST (2010)
8. Bamba, B., Mukherjea, S.: Utilizing resource importance for ranking semantic web query results. In: Proceedings of the Second International Conference on Semantic Web and Databases, SWDB ’04, pp. 185–198. Springer (2005). doi: 10.1007/978-3-540-31839-2_14
9. 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
10. Blanco, R., Halpin, H., Herzig, D.M., Mika, P., Pound, J., Thompson, H.S., Duc, T.T.: Entity search evaluation over structured web data. In: Proceedings of the 1st International Workshop on Entity-Oriented Search, EOS ’11, pp. 65–71 (2011)
11. Brin, S., Page, L.: The anatomy of a large-scale hypertextual web search engine. In: Seventh International World-Wide Web Conference, WWW ’98 (1998)
12. Bron, M., Balog, K., de Rijke, M.: Ranking related entities: Components and analyses. In: Proceedings of the 19th ACM International Conference on Information and Knowledge Management, CIKM ’10, pp. 1079–1088 (2010). doi: 10.1145/1871437.1871574
13. Bron, M., Balog, K., de Rijke, M.: Example-based entity search in the web of data. In: Proceedings of the 35th European conference on Advances in Information Retrieval, ECIR ’13, pp. 392–403. Springer (2013). doi: 10.1007/978-3-642-36973-5_33
14. Campinas, S., Delbru, R., Tummarello, G.: Effective retrieval model for entity with multi-valued attributes: BM25MF and beyond. In: Proceedings of the 18th International Conference on Knowledge Engineering and Knowledge Management, EKAW ’12, pp. 200–215. Springer (2012). doi: 10.1007/978-3-642-33876-2_19
15. Ciglan, M., Nørøvåg, K., Hluchý, L.: The SemSets model for ad-hoc semantic list search. In: Proceedings of the 21st International Conference on World Wide Web, WWW ’12, pp. 131–140. ACM (2012). doi: 10.1145/2187836.2187855
16. Dali, L., Fortuna, B., Duc, T.T., Mladenić, D.: Query-independent learning to rank for RDF entity search. In: Proceedings of the 9th International Conference on The Semantic Web: Research and Applications, ESWC ’12, pp. 484–498. Springer (2012). doi: 10.1007/978-3-642-30284-8_39

17. Delbru, R., Toupikov, N., Catasta, M., Tummarello, G., Decker, S.: Hierarchical link analysis for ranking web data. In: Proceedings of the 7th International Conference on The Semantic Web: Research and Applications - Volume Part II, ESWC’10, pp. 225–239. Springer (2010). doi: 10.1007/978-3-642-13489-0_16

18. Demartini, G., Firan, C.S., Iofciu, T., Krestel, R., Nejdl, W.: Why finding entities in Wikipedia is difficult, sometimes. Information Retrieval 13(5), 534–567 (2010a). doi: 10.1007/s10791-010-9135-7

19. Demartini, G., Iofciu, T., de Vries, A.: Overview of the INEX 2009 Entity Ranking track. In: Geva, S., Kamps, J., Trotman, A. (eds.) Focused Retrieval and Evaluation, Lecture Notes in Computer Science, vol. 6203, pp. 254–264. Springer (2010b). doi: 10.1007/978-3-642-14556-8_26

20. Demartini, G., de Vries, A.P., Iofciu, T., Zhu, J.: Overview of the INEX 2008 Entity Ranking track. In: Advances in Focused Retrieval: 7th International Workshop of the Initiative for the Evaluation of XML Retrieval (INEX 2008), pp. 243–252 (2009). doi: 10.1007/978-3-642-03761-0_25

21. Di, W., Sundaresan, N., Piramuthu, R., Bhardwaj, A.: Is a picture really worth a thousand words? - on the role of images in e-commerce. In: Proceedings of the 7th ACM International Conference on Web Search and Data Mining, WSDM ’14, pp. 633–642. ACM (2014). doi: 10.1145/2556195.2556226

22. Fang, Y., Si, L.: Related entity finding by unified probabilistic models. World Wide Web 18(3), 521–543 (2015). doi: 10.1007/s11280-013-0267-8

23. Franz, T., Schultz, A., Sizov, S., Staab, S.: TripleRank: Ranking semantic web data by tensor decomposition. In: Proceedings of the 8th International Semantic Web Conference, ISWC ’09, pp. 213–228. Springer (2009). doi: 10.1007/978-3-642-04930-9_14

24. Goswami, A., Chittar, N., Sung, C.H.: A study on the impact of product images on user clicks for online shopping. In: Proceedings of the 20th International Conference Companion on World Wide Web, WWW ’11, pp. 45–46. ACM (2011). doi: 10.1145/1963192.1963216

25. Hasibi, F., Balog, K., Bratsberg, S.E.: Exploiting entity linking in queries for entity retrieval. In: Proceedings of the 2016 ACM on International Conference on the Theory of Information Retrieval, ICTIR ’16, pp. 209–218. ACM (2016). doi: 10.1145/2970398.2970406

26. He, Y., Xin, D.: SEISA: Set expansion by iterative similarity aggregation. In: Proceedings of the 20th International Conference on World Wide Web, WWW ’11. ACM (2011). doi: 10.1145/1963405.1963467

27. Hoffart, J., Seufert, S., Nguyen, D.B., Theobald, M., Weikum, G.: KORE: Keyphrase overlap relatedness for entity disambiguation. In: Proceedings of the 21st ACM International Conference on Information and Knowledge Management, CIKM ’12, pp. 545–554. ACM (2012). doi: 10.1145/2396761.2396832

28. Hofmann, K., Balog, K., Bogers, T., de Rijke, M.: Contextual factors for finding similar experts. Journal of the American Society for Information Science and Technology 61(5), 994–1014 (2010). doi: 10.1002/asi.v61:5

29. Hogan, A., Harth, A., Decker, S.: ReConRank: A scalable ranking method for semantic web data with context. In: 2nd Workshop on Scalable Semantic Web Knowledge Base Systems (2006)

30. Jeh, G., Widom, J.: SimRank: A measure of structural-context similarity. In: Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD ’02, pp. 538–543. ACM (2002). doi: 10.1145/775047.775126

31. Kamvar, S.D., Haveliwala, T.H., Manning, C.D., Golub, G.H.: Extrapolation methods for accelerating pagerank computations. In: Proceedings of the 12th International Conference on World Wide Web, WWW ’03, pp. 261–270. ACM (2003). doi: 10.1145/775152.775190
32. Kaptein, R., Kamps, J.: Exploiting the category structure of Wikipedia for entity ranking. Artificial Intelligence 194, 111–129 (2013). doi: 10.1016/j.artint.2012.06.003
33. Kaptein, R., Koolen, M., Kamps, J.: Result diversity and entity ranking experiments: anchors, links, text and Wikipedia. In: Proceedings of the Eighteenth Text REtrieval Conference, TREC ’09. NIST (2010a)
34. Kaptein, R., Serdyukov, P., De Vries, A., Kamps, J.: Entity ranking using Wikipedia as a pivot. In: Proceedings of the 19th ACM international conference on Information and knowledge management, CIKM ’10, pp. 69–78. ACM (2010b). doi: 10.1145/1871437.1871451
35. Kleinberg, J.M.: Authoritative sources in a hyperlinked environment. J. ACM 46(5), 604–632 (1999). doi: 10.1145/324133.324140
36. Li, X., Li, C., Yu, C.: EntityEngine: Answering entity-relationship queries using shallow semantics. In: Proceedings of the 19th ACM International Conference on Information and Knowledge Management, CIKM ’10, pp. 1925–1926. ACM (2010). doi: 10.1145/1871437.1871766
37. Lin, Y., Michel, J.B., Aiden, E.L., Orwant, J., Brockman, W., Petrov, S.: Syntactic annotations for the Google books Ngram corpus. In: Proceedings of the ACL 2012 System Demonstrations, ACL ’12, pp. 169–174. Association for Computational Linguistics (2012)
38. Losada, D.E., Azzopardi, L.: Assessing multivariate Bernoulli models for information retrieval. ACM Trans. Inf. Syst. 26(3), 17:1–17:46 (2008). doi: 10.1145/1361684.1361690
39. Metzger, S., Schenkel, R., Sydow, M.: QBEES: Query-by-example entity search in semantic knowledge graphs based on maximal aspects, diversity-awareness and relaxation. J. Intell. Inf. Syst. 49(3), 333–366 (2017). doi: 10.1007/s10844-017-0443-x
40. Metzler, D., Croft, W.B.: A Markov Random Field model for term dependencies. In: Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR ’05, pp. 472–479. ACM (2005). doi: 10.1145/1076034.1076115
41. 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)
42. Milne, D., Witten, I.H.: Learning to link with Wikipedia. In: Proceedings of the 17th ACM Conference on Information and Knowledge Management, CIKM ’08, pp. 509–518 (2008). doi: 10.1145/1458082.1458150
43. Minkov, E., Cohen, W.W.: Improving graph-walk-based similarity with reranking: Case studies for personal information management. ACM Trans. Inf. Syst. 29(1), 4:1–4:52 (2010). doi: 10.1145/1877766.1877770
44. Nie, Z., Zhang, Y., Wen, J.R., Ma, W.Y.: Object-level ranking: Bringing order to web objects. In: Proceedings of the 14th International Conference on World Wide Web, WWW ’05, pp. 567–574. ACM (2005). doi: 10.1145/1060745.1060828
45. Pehcevska, J., Thom, J.A., Vercoustre, A.M., Naumovski, V.: Entity ranking in Wikipedia: utilising categories, links and topic difficulty prediction. Information Retrieval 13(5), 568–600 (2010). doi: 10.1007/s10791-009-9125-9
46. Pennington, J., Socher, R., Manning, C.D.: Glove: Global vectors for word representation. In: Empirical Methods in Natural Language Processing, EMNLP ’14, pp. 1532–1543 (2014)
47. Raviv, H., Carmel, D., Kurklenski, O.: A ranking framework for entity oriented search using Markov Random Fields. In: Proceedings of the 1st Joint International Workshop on Entity-Oriented and Semantic Search, JIWES ’12, pp. 1:1–1:6. ACM (2012). doi: 10.1145/2379307.2379308
48. Sarmento, L., Jijikoun, V., de Rijke, M., Oliveira, E.: “More like these”: Growing entity classes from seeds. In: Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management, CIKM ’07, pp. 959–962. ACM (2007). doi: 10.1145/1321440.1321585
49. Schuhmacher, M., Dietz, L., Paolo Ponzetto, S.: Ranking entities for web queries through text and knowledge. In: Proceedings of the 24th ACM International Conference on
References

Information and Knowledge Management, CIKM ’15, pp. 1461–1470. ACM (2015). doi: 10.1145/2806416.2806480

50. Schuhmacher, M., Ponzetto, S.P.: Knowledge-based graph document modeling. In: Proceedings of the 7th ACM International Conference on Web Search and Data Mining, WSDM ’14, pp. 543–552 (2014). doi: 10.1145/2556195.2556250

51. Sehgal, V., Getoor, L., Viechnicki, P.D.: Entity resolution in geospatial data integration. In: Proceedings of the 14th Annual ACM International Symposium on Advances in Geographic Information Systems, GIS ’06, pp. 83–90. ACM (2006). doi: 10.1145/1183471.1183486

52. Tonon, A., Demartini, G., Cudré-Mauroux, P.: Combining inverted indices and structured search for ad-hoc object retrieval. In: Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR ’12, pp. 125–134. ACM (2012). doi: 10.1145/2348283.2348304

53. 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)

54. de Vries, A.P., Vercoustre, A.M., Thom, J.A., Craswell, N., Lalmas, M.: Overview of the INEX 2007 Entity Ranking track. In: Proceedings of the 6th Initiative on the Evaluation of XML Retrieval, INEX ’07, pp. 245–251. Springer (2008). doi: 10.1007/978-3-540-85902-4_22

55. Wang, C., Chakrabarti, K., He, Y., Ganjam, K., Chen, Z., Bernstein, P.A.: Concept expansion using web tables. In: Proceedings of the 24th International Conference on World Wide Web, WWW ’15, pp. 1198–1208. International World Wide Web Conferences Steering Committee (2015). doi: 10.1145/2736277.2741644

56. Yang, L., Guo, Q., Song, Y., Meng, S., Shokouhi, M., McDonald, K., Croft, W.B.: Modeling user interests for zero-query ranking. In: Proceedings of the 38th European Conference on IR Research, ECIR ’16, pp. 171–184. Springer (2016). doi: 10.1007/978-3-319-30671-1_13

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.