TECHNISCHE UNIVERSITÄT DRESDEN

KNOWLEDGE GRAPHS

Lecture 14: Summary

Markus Krötzsch
Knowledge-Based Systems

TU Dresden, 29th Jan 2019

Results teaching evaluation

Review: Community detection

Community structure can be very helpful to understand graphs:

- Underlying principles of their creation (and future growth)
- Inhomogeneous features
- Optimised processing

“Community” can mean many things, depending on what the graph models:

- For example, using betweenness (Girvan and Newman), modularity, closeness
- Similar to situation for centrality
- More criteria possible (coming up next)

Further approaches to community detection

Further approaches to community detection (1)

Minimum Cuts: cut graph into disconnected components by removing edges

- Prefer cuts that (1) cut only few edges and (2) lead to components of similar size (the approach of assigning a to-be-minimised value to a cut may vary)
- Several efficient algorithms are available
- Good for partitioning network into "local" neighbourhoods by cutting communication bottlenecks (e.g., data placement in distributed computing), less suitable for community detection

Graph partitioning is mostly used in slightly different application areas.

Further approaches to community detection (2)

Bi-Cliques ("Trawling"): discover communities starting from large complete bi-partite subgraphs

- On general graphs, the approach works as follows:
  (1) Split graph into two equal-sized vertex sets
  (2) Look for sub-sets where all vertices on the left are connected to all vertices on the right (mathematical background: in sufficiently dense communities, large bi-cliques must exist)
  (3) Grow communities around the vertices by adding highly linked vertices
- Efficient computation uses techniques from frequent itemset mining to search such "bi-cliques"
- Good for finding also small communities; applicable also to k-partite graphs

Very common approach in social network analysis; can be applied to rather large networks.

Further approaches to community detection (3)

Statistical inference: try to find a model that explains the observed data

- Consider some parametrised graph generator ("the model"), and try to find the parameter values that are most likely to lead to the given graph
- Generally hard to solve and depending on model; many algorithms exists, e.g., search methods for continuous parameter fitting (e.g., gradient descent)
- Capable of finding overlapping communities; might lead to even deeper insights into the nature of the graph than mere community detection

More complex but potentially more insightful and of greater predictive value

Summary and Conclusion

Summary

What we learned:

- How to represent knowledge graphs (in RDF, in Property Graph, in mathematics)
- How to model knowledge in graphs (basic data structures, RDBMS encoding)
- How to query knowledge graphs (in SPARQL, in Cypher)
- Where to find interesting graph data (Wikidata)
- How to ensure knowledge graph quality (general principles, SHACL, ShEx)
- How to analyse network structures (centrality, communities)

. . . and how to do much of this in code.

Summary of Unknowns

What we omitted:

- How to extract knowledge from unstructured sources
  (see courses on information extraction, natural language processing)
- How to build database systems for large graph data
  (see courses on database technologies)
- How to extend graph data with ontological models
  (see course “Semantic Web Technologies”)
- How to learn and predict knowledge graph structures
  (mostly open – significant research but no breakthrough yet)
- How to model knowledge in graphs
  (vaguely defined, but there are some methodologies/paradigms)
- How to apply knowledge graphs in specific scenarios (find your own!)

. . . and many more.

Lecture 2: Encoding Graphs with RDF

RDF is a W3C standard for describing directed, edge-labelled graphs in an interoperable way

It identifies vertices and edge-types using IRIs, and it can use many datatypes

Several syntactic formats exist for exchanging RDF data, the simplest being N-Triples

What’s next?

- How can we encode data in RDF?
- Where can we get RDF data? Application examples?
- How can we query and analyse such graphs?

Lecture 3: Modelling in RDF

Edge labels (predicates) in RDF are represented by their respective properties, which can be used to add further information

RDF can most naturally be viewed as hypergraphs with ternary edges

RDF can express n-ary relations and (with some effort) lists

What’s next?

- How to query RDF graphs with SPARQL
- Wikidata as a working example to try out our knowledge
- More powerful SPARQL features

Lecture 4: Introduction to SPARQL

SPARQL, the main query language for RDF, is based on matching graph patters

Basic Graph Pattern matching is already rather complex (NP-complete), but can be implemented using joins

What's next?

- Wikidata as a working example to try out our knowledge
- More SPARQL query features
- Further background on SPARQL complexity and semantics

Lecture 5: Wikidata

Wikidata, the knowledge base of Wikipedia, is a freely available knowledge graph

Wikidata supports a document-centric and a graph-centric perspective

Content can be converted to RDF and a public SPARQL query service is available

What's next?

- More SPARQL query features
- Further background on SPARQL complexity and semantics
- Graphs beyond RDF

Lecture 6: Advanced Features of SPARQL

Property Path Patterns are used to describe (arbitrarily long) paths in graphs

Filters can express many conditions to eliminate some of the query results

Solutions set modifiers define standard operations on result sets

What's next?

- More SPARQL features: aggregates, subqueries, union, optional
- Further background on SPARQL complexity and semantics
- Other graph models and their query languages

Lecture 7: Advanced Features of SPARQL (2)

Solutions set modifiers define standard operations on result sets

Important SPARQL query operators are **`UNION`**, **`MINUS`**, **`OPTIONAL`**, **`BIND`**, and **`VALUES`**

The semantics of SPARQL is defined using algebraic operators

Aggregates are used to obtain answers that combine several solutions.

What's next?

- Further background on SPARQL complexity and semantics
- Other graph models and their query languages
- Other aspects of graph analysis and management

Lecture 8: Expressive Power and Complexity of SPARQL

SPARQL is PSpace-complete for query and combined complexity[1]

SPARQL is NL-complete for data complexity, hence practically tractable and well parallelisable[1]

SPARQL expressivity is still limited, partly by design.

What's next?

- Property graph: another popular graph data model
- The Cypher query language
- Quality assurance in knowledge graphs

[1]The matching upper bound has not been proven with the full set of features.

Lecture 9: Limits of SPARQL / Introduction to Property Graphs

SPARQL expressivity is still limited, partly by design

Property Graph is a general concept for organising graph data in two layers: a primary graph layer and a sub-ordinate key-value-set layer

Property graph has many different, incompatible implementations, based on several database paradigms (object, relational, RDF graph)

What's next?

- The Cypher query language
- Quality assurance in knowledge graphs
- More graph analysis

Lecture 10: Querying Property Graphs with Cypher

Cypher is a query language for property graphs

It is based on clauses (like SQL) and graph patterns (like SPARQL)

Not all regular expressions are supported, but powerful path manipulation features are available.

Graph pattern matching is not based on homomorphisms like in other query languages (SQL, SPARQL, etc.) but on a notion of isomorphim on edges

What's next?

- Holidays
- Final remarks on Cypher
- Quality assurance in knowledge graphs

Lecture 11: Cypher / Knowledge Graph Quality

OpenCypher supports aggregates, subqueries, unions, and optional matches (like SPARQL, but differing in some design choices)

Defining and measuring knowledge graph quality is difficult; there are many criteria

Competency questions and unit tests are basic approaches for automatic quality checks

RDF constraint languages like SHACL and ShEx can declaratively specify constraints

What's next?

- Ontological modelling
- More graph analysis
- Examinations

Lecture 12: Centrality

Centrality criteria aim at identifying the most important vertices in a graph, based on graph structures

Eigenvalue centrality and related measures have been applied in a wide range of areas, notably Web search

What's next?

- More network analysis
- Summary
- Examinations

Lecture 13: Community detection

Closeness and betweenness centrality evaluate the shortest paths for rather different results

The Girvan and Newman algorithm exploits (edge) betweenness for clustering networks into communities

Modularity is an imperfect measure for the quality of a hypothesised community structure

What's next?

- More network analysis
- Summary
- Examinations

Lecture 14: Summary

What's next?

- Concentrated learning
- Successful examinations
- Free time