Querying Knowledge Graphs with KGTK Kypher

Hans Chalupsky
USC Information Sciences Institute, hans@isi.edu

Outline

- KGTK data model
- KGTK commands
- Kypher query language
- Hands-on Kypher tutorial

KGTK Data Model

KGTK Design: simplicity + maximal flexibility

One KG = one or more TSV files:

- Columns: <node1, label, node2, edge-id>

Toolkit commands:

- command_i(TSV_1, TSV_2, ...) → TSV_1, TSV_2, …
- input and output are sets of TSV files

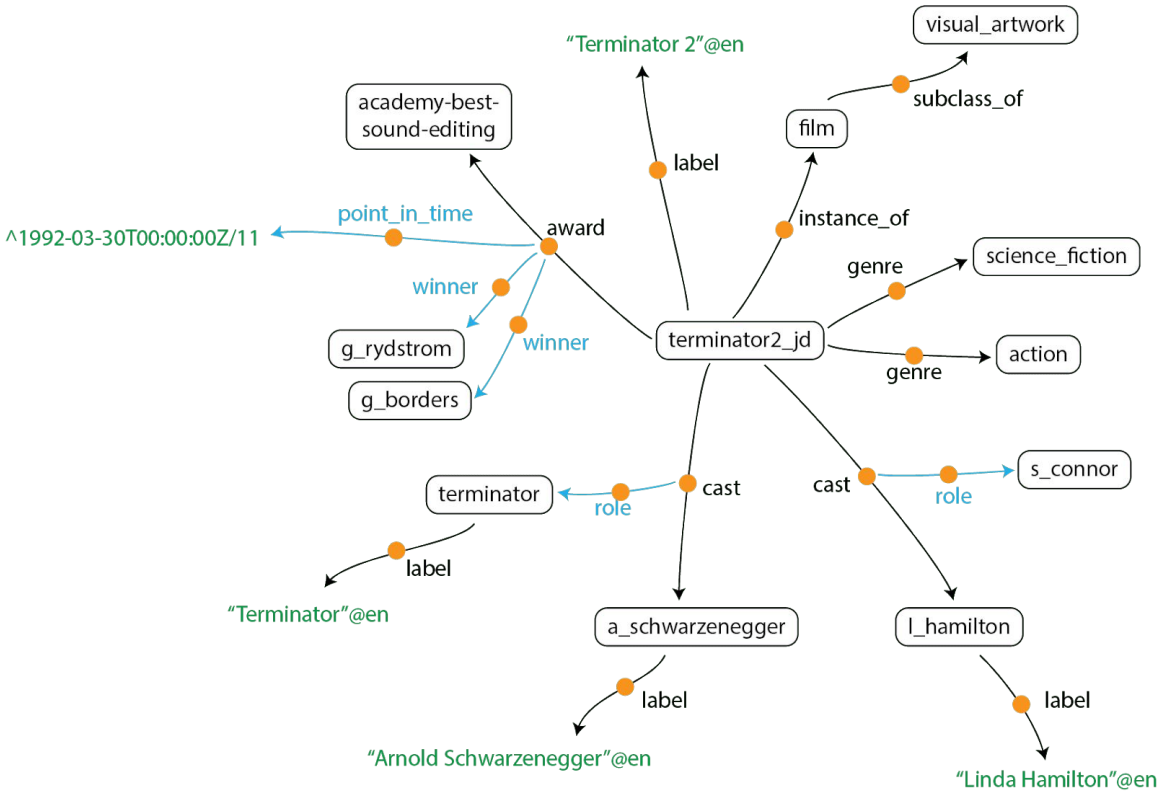
Schema free:

- nodes can be anything (identifiers, strings, numbers, dates, …)
- edge labels can also be anything
- don't need to declare anything

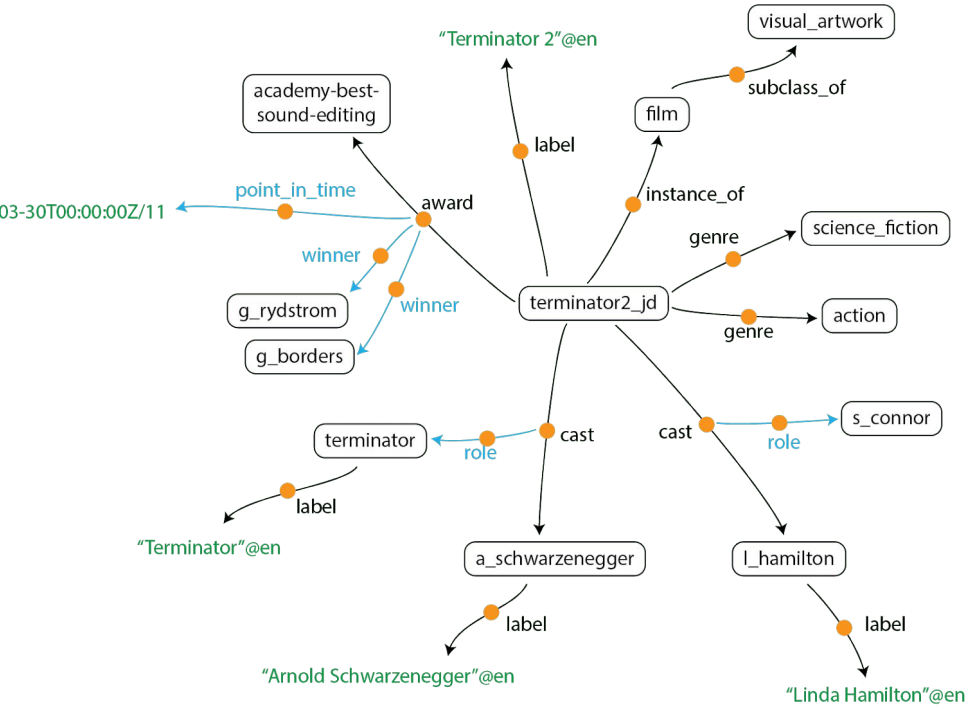
Structured literals:

- commonly used literal types are represented as one symbol
- syntax designed for easy, efficient parsing

KGTK Knowledge Graph Representation



KGTK Graphs: <edge-id, subject, predicate, object> in TSV



id	node1	label	node2
	terminator2_jd	label	"Terminator 2"@en
	terminator2_jd	instance_of	film
	terminator2_jd	genre	science_fiction
	terminator2_jd	genre	action
t4	terminator2_jd	cast	a_schwarzenegger
	t4	role	terminator
t6	terminator2_jd	cast	l_hamilton
	t6	role	s_connor
t8	terminator2_jd	award	academy_best_sound_editing
	t8	point_in_time	^1992-03-30T00:00:00Z/11
	t8	winner	g_rydstrom
	t8	winner	g_borders

KGTK structured literals

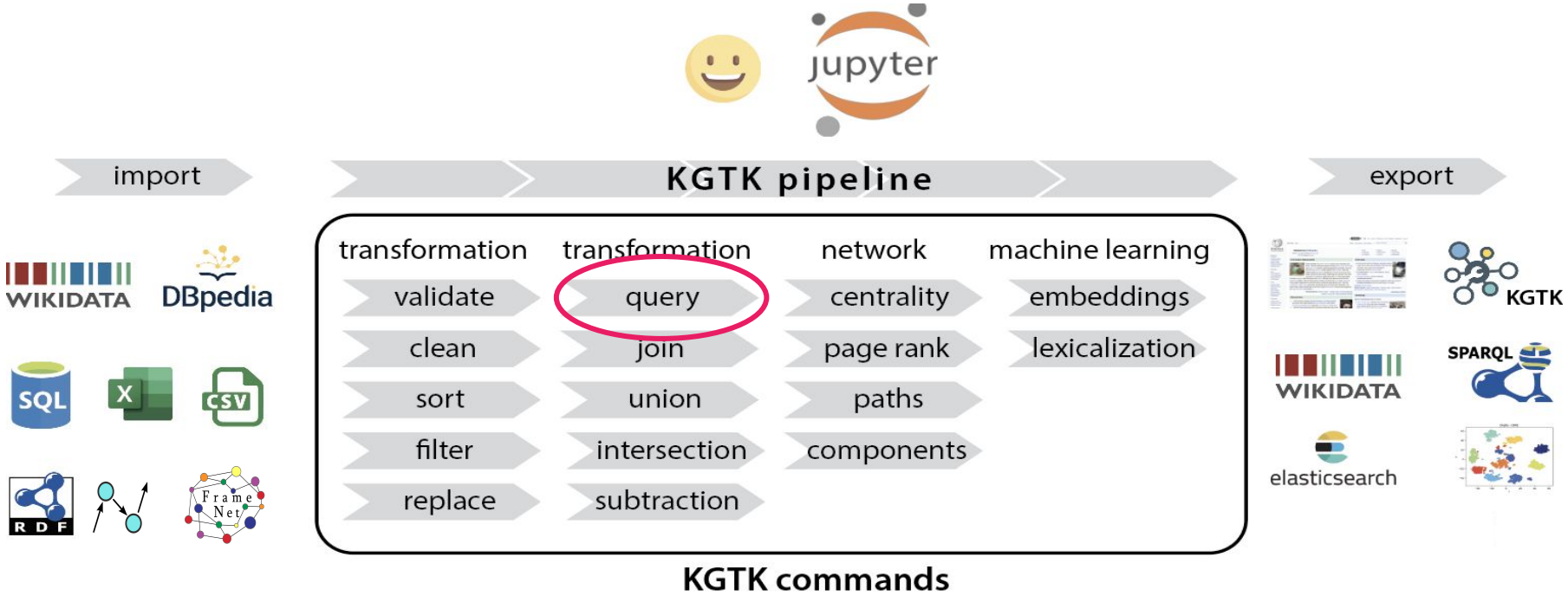
- Strings
  - "KGTK", "Terminator 2"
- Language-qualified strings
  - 'Arnold Schwarzenegger'@de
- Date/time
  - ^2020-10-30T02:03:57+10:30/9, ^1801-00-00T00:00:00Z/7
- Quantities
  - 3.14, -12345.1234, 100m, 1.609344e03[-0.1,+0.2]Q11573
- Lat/Long
  - @-42.42/67.123
- Extensible !-syntax for arbitrary types
  - !1000^^dbpedia:USD
- First char indicates type, rest can be parsed with regex
  - minimal dependency within KGTK on these types

KG data models summary

- Labeled property graphs
  - Nodes and relationships (edges) form labeled graphs
  - Properties on nodes and edges (key/value pairs), no nesting
- RDF
  - subject, predicate, object triples over URIs, objects can be literals
  - Triple properties through reification (4 triples instead of 1)
- RDF*
  - RDF plus nested triples about triples without reification
  - Simpler representation of triple properties, arbitrary nesting
- Wikidata
  - Property-value statements describe items (Q-nodes)
  - References describe sources of statements, etc.
  - Qualifiers describe other properties about statements (no nesting)
- KGTK
  - Quad-based edges: id, node1, label, node2
  - Anything can be a node or label
  - Arbitrarily nested edges using ids as nodes
  - https://kgtk.readthedocs.io/en/latest/data_model/

KGTK Commands

KGTK - Knowledge Graph Toolkit: Rich Support For Working With Any KG



https://github.com/usc-isi-i2/kgtk

KGTK Examples

text embeddings for all nodes

```
[ ]: !$kgtk text-embedding -i $OUT/all.tsv.gz \
--embedding-projector-metadata-path none \
--label-properties label \
--isa-properties P31 P279 P452 P106 \
--description-properties description \
--property-value P186 P17 P127 P176 P169 \
--has-properties "" \
-f kgtk_format \
--output-data-format kgtk_format \
--save-embedding-sentence \
--model bert-large-nli-cls-token \
-o "$TE" \
> "$TE"/text-embedding.tsv
```

"grep"

cast & genre edges

```
[10]: lines = !$kgtk filter -i "$TEMP"/movies.ids.tsv -p ";cast,genre;"
kgtk_to_dataframe(lines)
```

[10]:

	id	node1	label	node2
0	terminator2_jd-genre-2e6128	terminator2_jd	genre	science_fiction
1	terminator2_jd-genre-bd938c	terminator2_jd	genre	action
2	t4	terminator2_jd	cast	a_schwarzenegger
3	t6	terminator2_jd	cast	l_hamilton

```
[11]: lines = !$kgtk filter -i "$TEMP"/movies.ids.tsv -p ";;mi.*@en" --regex --match-type search
kgtk_to_dataframe(lines)
```

[11]:

	id	node1	label	node2
0	terminator2_jd-label-01de63	terminator2_jd	label	"Terminator 2"@en
1	l_hamilton-label-2b3667	l_hamilton	label	"Linda Hamilton"@en

using regex

complex queries using "Cypher"

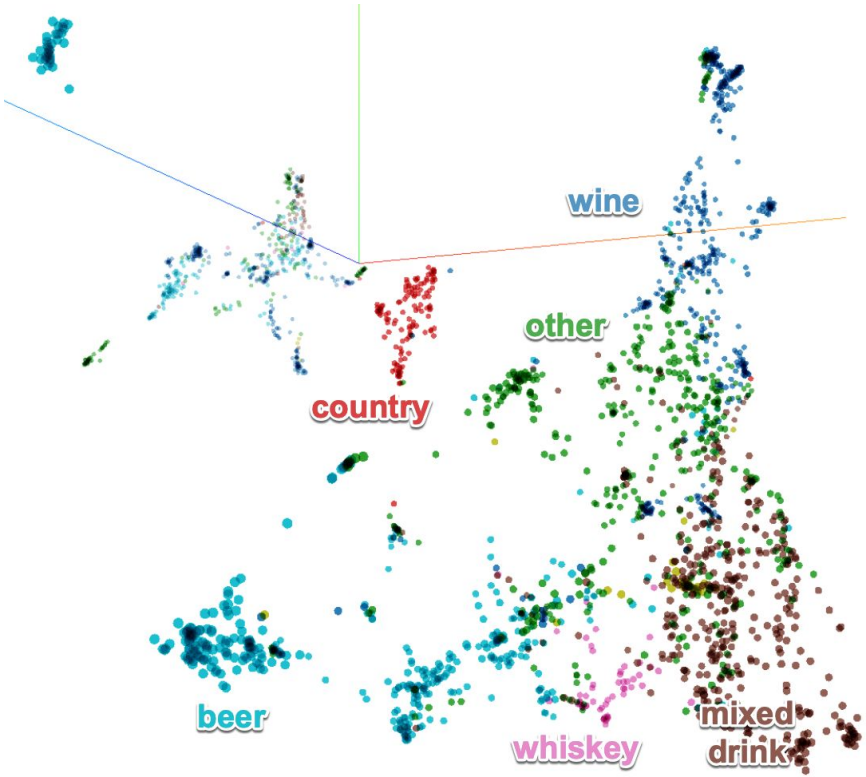
```
[9]: lines = !$kypher -i "$CLAIMS" -i "$LABEL" \
--match 'claims: (:Q82955)<-[:P106]-(n1), claims: (n1)-[p {label: property}]->(), label: (property)-[]->(property_label)' \
--return 'distinct property as property, count(property) as count, property_label as `property label`' \
--order-by 'count desc' \
--limit 50
kgtk_to_dataframe(lines)
```

[9]:

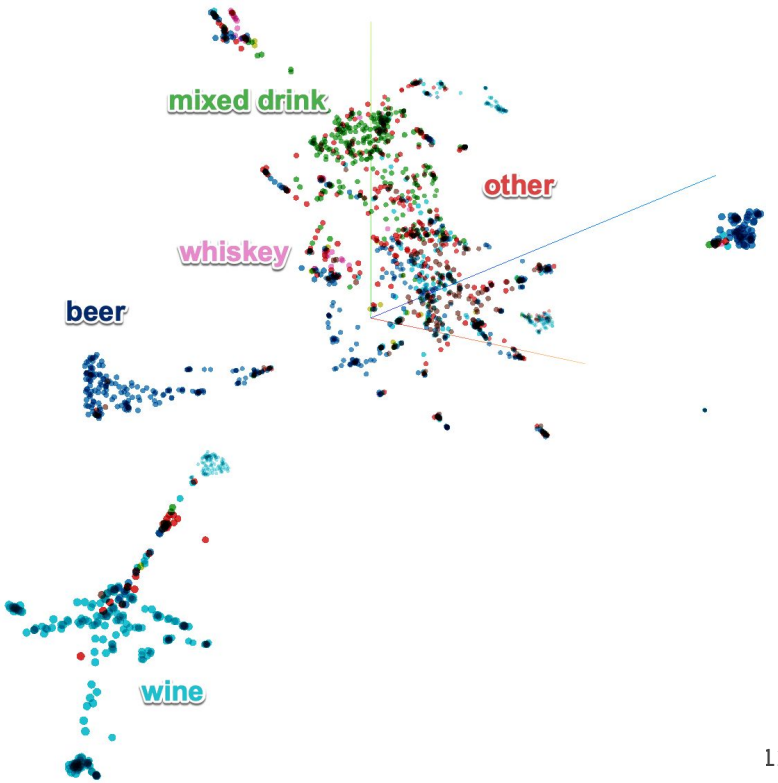
	property	count	property label
0	P106	888992	'occupation'@en
1	P39	838822	'position held'@en
2	P31	617239	'instance of'@en
3	P21	579438	'sex or gender'@en
4	P27	534961	'country of citizenship'@en

counts of properties of politicians

Text Embeddings



Graph Embeddings



KGTK Pipelines

```
kgtk import-wikidata … /
filter -p ' ; P463 ; ' /
clean /
remove-columns -c "$ignore_cols" /
graph-statistics --directed --degrees
--pagerank -o statistics.tsv
```

1. Import wikidata into KGTK
2. Select all P463 edges
3. Curate the data
4. Ignore certain columns
5. Compute PageRank & degrees

Kypher - a Query Language for KGTK

Why Kypher?

- KGTK use cases need KG assembly from a variety of sources
- KGTK supports that with a large number of ETL commands
  - Designing a proper workflow can be challenging
  - There are many commands and options to master and navigate

- **Query languages** provide flexible mix of pattern matching and computation that address many different use cases
- We had anecdotal evidence that Cypher (QL of neo4j) was easy to use by non-experts
- **K**GTK C**ypher** aka "*Kypher*" was born

KGTK Cypher aka “Kypher”: Efficient knowledge graph queries without the hassle

---

- Kypher queries translate into SQL over KGTK tsv data tables
  - KGTK data files in, result file out, pipable to/from other KGTK commands
  - Executed via SQLite3 embedded in Python
- Automatic behind-the-scenes data import, indexing and caching
  - **No need to know about DB**, no server or accounts to set up
  - Data is cached in graph cache over KGTK files for efficient reuse
- Excellent scalability
  - Tested successfully on Wikidata-scale graphs with 1.5B edges
  - Import and indexing of this size takes about **2 hours on a laptop**
    - RDF Blazegraph import of WD takes **10 days on high-end server!**
  - Creates DB file of about 200GB
  - Queries run in milliseconds to minutes depending on result sizes
- See https://kgtk.readthedocs.io/en/latest/transform/query/

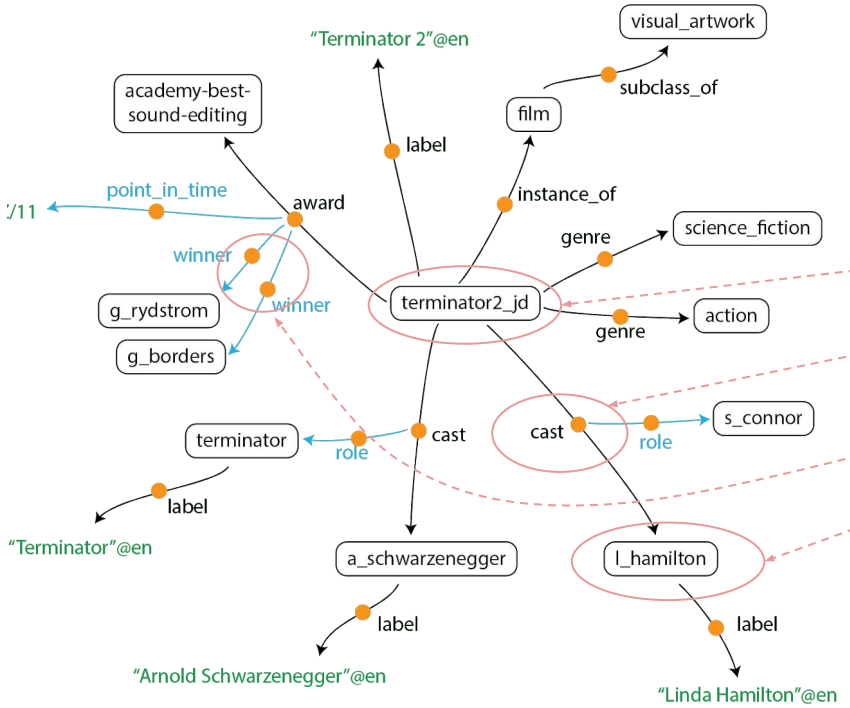
Example query:

"Find all Linda Hamilton movies that won an award and return movie, award & winner"

movies.tsv

id	node1	label	node2
	terminator2_jd	label	"Terminator 2"@en
	terminator2_jd	instance_of	film
	terminator2_jd	genre	science_fiction
	terminator2_jd	genre	action
t4	terminator2_jd	cast	a_schwarzenegger
	t4	role	terminator
t6	terminator2_jd	cast	l_hamilton
	t6	role	s_connor
t8	terminator2_jd	award	academy_best_sound_editing
	t8	point_in_time	"1992-03-30T00:00:00Z/11
	t8	winner	g_rydstrom
	t8	winner	g_borders

Specify input:




```
kgtk query -i movies.tsv
    --match (mov)-[:label]->(name),
            (mov)-[:cast]->(:l_hamilton),
            (mov)-[r:award]->(aw),
            (r)-[:winner]->(win)
    --where kgtk_lq_lang(name) = "en"
    --return name as movie, aw as award,
             win as winner
```

Specify output:

movie	award	winner
'Terminator 2'@en	academy_best_sound_editing	g_rydstrom
'Terminator 2'@en	academy_best_sound_editing	g_borders

Kypher benefits

- Simple ASCII art edge matching
  - no complex join syntax
- Simple to go from edge to qualifier edge to...
  - compare RDF reification in SPARQL
- Multiple named input graphs via files
  - efficient partitioning and indexing of data
- One efficient join instead of 1000’s of SPARQL queries
  - want info for these 100k people and how they are related
  - can’t easily ask single SPARQL query for large list of inputs
- Enables easy personal Wikidata endpoints
  - can distribute DB file and you are ready to go
  - difficult to achieve with public SPARQL endpoints
- No servers / accounts / users to set up or know about
- Seamlessly integrated into KGTK toolchain

Hands-on Kypher tutorial

- https://github.com/usc-isi-i2/kgtk-notebooks

- Alternatively, go on github.com
  - Search for **kgtk** and select **kgtk-notebooks** repository

- In the top-level repo description
  - Find “Running the notebooks in Google Colab”
  - Click on 01-kgtk-introduction.ipynb
  - Follow my instructions here, or the directions there
  - You need to **make a copy to your GDrive** (requires Google account)

- Or install KGTK on your own and copy and run notebooks yourself

Contacting us during the tutorial

slack

iswc-conf.slack.com
#kgtk-tutorial

zoom

chat, raise your hand or speak up

colab

if your colab doesn't work

- not the end of the world
- watch on zoom
- we'll try to get you going during a break

reminder:

- make a copy to your drive
- reset runtime after pip install kgtk==1.0.1