

Finding Graph Isomorphisms in GraphX and GraphFrames

Michael Malak

Source code:
<https://www.springer.com/trackback/graphisomorphism>
<https://github.com/michaelmalak>



SPARK SUBMITTING
A SPARK SUBMITTING



Isomorphisms

Definition



Finding



Summary

What is a graph isomorphism? It is a mapping between two graphs that preserves the structure of the graphs. In other words, it is a bijection between the vertices of two graphs such that two vertices are adjacent in one graph if and only if their corresponding vertices are adjacent in the other graph.



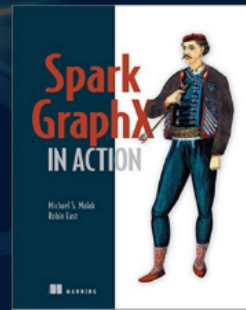
Yet Another Great Ontology (YAGO)

How It's Distributed



Finding Graph Isomorphisms in GraphX and GraphFrames

Michael Malak



Source code:

<https://www.manning.com/books/spark-graphx-in-action>

<https://gist.github.com/michaelmalak>

ORACLE®

Powered by
Spark



**Spark
summit**

SPARK SUMMIT 2016
DATA SCIENCE AND ENGINEERING AT SCALE
JUNE 6-8, 2016 SAN FRANCISCO

ORACLE®

Powered by

Spark

PaaS

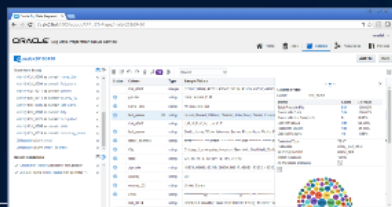

 Big Data


 GoldenGate


 Big Data Preparation

DaaS


 Data Cloud



The screenshot displays the Oracle Big Data Preparation Cloud Service interface. The main window is titled 'ORACLE Big Data Preparation Cloud Service'. The left sidebar shows a 'Transform Script' with a list of domain identities. The central pane shows a table with columns: Status, Column, Type, and Sample Values. The 'first_name' column is highlighted. The right pane shows a 'Column Profile' for 'first_name' with metrics like 'Total Rows in File' (514, 100.00%), 'Rows with Data' (514, 100.00%), 'Distinct Values' (330, 64.26%), 'Duplicate Values' (184, 35.80%), and 'Distinct Patterns' (10, 1.95%). A 'Deleted Type Rationale' table is also visible. The bottom right corner features a colorful circular logo with letters.

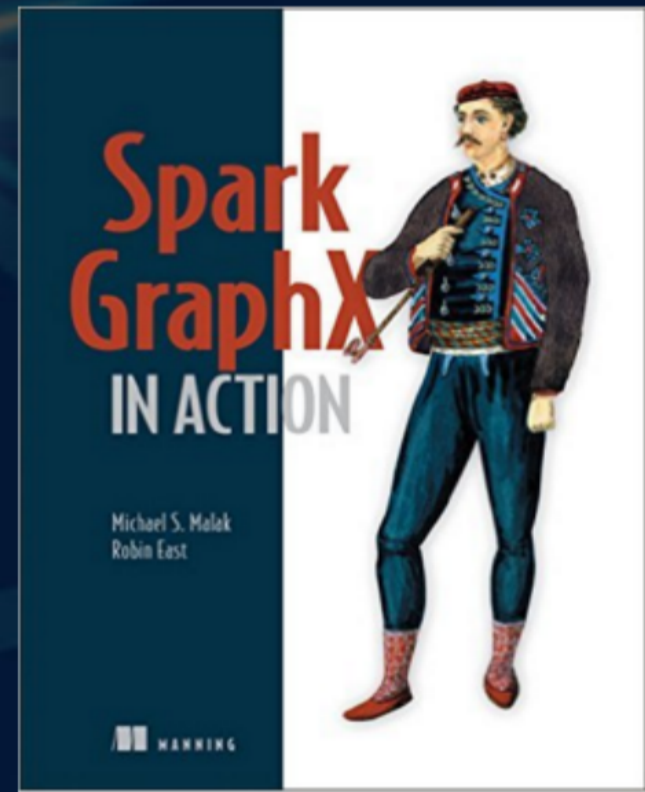
Status	Column	Type	Sample Values
	Col_0001	integer	37561; 39064; 42261; 49347; 39134; 43506; 40152; 44965
✓	gender	string	male; female; F; M
✓	name_title	string	Mr.; Ms.; Mrs.; Dr
✓	first_name	string	James; Robert; William; Charles; John; Mary; David; Joseph
	Col_0005	string	J; R; S; D; C; M; L; A; K; T
✓	last_name	string	Smith; Jones; White; Johnson; Brown; Richardson; Martin; F
✓	street_address	string	#####; #####; ###; #####; #
✓	City	string	Chicago; Los Angeles; Houston; New York; Southfield; Dallas
✓	state	string	CA; TX; MI; IL; PA; NY; FL; MA; OH; WI
✓	zipcode	string	48933; 60605; 02110; 20036; 90017; 43085; 13202; 48235;
✓	country	string	US
✓	country_02	string	United States
✓	email	string	#####@####.###; #####@#####.###; #####
	Col_0014	string	Hembeck; Cooliker; Yone1939; Drivels; Fering; Theplain; W

Metric	Count	Percent
Total Rows in File	514	100.00%
Rows with Data	514	100.00%
Rows with No Data(null)	0	0.00%
Distinct Values	330	64.26%
Duplicate Values	184	35.80%
Distinct Patterns	10	1.95%

Deleted Type Rationale	Match Accuracy
noisy_text_label	100%
name_first	100%

X and GraphX

Michael Malak



the code:

<http://www.manning.com/books/spark-graphx-in-action>

<https://gist.github.com/michaelmalak>

Finding Graph Isomorphisms in GraphX and GraphFrames

Michael Malak

Source code:
<https://github.com/michaelmalak/spark-graphx-is-action>
<https://github.com/michaelmalak>

ORACLE



SPARK SUMMIT 2016
MAY 10-11, 2016 SAN FRANCISCO



History

Timeline



Isomorphisms

Definition



Detection



Finding



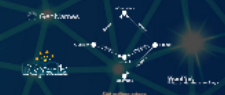
Summary

- Many frameworks (e.g., Spark, GraphX) support graph processing
- Some frameworks (e.g., Gremlin, Cypher) support graph database
- Graph processing is more scalable than graph database
- Graph database is more flexible than graph processing
- Graph processing is more efficient than graph database
- Graph database is more secure than graph processing
- Graph processing is more reliable than graph database
- Graph database is more flexible than graph processing
- Graph processing is more efficient than graph database
- Graph database is more secure than graph processing
- Graph processing is more reliable than graph database

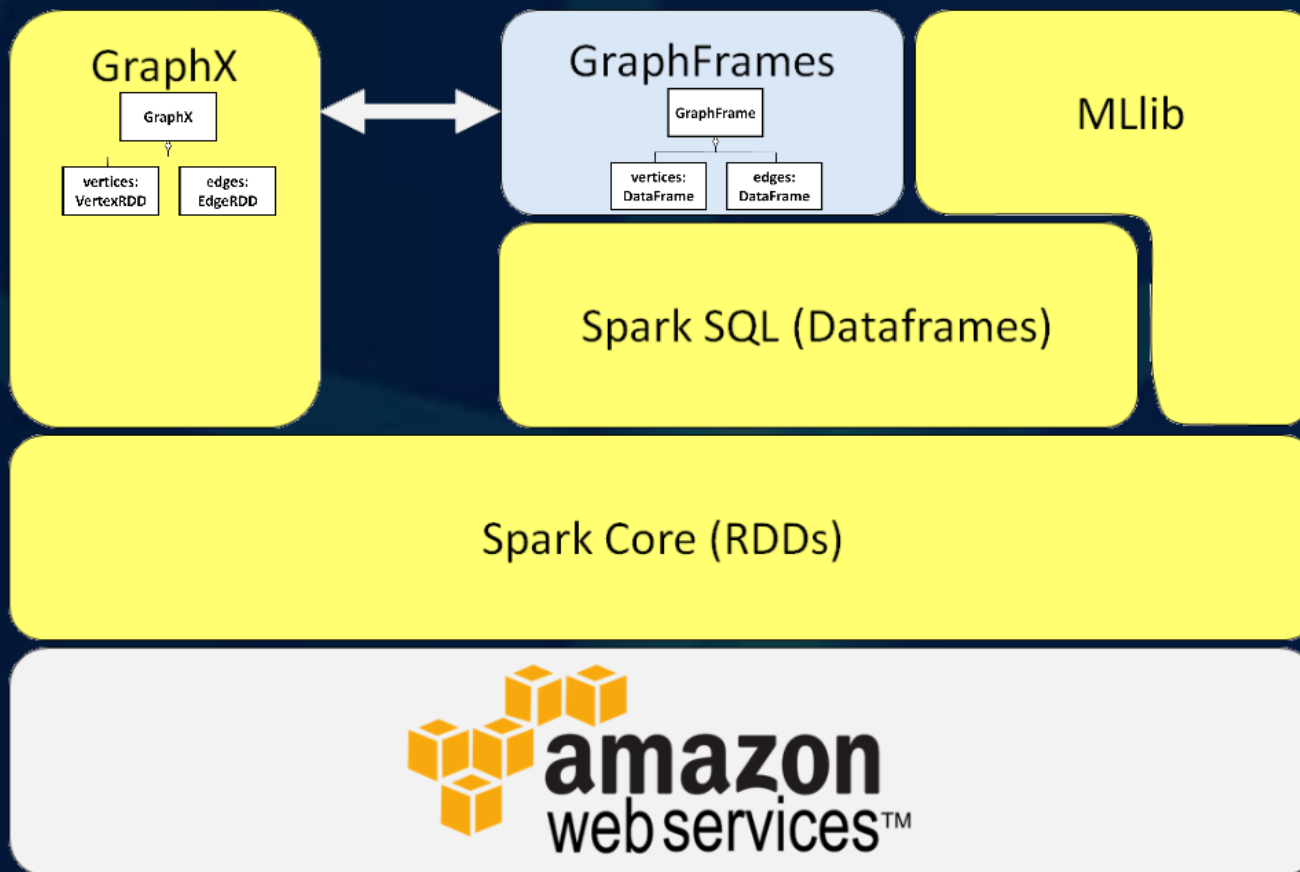


Yet Another Great Ontology (YAGO)

How It's Distributed



Stack



- Huge SSD for Spark Shuffle (instead of EBS)
 - Scaling out doesn't make straggler tasks finish sooner.
- Therefore partition both data and shuffles:

```
g.vertices.repartition(1000)
g.vertices.sqlContext.setConf("spark.sql.shuffle.partitions", "1000")
g.edges.repartition(1000)
g.edges.sqlContext.setConf("spark.sql.shuffle.partitions", "1000")
```

GraphX

GraphX

vertices:
VertexRDD

edges:
EdgeRDD

GraphFrames

GraphFrame

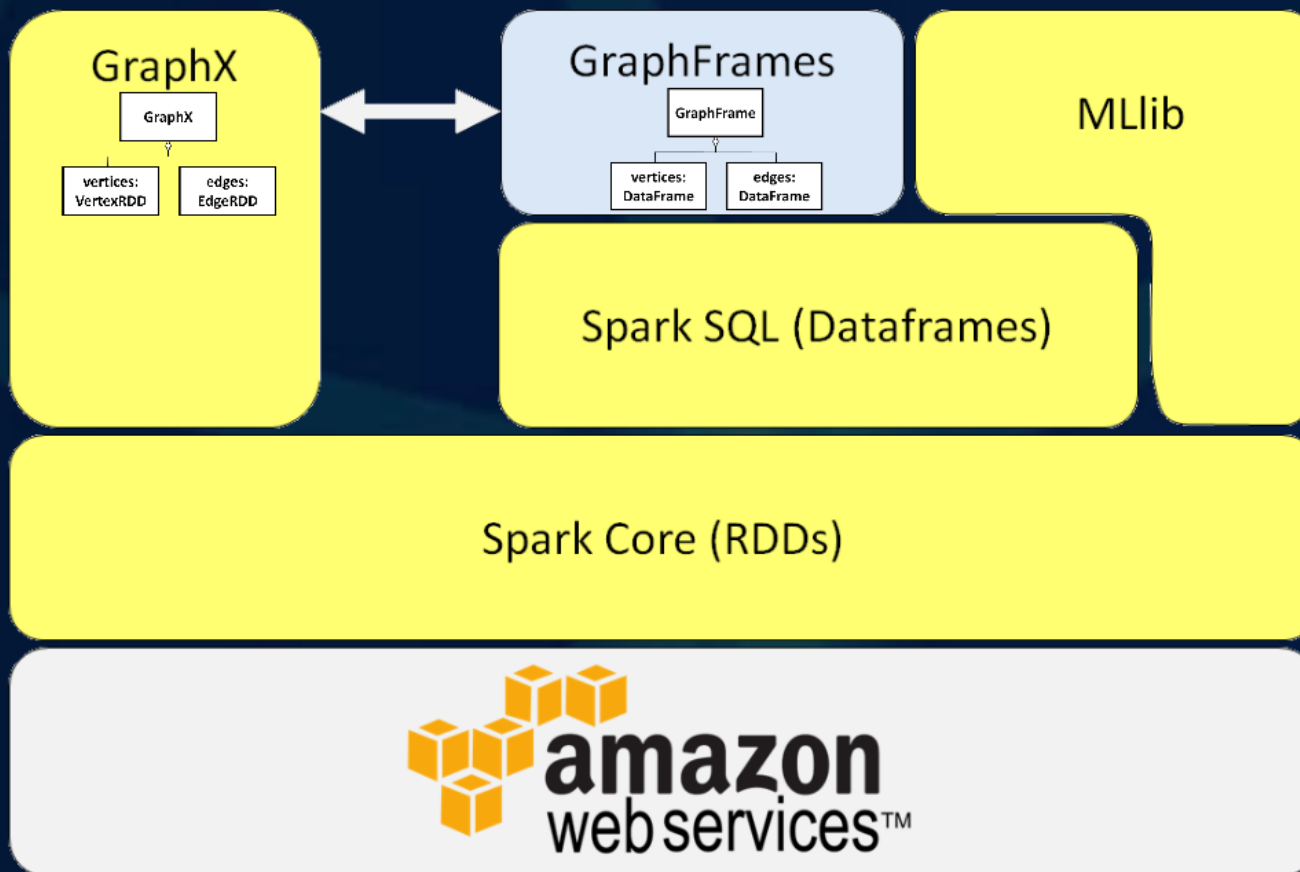
vertices:
DataFrame

edges:
DataFrame

Spark SQL (D)

Spark Core (RDDs)

Stack



- Huge SSD for Spark Shuffle (instead of EBS)
 - Scaling out doesn't make straggler tasks finish sooner.
- Therefore partition both data and shuffles:

```
g.vertices.repartition(1000)
g.vertices.sqlContext.setConf("spark.sql.shuffle.partitions", "1000")
g.edges.repartition(1000)
g.edges.sqlContext.setConf("spark.sql.shuffle.partitions", "1000")
```

Graph Processing vs. Graph Database

OLAP-like

OLTP-like

**Parallel
algorithms**

**Parallel
query**

**Single-anchor
query**

Transactions

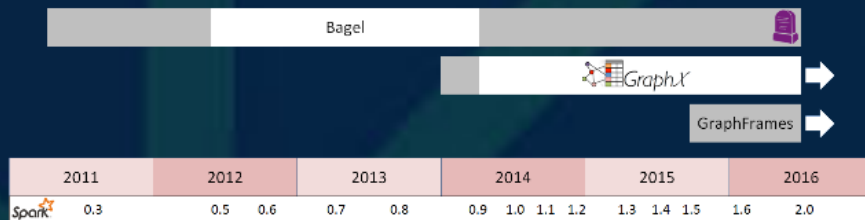


GraphFrames



History

Timeline



Features & Performance

Bagel

- Features
 - Pregel
 - Property graphs

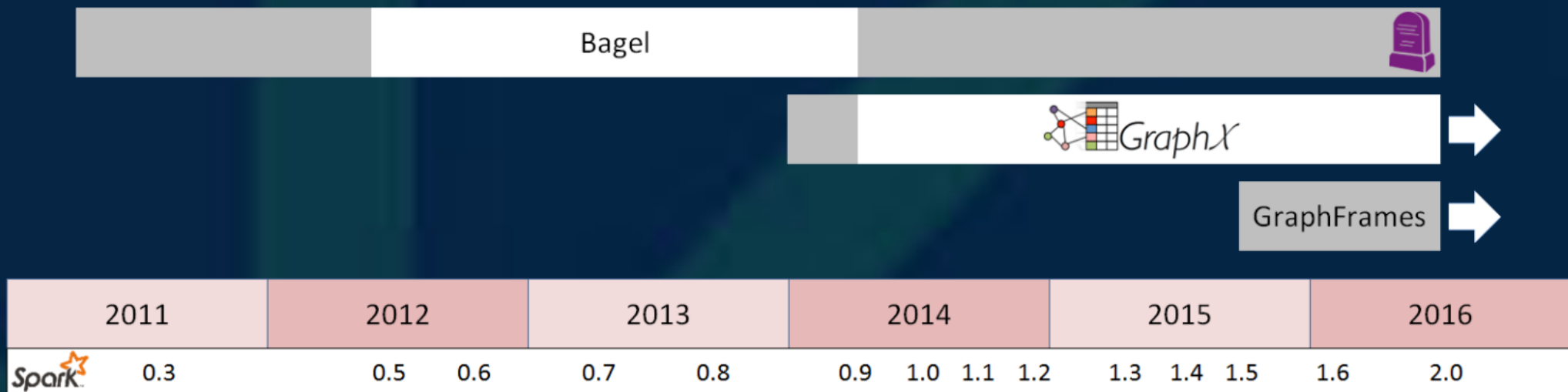


- Features
 - Pregel
 - aggregateMessages
 - True property graphs
 - Act on edge properties
- Performance
 - Routing table

GraphFrames

- Features
 - aggregateMessages
 - True property graphs
 - Act on edge properties
 - Java, Python bindings
 - Querying: Cypher subset (plus SQL from Dataframes)
- Performance
 - Dataframes
 - Catalyst
 - Tungsten
 - GraphFrames
 - Join Elimination
 - Materialized Views

Timeline



Features & Performance

Bagel

- Features
 - Pregel
 - Property graphs



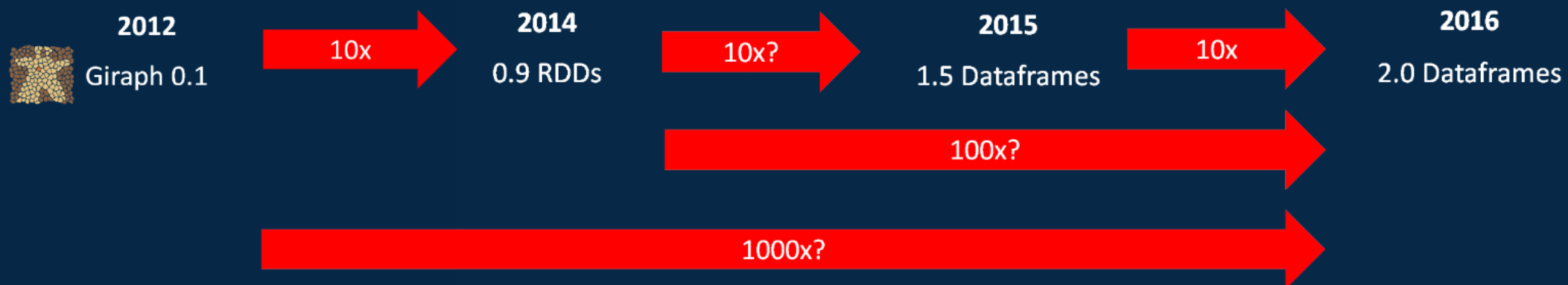
- Features
 - Pregel
 - aggregateMessages
 - True property graphs
 - Act on edge properties
- Performance
 - Routing table

GraphFrames

- Features
 - aggregateMessages
 - True property graphs
 - Act on edge properties
 - Java, Python bindings
 - Querying: Cypher subset (plus SQL from Dataframes)
- Performance
 - Dataframes
 - Catalyst
 - Tungsten
 - GraphFrames
 - Join Elimination
 - Materialized Views

Performance

Expected from Dataframes (Catalyst & Tungsten)



Actual (anecdotal)

1.5 GraphX
(RDDs)
`readRDF()`

8x

1.5 Graphframes
(Dataframes)
`readRDF()`

Before:

- Spark 2.0 improvements
- Upcoming Graphframes improvements

Features & Performance

Bagel

- Features
 - Pregel
 - Property graphs



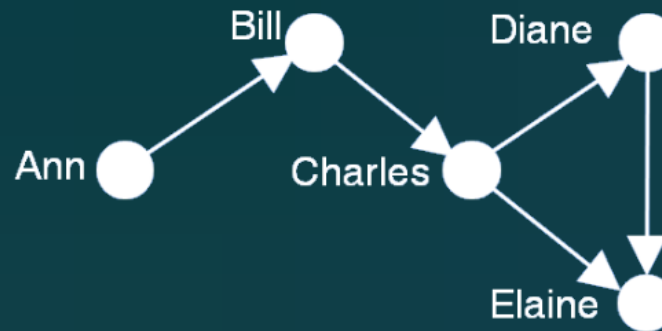
- Features
 - Pregel
 - aggregateMessages
 - True property graphs
 - Act on edge properties
- Performance
 - Routing table

GraphFrames

- Features
 - aggregateMessages
 - True property graphs
 - Act on edge properties
 - Java, Python bindings
 - Querying: Cypher subset (plus SQL from Dataframes)
- Performance
 - Dataframes
 - Catalyst
 - Tungsten
 - GraphFrames
 - Join Elimination
 - Materialized Views

Querying

Who are the friends of friends of Ann?



GraphX

```
val g2 = g.outerJoinVertices(g.aggregateMessages[Int](
  ctx => if (ctx.srcAttr == "Ann" &&
    ctx.attr == "knows") ctx.sendToDst(1),
  math.max(_,_))((vid, vname, d) =>
    (vname, d.getOrElse(0)))
g2.outerJoinVertices(g2.aggregateMessages[Int](
  ctx => if (ctx.srcAttr._2 == 1 &&
    ctx.attr == "knows") ctx.sendToDst(2),
  math.max(_,_))((vid, vname, d) =>
    (vname, d.getOrElse(0))).
vertices.map(_._2).filter(_._2 == 2)
  .map(_._1._1).collect
```

Neo4j

```
MATCH (ann { name: 'Ann' })-[:knows*2..2]-(p)
RETURN p
```

GraphFrames

```
val gf = GraphFrame.fromGraphX(myGraph)
gf.find("(u)-[e1]->(v); (v)-[e2]->(w)")
  .filter("e1.attr = 'is-friends-with' AND " +
    "e2.attr = 'is-friends-with' AND " +
    "u.attr='Ann'")
  .select("w.attr")
  .collect
  .map(_(0).toString)
```



Elaine

Neo4j

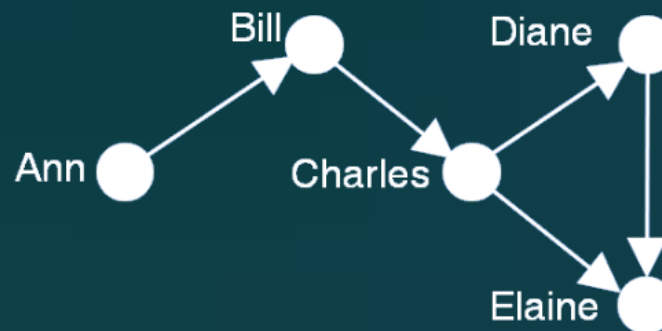
```
MATCH (ann { name: 'Ann' })-[:knows*2..2]-(p)
RETURN p
```

GraphFrames

```
val gf = GraphFrame.fromGraphX(myGraph)
gf.find("(u)-[e1]->(v); (v)-[e2]->(w)")
  .filter("e1.attr = 'is-friends-with' AND " +
    "e2.attr = 'is-friends-with' AND " +
    "u.attr='Ann'")
  .select("w.attr")
  .collect
  .map(_(0).toString)
```

Querying

Who are the friends of friends of Ann?



GraphX

```
val g2 = g.outerJoinVertices(g.aggregateMessages[Int](
  ctx => if (ctx.srcAttr == "Ann" &&
    ctx.attr == "knows") ctx.sendToDst(1),
  math.max(_,_))((vid, vname, d) =>
    (vname, d.getOrElse(0)))
g2.outerJoinVertices(g2.aggregateMessages[Int](
  ctx => if (ctx.srcAttr._2 == 1 &&
    ctx.attr == "knows") ctx.sendToDst(2),
  math.max(_,_))((vid, vname, d) =>
    (vname, d.getOrElse(0))).
vertices.map(_._2).filter(_._2 == 2)
  .map(_._1._1).collect
```

Neo4j

```
MATCH (ann { name: 'Ann' })-[:knows*2..2]-(p)
RETURN p
```

GraphFrames

```
val gf = GraphFrame.fromGraphX(myGraph)
gf.find("(u)-[e1]->(v); (v)-[e2]->(w)")
  .filter("e1.attr = 'is-friends-with' AND " +
    "e2.attr = 'is-friends-with' AND " +
    "u.attr='Ann'")
  .select("w.attr")
  .collect
  .map(_(0).toString)
```

Features & Performance

Bagel

- Features
 - Pregel
 - Property graphs



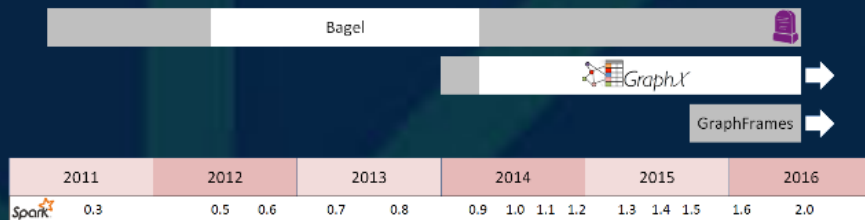
- Features
 - Pregel
 - aggregateMessages
 - True property graphs
 - Act on edge properties
- Performance
 - Routing table

GraphFrames

- Features
 - aggregateMessages
 - True property graphs
 - Act on edge properties
 - Java, Python bindings
 - Querying: Cypher subset (plus SQL from Dataframes)
- Performance
 - Dataframes
 - Catalyst
 - Tungsten
 - GraphFrames
 - Join Elimination
 - Materialized Views

History

Timeline



Features & Performance

Bagel

- Features
 - Pregel
 - Property graphs

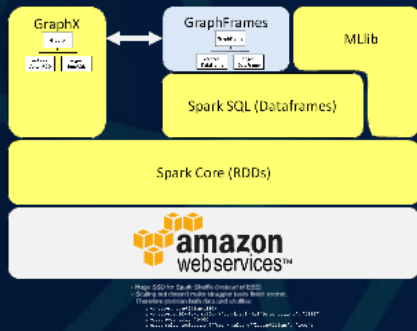


- Features
 - Pregel
 - aggregateMessages
 - True property graphs
 - Act on edge properties
- Performance
 - Routing table

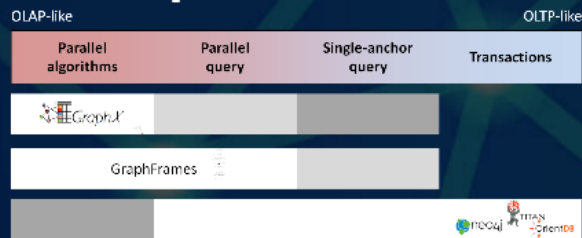
GraphFrames

- Features
 - aggregateMessages
 - True property graphs
 - Act on edge properties
 - Java, Python bindings
 - Querying: Cypher subset (plus SQL from Dataframes)
- Performance
 - Dataframes
 - Catalyst
 - Tungsten
 - GraphFrames
 - Join Elimination
 - Materialized Views

Stack



Graph Processing vs. Graph Database



History

Timeline



Features & Performance



Isomorphisms

Definition



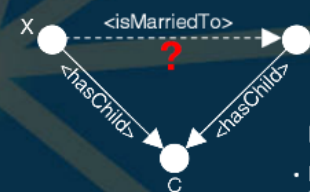
from cs.stonybrook.edu

Detection



(Not the problem
we're (I'm) trying
to solve)

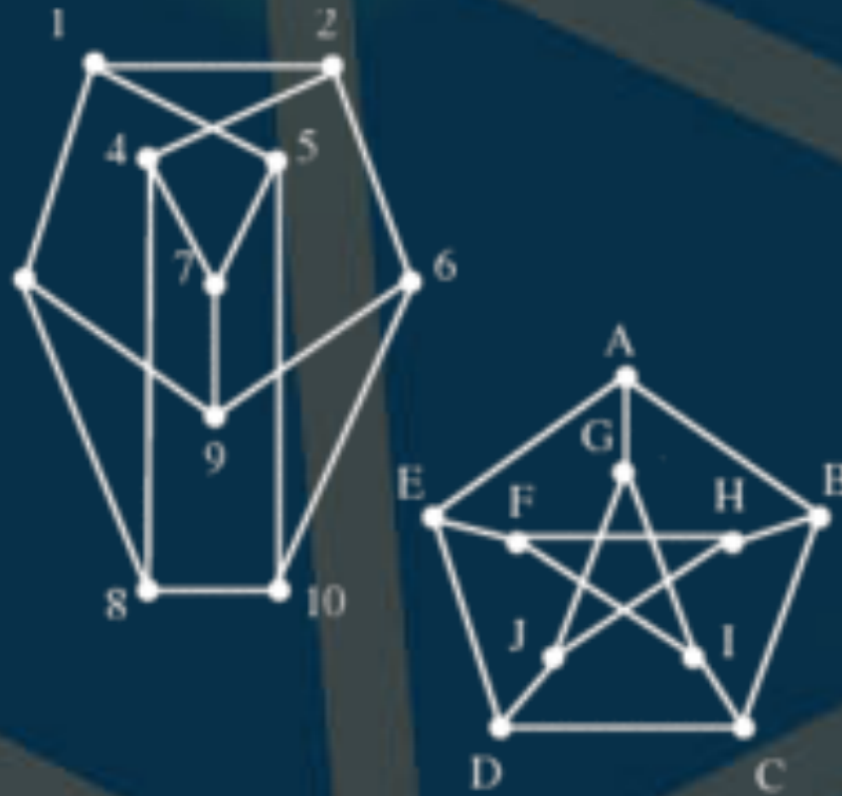
Finding



Rule Mining

- Inductive Logic Programming
- Network Analysis
- Social Network Analysis
- Social Link Analysis

Definition



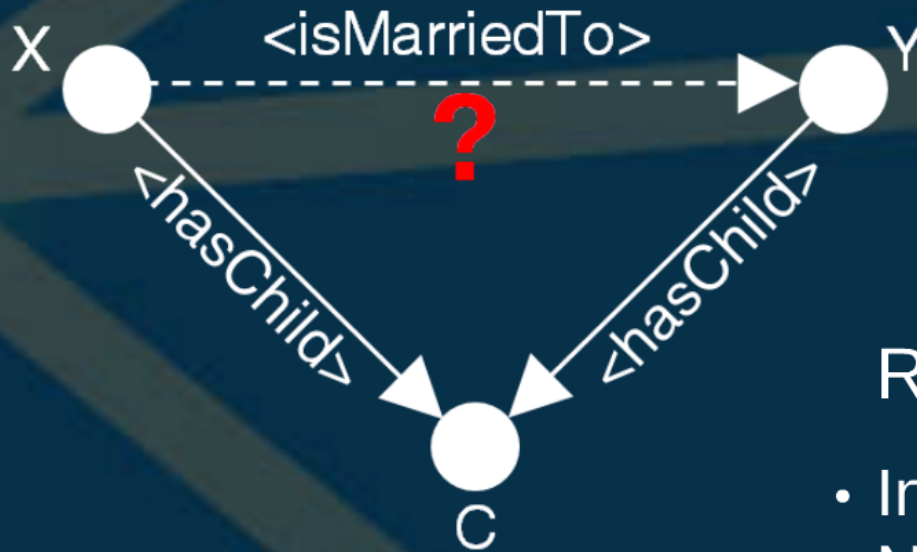
from cs.stonybrook.edu

Detection



(Not the problem
we're (I'm) trying
to solve)

Finding



Rule Mining

- Inductive Logic Programming
- Network Analysis
- Social Network Analysis
- Social Link Analysis

Isomorphisms

Definition



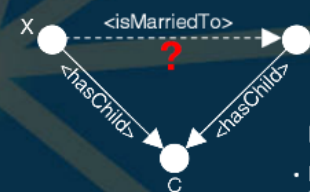
from cs.stonybrook.edu

Detection



(Not the problem
we're (I'm) trying
to solve)

Finding



Rule Mining

- Inductive Logic Programming
- Network Analysis
- Social Network Analysis
- Social Link Analysis

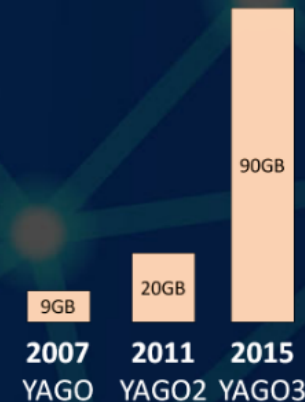
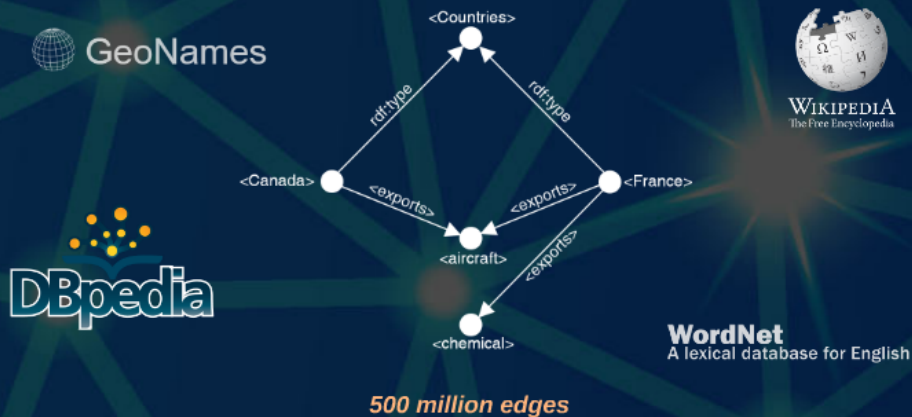


yago

select knowledge

Yet Another Great Ontology (YAGO)

How It's Distributed



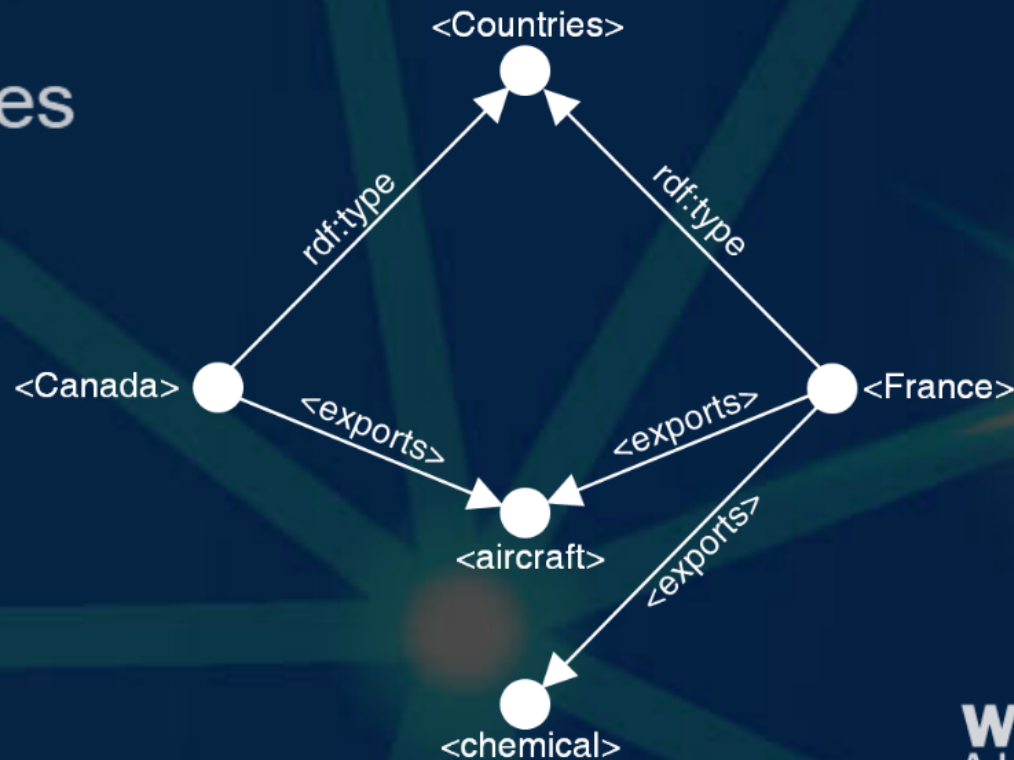
Yet Another Great Ontology (YAGO)



GeoNames



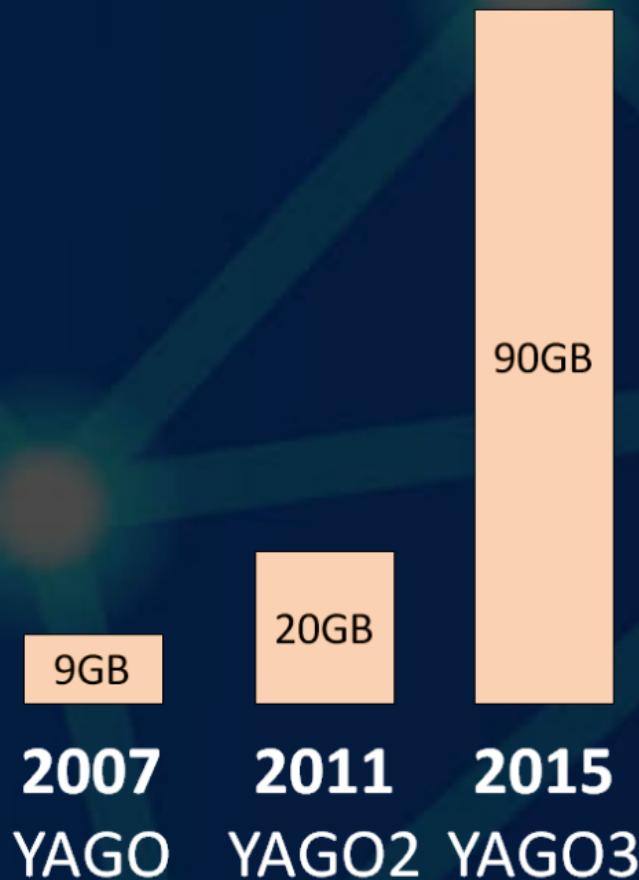
WIKIPEDIA
The Free Encyclopedia



WordNet
A lexical database for English

500 million edges

How It's Distributed



76 .tsv Files in YAGO3

yagoFacts.tsv: 425 MB

```
<relation Within_Range_Edge type <lat geo> <lat geo> <lat geo> <lat geo>
<relation_Creator_Label> <name> <lat geo> <lat geo> <lat geo> <lat geo>
```

yagoLabels.tsv: 3348 MB

```
<Francisco_I_Crispo> rdfs:label "Francisco Ier Crispo" rdfs:
<Monroe_City_Schools> skos:prefLabel "Monroe City Schools" rdfs:
```

yagoTaxonomy.tsv: 62 MB

```
<lat geo> <lat geo> <lat geo> <lat geo> <lat geo> <lat geo> <lat geo> <lat geo>
<lat geo> <lat geo> <lat geo> <lat geo> <lat geo> <lat geo> <lat geo> <lat geo>
```

Functions Needed To Read YAGO

```
readRdf()
1. Split the data into chunks
2. Read the data into memory
3. Join the data into a single table

mergeGraphs()
1. Read the data into memory
2. Join the data into a single table
3. Write the data to disk
```

76 .tsv Files in YAGO3

yagoFacts.tsv: 425 MB

```
<Friedrich_Wilhelm_Joseph_Schelling> <influences> <Alexander_von_Humboldt>  
<Jefferson_County,_Texas> <owns> <Jack_Brooks_Regional_Airport>
```

yagoLabels.tsv: 3348 MB

```
<Francesco_I_Crispo> rdfs:label "Francesco Ier Crispo"@fra  
<Monroe_City_Schools> skos:prefLabel "Monroe City Schools"@eng
```

yagoTaxonomy.tsv: 62 MB

```
<wikicat_Birds_of_Tunisia> rdfs:subClassOf <wordnet_bird_101503061>  
<wordnet_knot_113885836> rdfs:subClassOf <wordnet_distorted_shape_113867276>
```

Functions Needed To Read YAGO

readRdf()

1. ZipWithIndex: Number the vertices
2. Join: convert source vertex name to ID
3. Join: convert destination vertex name to ID

GraphX

```
def readRdf(sc:org.apache.spark.SparkContext, filename:String) = {  
  val r = sc.textFile(filename).map(_.split("\t"))  
  val v = r.map(_(1)).union(r.map(_(3))).distinct.zipWithIndex  
  Graph(v.map(_._swap),  
        r.map(x => (x(1),x(2),x(3))))  
    .join(v)  
    .map(x => (x._2._1._2,(x._2._2,x._2._1._1)))  
    .join(v)  
    .map(x => new Edge(x._2._1._1, x._2._2, x._2._1._2))  
}
```

GraphFrames

```
def readRdf(sc:org.apache.spark.SparkContext, filename:String) = {  
  val r = sc.textFile(filename).map(_.split("\t"))  
  val v = r.map(_(1)).union(r.map(_(3))).distinct.zipWithIndex.map(  
    x => Row(x._2,x._1)).cache  
  val stv = StructType(StructField("id",LongType) ::  
    StructField("attr",StringType) :: Nil)  
  val vdf = sqlContext.createDataFrame(v,stv).cache  
  val str = StructType(StructField("rdfId",StringType) ::  
    StructField("subject",StringType) ::  
    StructField("predicate",StringType) ::  
    StructField("object",StringType) :: Nil)  
  val odr = sqlContext.createDataFrame(r.map(Row.fromSeq(_)),str)  
    .join(vdf, $"subject" === $"attr")  
    .selectExpr("id AS src", "predicate", "object")  
    .join(vdf, $"object" === $"attr")  
    .selectExpr("src", "id AS dst", "predicate AS attr")  
  v.unpersist(false)  
  GraphFrame(vdf,odr)  
}
```

mergeGraphs()

GraphX

```
def mergeGraphs(g1:Graph[String,String], g2:Graph[String,String]) = {  
  val v = g1.vertices.map(_._2).union(g2.vertices.map(_._2)).distinct  
    .zipWithIndex  
  def edgesWithNewVertexIds(g:Graph[String,String]) =  
    Rdd(g.edges  
      .map(et => (et.srcAttr, (et.attr,et.dstAttr)))  
      .join(v)  
      .map(x => (x._2._1._2, (x._2._2,x._2._1._1)))  
      .join(v)  
      .map(x => new Edge(x._2._1._1,x._2._2,x._2._1._2))  
      .Graph(x.map(_._swap))  
    ).edgesWithNewVertexIds(g1).union(edgesWithNewVertexIds(g2)))  
}
```

GraphFrames

```
def mergeGraphs(g1:GraphFrame, g2:GraphFrame) = {  
  val union = g1.vertices.sqlContext.createDataFrame(  
    g1.vertices.select($"attr").unionAll(g2.vertices.select($"attr"))  
    .distinct().asInstanceOf[  
      .zipWithIndex.map(t => Row.fromTuple(t._swap)),  
      new StructType(Array(StructField("id",LongType),  
        StructField("attr",StringType)))  
    ).cache  
  def edgesWithNewVertexIds(gf:GraphFrame) = {  
    gf.triplets  
      .join(union, $"src.attr" === $"attr")  
      .selectExpr("id AS src", "edge.attr AS edgeattr",  
        "dst.attr AS dstattr")  
      .join(union, $"dstattr" === $"attr")  
      .selectExpr("src", "id AS dst", "edgeattr AS attr")  
  }  
  GraphFrame(union, edgesWithNewVertexIds(g1).unionAll(  
    edgesWithNewVertexIds(g2)))  
}
```

readRdf()

1. ZipWithIndex: Number the vertices
2. Join: convert source vertex name to ID
3. Join: convert destination vertex name to ID

GraphX

```
def readRdf(sc:org.apache.spark.SparkContext, filename:String) = {  
  val r = sc.textFile(filename).map(_.split("\t"))  
  val v = r.map(_(1)).union(r.map(_(3))).distinct.zipWithIndex  
  Graph(v.map(_._swap),  
    r.map(x => (x(1),(x(2),x(3))))  
    .join(v)  
    .map(x => (x._2._1._2,(x._2._2,x._2._1._1)))  
    .join(v)  
    .map(x => new Edge(x._2._1._1, x._2._2, x._2._1._2)))  
}
```

GraphFrames

```
def readRdfDf(sc:org.apache.spark.SparkContext, filename:String) = {  
  val r = sc.textFile(filename).map(_.split("\t"))  
  val v = r.map(_(1)).union(r.map(_(3))).distinct.zipWithIndex.map(  
    x => Row(x._2,x._1)).cache  
  val stv = StructType(StructField("id",LongType) ::  
    StructField("attr",StringType) :: Nil)  
  val vdf = sqlContext.createDataFrame(v,stv).cache  
  val str = StructType(StructField("rdfId",StringType) ::  
    StructField("subject",StringType) ::  
    StructField("predicate",StringType) ::  
    StructField("object",StringType) :: Nil)  
  val edf = sqlContext.createDataFrame(r.map(Row.fromSeq(_)),str)  
    .join(vdf, $"subject" === $"attr")  
    .selectExpr("id AS src", "predicate", "object")  
    .join(vdf, $"object" === $"attr")  
    .selectExpr("src", "id AS dst", "predicate AS attr")  
  
  v.unpersist(false)  
  GraphFrame(vdf,edf)  
}
```


mergeGraphs()

GraphX

```
def mergeGraphs(g1:Graph[String,String], g2:Graph[String,String]) = {
  val v = g1.vertices.map(_._2).union(g2.vertices.map(_._2)).distinct
    .zipWithIndex
  def edgesWithNewVertexIds(g:Graph[String,String]) =
    g.triplets
      .map(et => (et.srcAttr, (et.attr,et.dstAttr)))
      .join(v)
      .map(x => (x._2._1._2, (x._2._2,x._2._1._1)))
      .join(v)
      .map(x => new Edge(x._2._1._1,x._2._2,x._2._1._2))
  Graph(v.map(_._swap),
    edgesWithNewVertexIds(g1).union(edgesWithNewVertexIds(g2)))
}
```

GraphFrames

```
def mergeGraphs(g1:GraphFrame, g2:GraphFrame) = {
  val vunion = g1.vertices.sqlContext.createDataFrame(
    g1.vertices.select($"attr").unionAll(g2.vertices.select($"attr"))
      .distinct.rdd.map(_(0))
      .zipWithIndex.map(t => Row.fromTuple(t.swap)),
    new StructType(Array(StructField("id",LongType),
      StructField("attr",StringType)))
  ).cache

  def edgesWithNewVertexIds(gf:GraphFrame) = {
    gf.triplets
      .join(vunion, $"src.attr" === $"attr")
      .selectExpr("id AS src","edge.attr AS edgeattr",
        "dst.attr AS dstattr")
      .join(vunion, $"dstattr" === $"attr")
      .selectExpr("src","id AS dst","edgeattr AS attr")
  }

  GraphFrame(vunion, edgesWithNewVertexIds(g1).unionAll(
    edgesWithNewVertexIds(g2)))
}
```

Functions Needed To Read YAGO

readRdf()

1. ZipWithIndex: Number the vertices
2. Join: convert source vertex name to ID
3. Join: convert destination vertex name to ID

GraphX

```
def readRdf(sc:org.apache.spark.SparkContext, filename:String) = {  
  val r = sc.textFile(filename).map(_.split("\t"))  
  val v = r.map(_(1)).union(r.map(_(3))).distinct.zipWithIndex  
  Graph(v.map(_._swap),  
        r.map(x => (x(1),x(2),x(3))))  
    .join(v)  
    .map(x => (x._2._1._2,(x._2._2,x._2._1._1)))  
    .join(v)  
    .map(x => new Edge(x._2._1._1, x._2._2, x._2._1._2))  
}
```

GraphFrames

```
def readRdf(sc:org.apache.spark.SparkContext, filename:String) = {  
  val r = sc.textFile(filename).map(_.split("\t"))  
  val v = r.map(_(1)).union(r.map(_(3))).distinct.zipWithIndex.map(  
    x => Row(x._2,x._1)).cache  
  val stv = StructType(StructField("id",LongType) ::  
    StructField("attr",StringType) :: Nil)  
  val vdf = sqlContext.createDataFrame(v,stv).cache  
  val str = StructType(StructField("rdfId",StringType) ::  
    StructField("subject",StringType) ::  
    StructField("predicate",StringType) ::  
    StructField("object",StringType) :: Nil)  
  val odr = sqlContext.createDataFrame(r.map(Row.fromSeq(_)),str)  
    .join(vdf, $"subject" === $"attr")  
    .selectExpr("id AS src", "predicate", "object")  
    .join(vdf, $"object" === $"attr")  
    .selectExpr("src", "id AS dst", "predicate AS attr")  
  v.unpersist(false)  
  GraphFrame(vdf,odr)  
}
```

mergeGraphs()

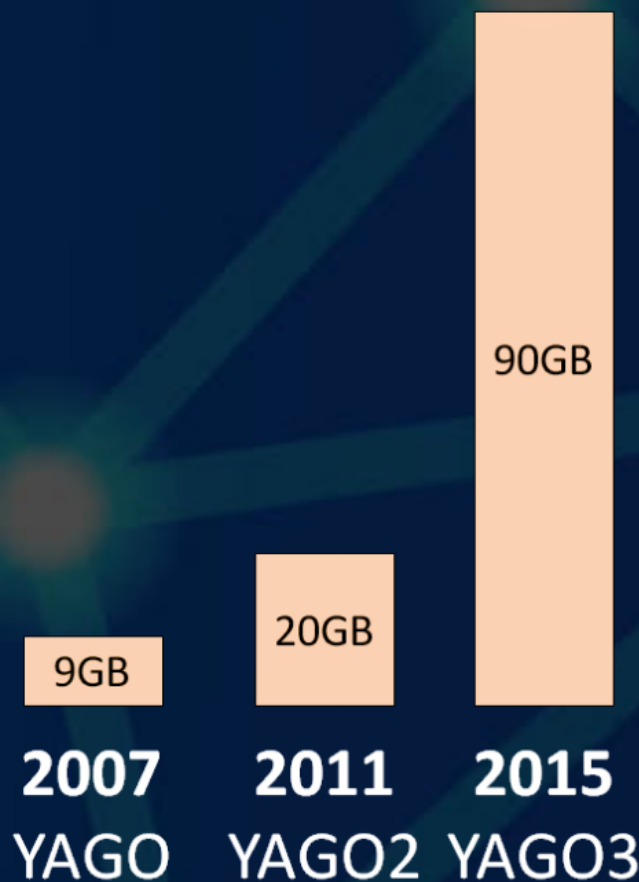
GraphX

```
def mergeGraphs(g1:Graph[String,String], g2:Graph[String,String]) = {  
  val v = g1.vertices.map(_._2).union(g2.vertices.map(_._2)).distinct  
    .zipWithIndex  
  def edgesWithNewVertexIds(g:Graph[String,String]) =  
    Rdd(g.edges  
      .map(et => (et.srcAttr, (et.attr,et.dstAttr)))  
      .join(v)  
      .map(x => (x._2._1._2, (x._2._2,x._2._1._1)))  
      .join(v)  
      .map(x => new Edge(x._2._1._1,x._2._2,x._2._1._2))  
      .Graph(x.map(_._swap))  
    ).edgesWithNewVertexIds(g1).union(edgesWithNewVertexIds(g2)))  
}
```

GraphFrames

```
def mergeGraphs(g1:GraphFrame, g2:GraphFrame) = {  
  val union = g1.vertices.sqlContext.createDataFrame(  
    g1.vertices.select($"attr").unionAll(g2.vertices.select($"attr"))  
    .distinct().asInstanceOf[  
      .zipWithIndex.map(t => Row.fromTuple(t._swap)),  
      new StructType(Array(StructField("id",LongType),  
        StructField("attr",StringType)))  
    ).cache  
  def edgesWithNewVertexIds(gf:GraphFrame) = {  
    gf.triplets  
      .join(union, $"src.attr" === $"attr")  
      .selectExpr("id AS src", "edge.attr AS edgeattr",  
        "dst.attr AS dstattr")  
      .join(union, $"dstattr" === $"attr")  
      .selectExpr("src", "id AS dst", "edgeattr AS attr")  
  }  
  GraphFrame(union, edgesWithNewVertexIds(g1).unionAll(  
    edgesWithNewVertexIds(g2)))  
}
```

How It's Distributed



76 .tsv Files in YAGO3

yagoFacts.tsv: 425 MB

```
<relation Within_Range_Edge type <lat geo> <lat geo> <lat geo> <lat geo>
<relation_Creator_Label> <name> <lat geo> <lat geo> <lat geo> <lat geo>
```

yagoLabels.tsv: 3348 MB

```
<Francisco_I_Crispo> rdfs:label "Francisco Ier Crispo" #fra
<Monroe_City_Schools> skos:prefLabel "Monroe City Schools" #eng
```

yagoTaxonomy.tsv: 62 MB

```
<lat geo> <lat geo> <lat geo> <lat geo> <lat geo> <lat geo> <lat geo> <lat geo>
<lat geo> <lat geo> <lat geo> <lat geo> <lat geo> <lat geo> <lat geo> <lat geo>
```

Functions Needed To Read YAGO

```
readRdf()
1. Split YAGO into chunks for reading
2. Read YAGO into memory
3. Read YAGO into memory

mergeGraphs()
1. Read YAGO into memory
2. Read YAGO into memory
3. Read YAGO into memory
```

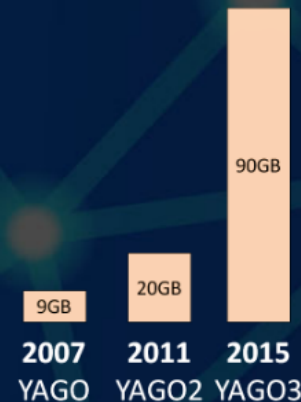
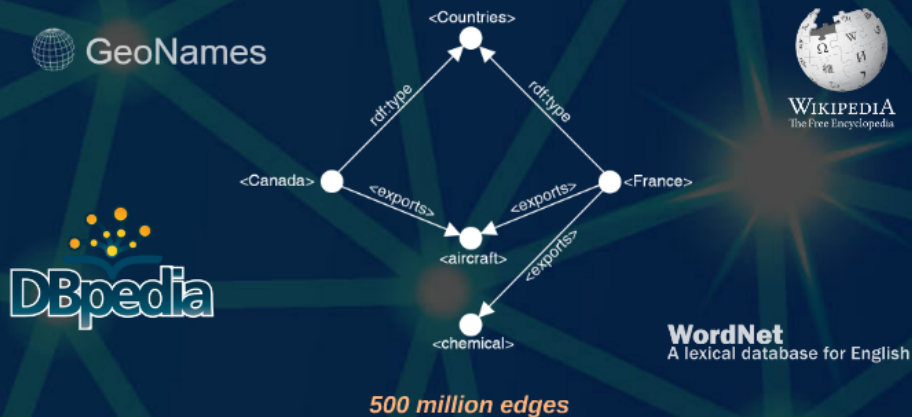


yago

select knowledge

Yet Another Great Ontology (YAGO)

How It's Distributed



Finding Graph Isomorphisms in GraphX and GraphFrames

Michael Malak

Source code:
<https://github.com/michaelmalak/spark-graphx-is-action>
<https://github.com/michaelmalak>

ORACLE



SPARK SUMMIT 2016
MAY 10-11, 2016 SAN FRANCISCO, CA



History



Isomorphisms

Definition



Detection



Finding



Summary

- Many frameworks (e.g., Spark, GraphX) support graph processing
- Some frameworks (e.g., Gremlin, Cypher) support graph database
- Graph processing is faster than graph database
- Graph database is more scalable than graph processing
- Graph processing is more flexible than graph database
- Graph database is more secure than graph processing
- Graph processing is more reliable than graph database

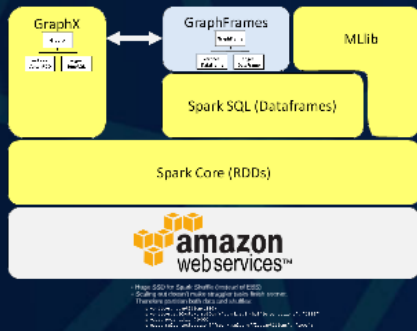


Yet Another Great Ontology (YAGO)

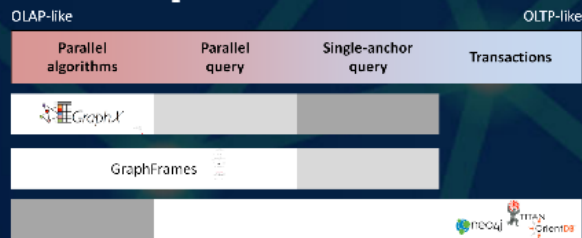
How It's Distributed



Stack



Graph Processing vs. Graph Database



History

Timeline



Features & Performance



Graph Processing vs. Graph Database

OLAP-like

OLTP-like

**Parallel
algorithms**

**Parallel
query**

**Single-anchor
query**

Transactions

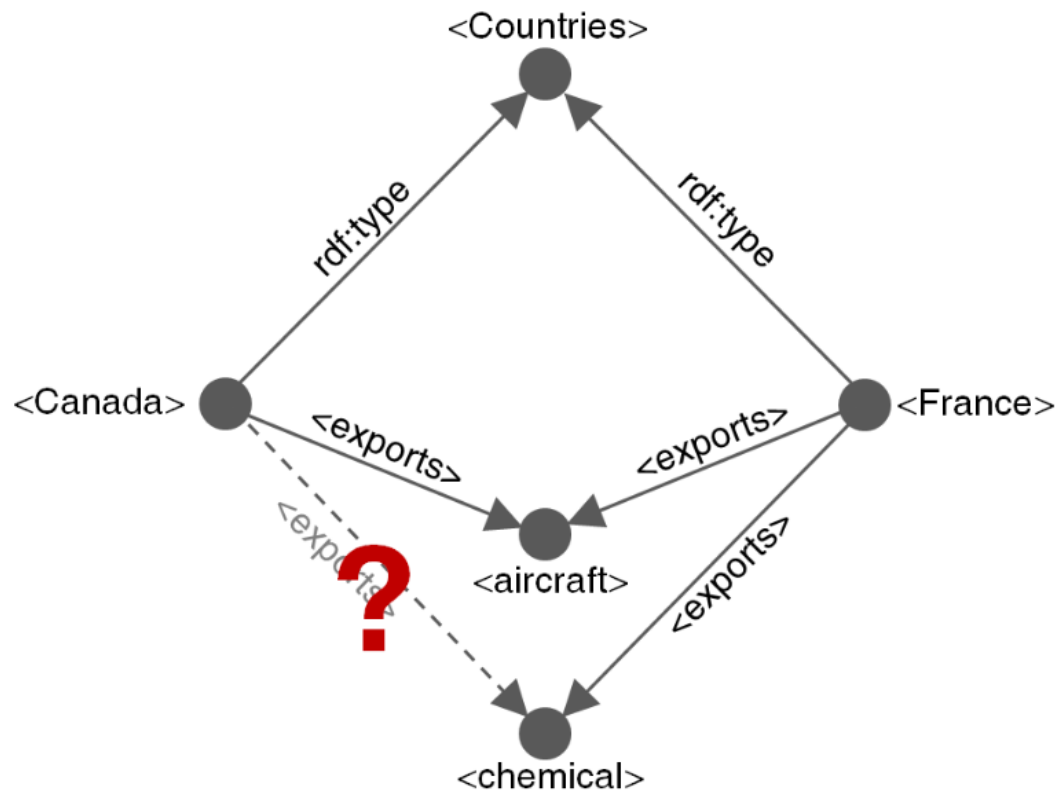


GraphFrames

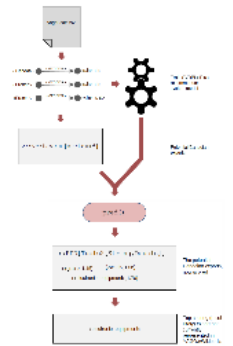




Country Exports Example



Dataflow



Results

```

val oid = g.vertices.filter(_._2 == "Canada").first._1
val r = r.map(v => (v,prod(v,oid,0)))

val msgKey = new Ordering("Tuple2"over"val", "oid")
override def compare(x: VertexId, Double, y: VertexId, Double): Int =
  Ordering[Double].compare(x, y, y, x)

g.vertices.filter(_._1 == msgKey).collect

res0: Array[(Long,spark.package.graphs.ventailid, String)] =
  Array((1021488, "electronic_equipment_1032982480"))
  
```

```

val r = g.vertices.filter(_._2 == "Canada").first._1
val r = r.map(v => (v,prod(v,oid,0)))

val msgKey = new Ordering("Tuple2"over"val", "oid")
override def compare(x: VertexId, Double, y: VertexId, Double): Int =
  Ordering[Double].compare(x, y, y, x)

g.vertices.filter(_._1 == msgKey).collect

res0: Array[(Long,spark.package.graphs.ventailid, String)] =
  Array((1021488, "electronic_equipment_1032982480"))
  
```

```

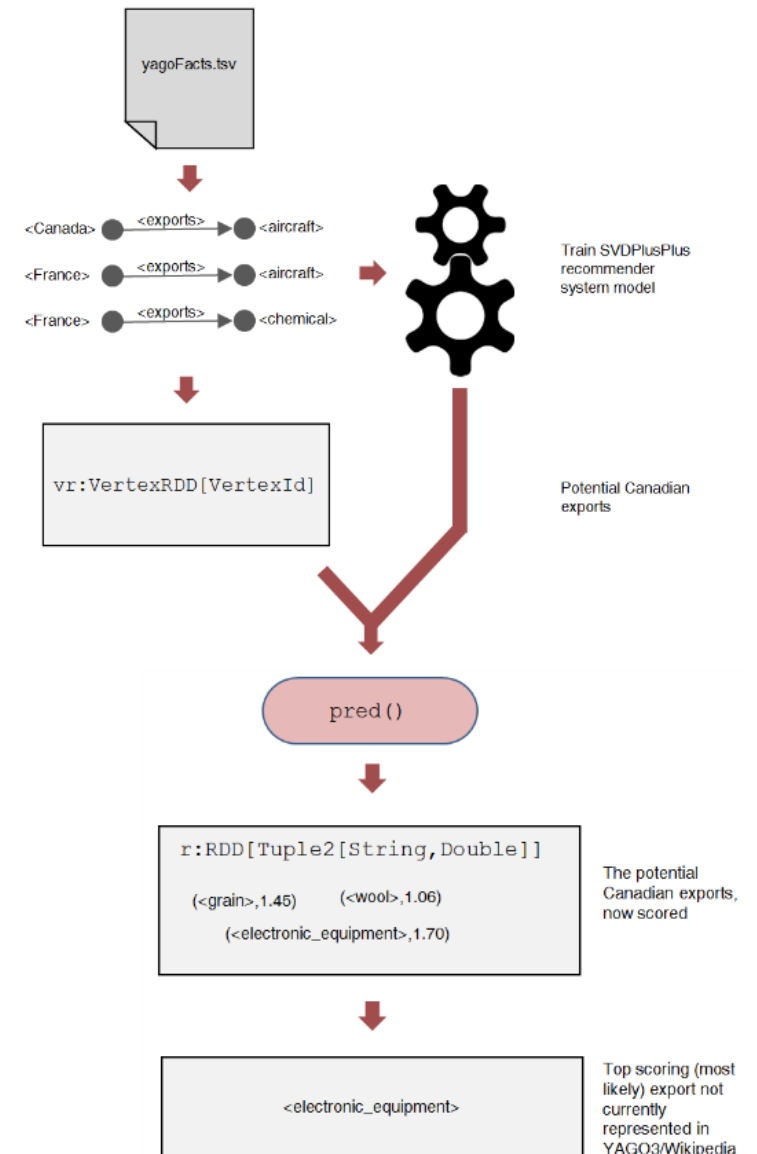
val r = g.vertices.filter(_._2 == "Canada").first._1
val r = r.map(v => (v,prod(v,oid,0)))

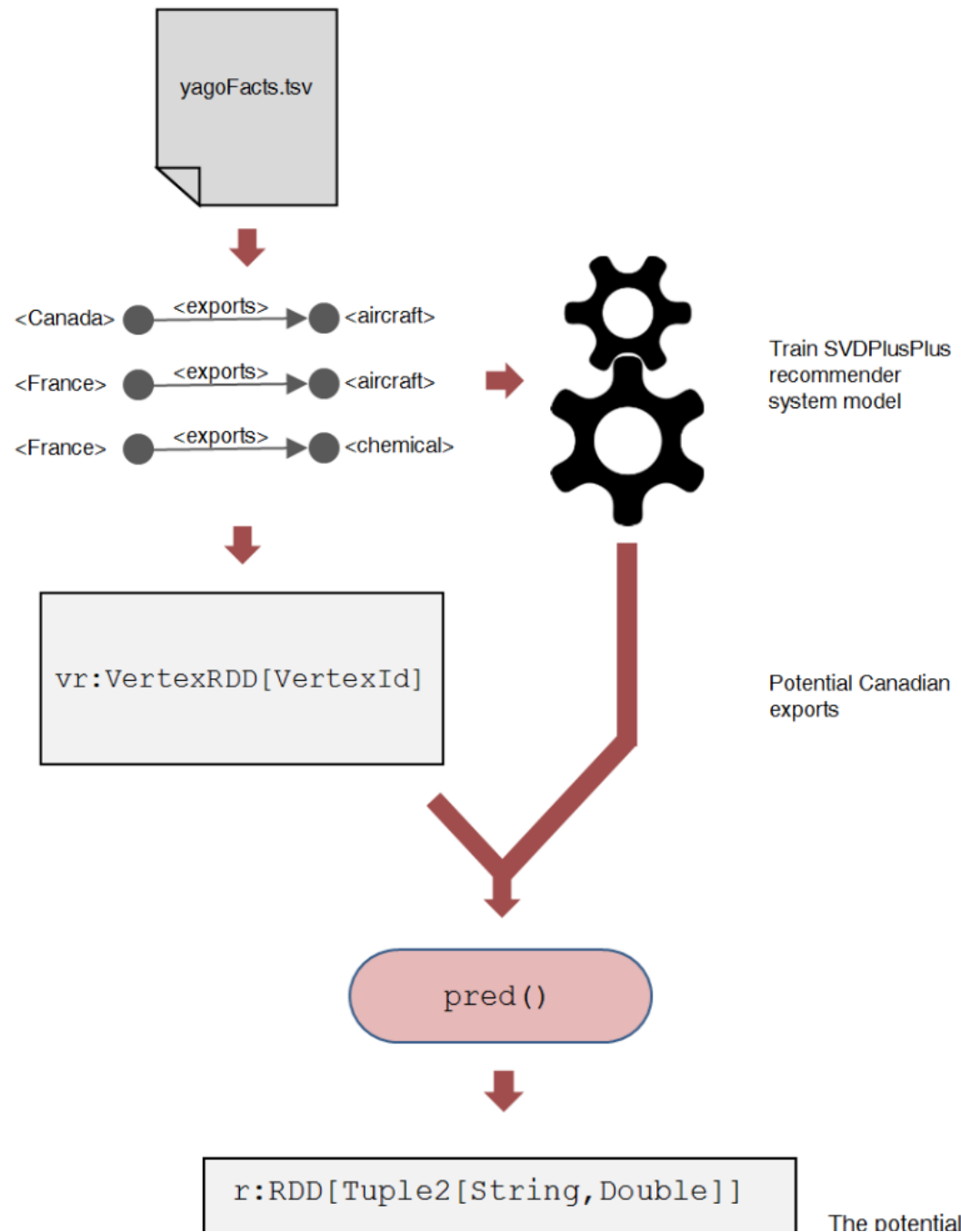
val msgKey = new Ordering("Tuple2"over"val", "oid")
override def compare(x: VertexId, Double, y: VertexId, Double): Int =
  Ordering[Double].compare(x, y, y, x)

g.vertices.filter(_._1 == msgKey).collect

res0: Array[(Long,spark.package.graphs.ventailid, String)] =
  Array((1021488, "electronic_equipment_1032982480"))
  
```

Dataflow





`vr:VertexRDD[VertexId]`

Potential Canadian exports



`pred()`



`r:RDD[Tuple2[String,Double]]`

`(<grain>,1.45)` `(<wool>,1.06)`
`(<electronic_equipment>,1.70)`

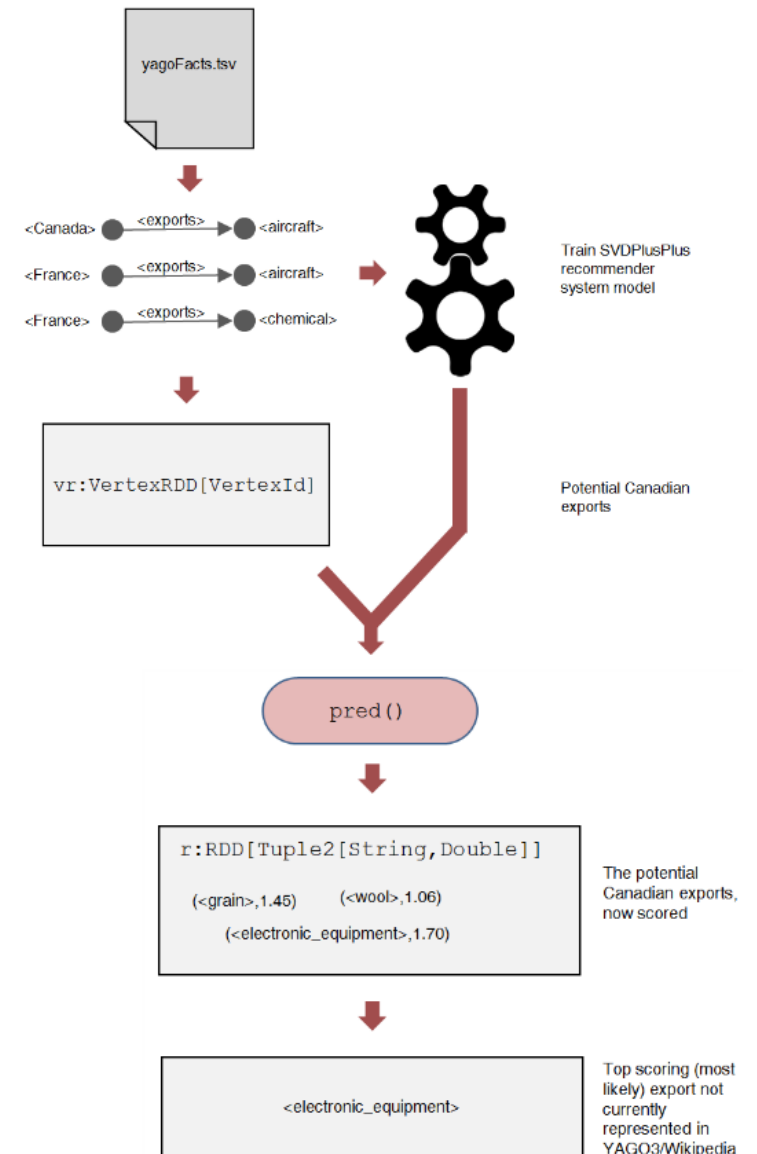
The potential Canadian exports, now scored



`<electronic_equipment>`

Top scoring (most likely) export not currently represented in YAGO3/Wikipedia

Dataflow



Results

```
val cid = gf.vertices.filter(_._2 == "<Canada>").first._1
val r = vr.map(v => (v, pred(vm, mean, cid, v)))

val maxKey = r.max()(new Ordering[Tuple2[VertexId, Double]]() {
  override def compare(x: (VertexId, Double), y: (VertexId, Double)): Int =
    Ordering[Double].compare(x._2, y._2)
})._1

gf.vertices.filter(_._1 == maxKey).collect

res0: Array[(org.apache.spark.graphx.VertexId, String)] =
  Array((1721488, <wordnet_electronic_equipment_103278248>))
```

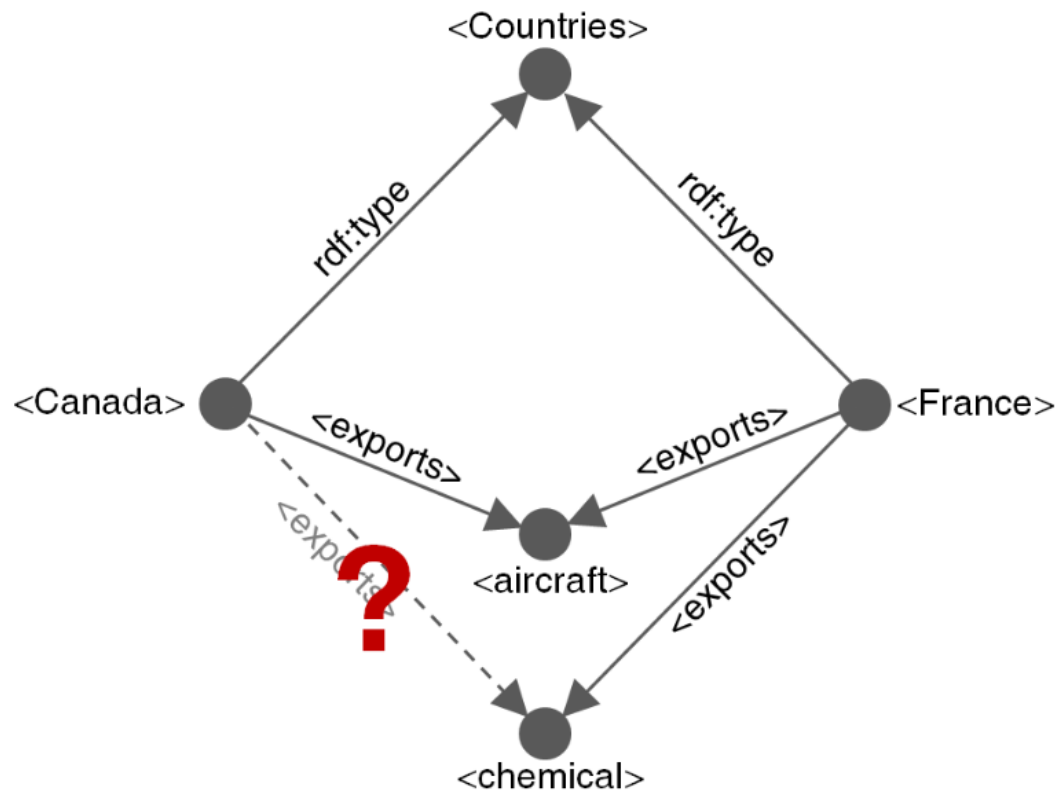
worldstopexports.com

1. Oil: US\$77.8 billion (19% of total exports)
2. Vehicles: \$60 billion (14.7%)
3. Machines, engines, pumps: \$31.1 billion (7.6%)
4. Gems, precious metals: \$19 billion (4.7%)
5. Electronic equipment: \$13.2 billion (3.2%)
6. Plastics: \$12.5 billion (3.1%)
7. Aircraft, spacecraft: \$12.3 billion (3%)
8. Wood: \$11.8 billion (2.9%)
9. Aluminum: \$8.2 billion (2%)
10. Paper: \$7.7 billion (1.9%)

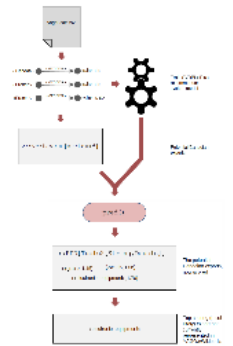
```
grep "<exports>" yagoFacts.tsv | grep "<Canada>"
<id_lwrxlwu_dv6_1pgb7a4> <Canada> <exports> <wordnet_aluminum_114627820>
<id_lwrxlwu_dv6_j2l8e6> <Canada> <exports> <wordnet_electricity_111449907>
<id_lwrxlwu_dv6_t6wmo1> <Canada> <exports> <wordnet_lumber_114943580>
<id_lwrxlwu_dv6_jhowo0> <Canada> <exports> <wordnet_natural_gas_114960090>
<id_lwrxlwu_dv6_s9bzqx> <Canada> <exports> <wordnet_aircraft_102686568>
<id_lwrxlwu_dv6_12fzkkg> <Canada> <exports> <wordnet_plastic_114592610>
```



Country Exports Example



Dataflow



Results

```

val oid = g.vertices.filter(_._2 == "Canada").first._1
val r = r.map(v => (v,prod(v,oid,0)))

val msgMap = r.collect { case (v,prod) => (v,prod) }
override def domain(x: VertexId, Double): Int =
  Ordering[Double].compare(x, y, y)

r.vertices.filter(_._1 == msgMap).collect
res0: Array[(org.apache.spark.graphs.VertexId, String)] =
  Array((1021488, "electronic_equipment_1032982480"))
  
```

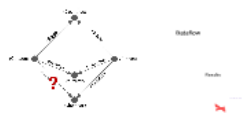
```

val r = r.map(v => (v,prod(v,oid,0)))
val msgMap = r.collect { case (v,prod) => (v,prod) }
override def domain(x: VertexId, Double): Int =
  Ordering[Double].compare(x, y, y)

r.vertices.filter(_._1 == msgMap).collect
res0: Array[(org.apache.spark.graphs.VertexId, String)] =
  Array((1021488, "electronic_equipment_1032982480"))
  
```



Country Exports Example



Graph Processing vs. Graph Database

OLAP-like

OLTP-like

**Parallel
algorithms**

**Parallel
query**

**Single-anchor
query**

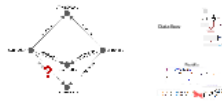
Transactions



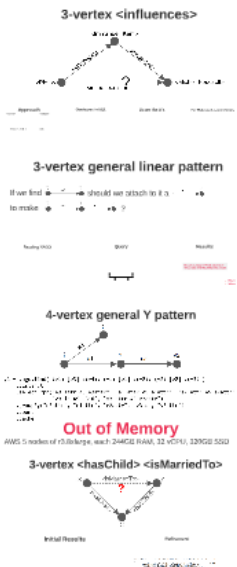
GraphFrames



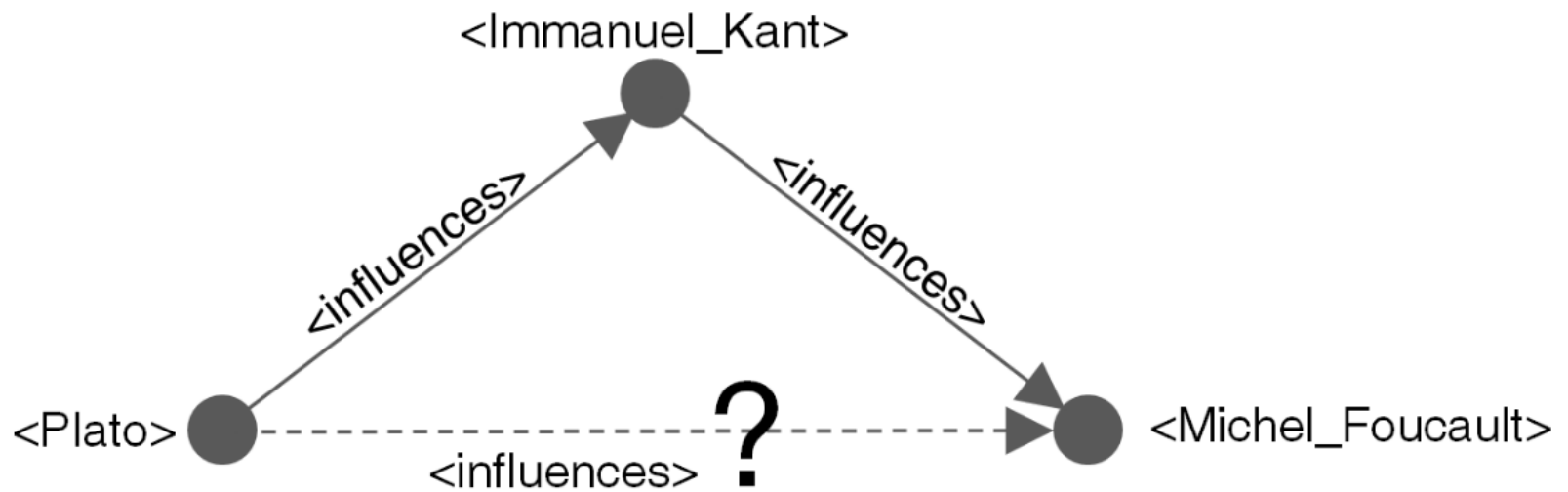
Country Exports Example



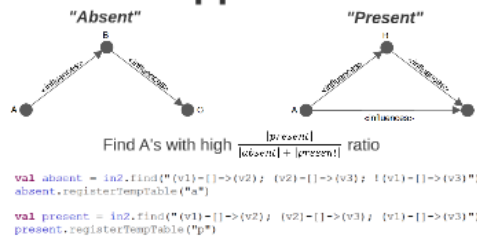
GraphFrames



3-vertex <influences>



Approach



Outdegree in SQL

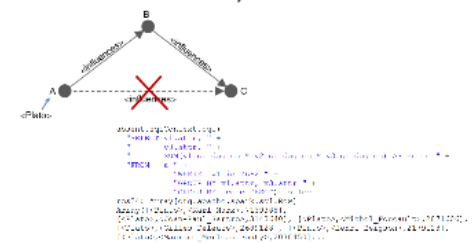
```
ygraph <"influences"> ygraph::Influencer ygraph::Influenced{Influencer, In-
val in = newEdgeID(wr, "ygraph::Influences.tlv")
is_edges.registerTermTable(in)
is_vertices.registerTermTable("v")

val in2 = CreateFrame(is.vertices.getContext, get
    "AFFECT v.id,"
    " FIRST(ATTR) AS attr,"
    " COUNT(v) AS outdegree ")
    "FROM
    " IN
    " JOIN
    " ON (IN.outdegree) * I
    " GROUP BY v.id" ) as_in2
is_edges
is_vertices
```

Score the A's

[illegible]

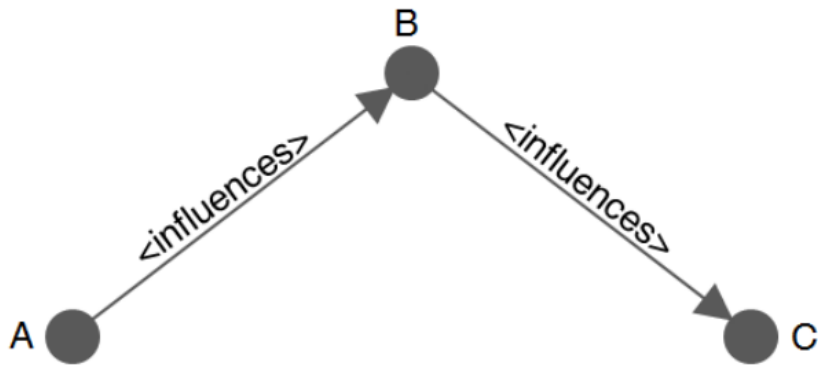
For Plato as A, score the C's



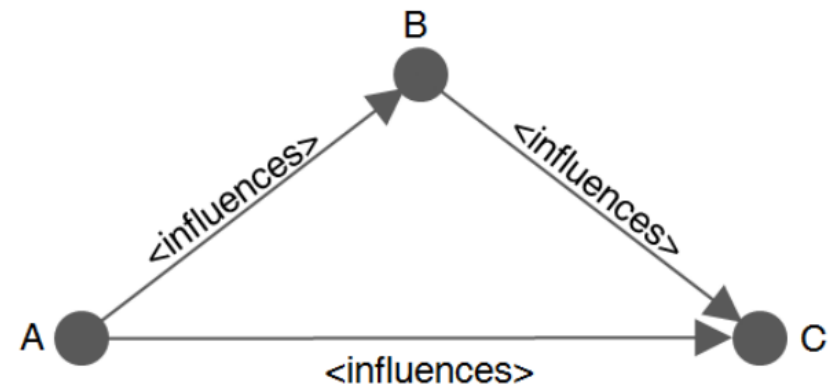
3-vertex general linear pattern

Approach

"Absent"



"Present"



Find A's with high $\frac{|present|}{|absent| + |present|}$ ratio

```
val absent = in2.find("(v1)-[]->(v2); (v2)-[]->(v3); !(v1)-[]->(v3)")
absent.registerTempTable("a")
```

```
val present = in2.find("(v1)-[]->(v2); (v2)-[]->(v3); (v1)-[]->(v3)")
present.registerTempTable("p")
```

Outdegree in SQL

```
grep "<influences>" yagoFacts.tsv >yagoFactsInfluences.tsv
```

```
val in = readRdfDf(sc, "yagoFactsInfluences.tsv")
```

```
in.edges.registerTempTable("e")
```

```
in.vertices.registerTempTable("v")
```

```
val in2 = GraphFrame(in.vertices.sqlContext.sql(
    "SELECT v.id, " +
    "        FIRST(v.attr) AS attr, " +
    "        COUNT(*) AS outdegree " +
    "FROM    v " +
    "JOIN    e " +
    "  ON    v.id=e.src " +
    "GROUP BY v.id").cache,
    in.edges)
```

Score the A's

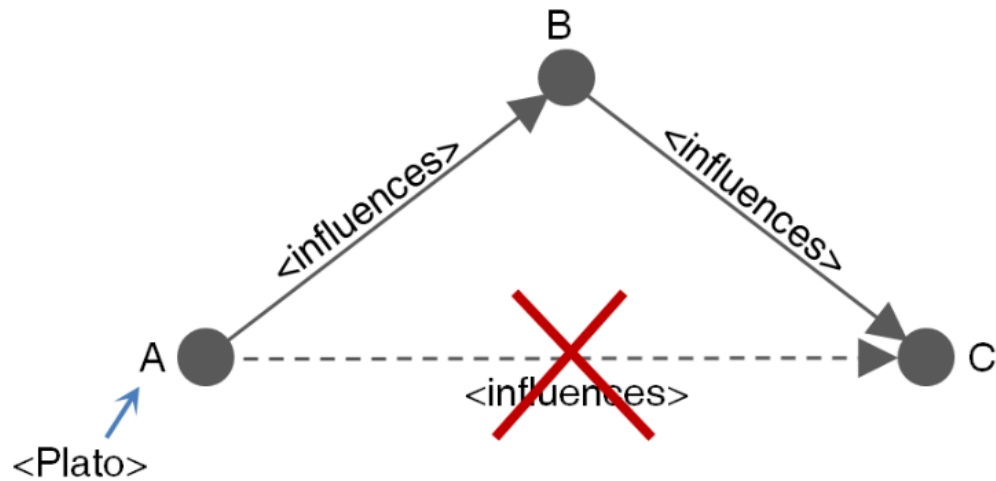
```
absent.sqlContext.sql(
  "SELECT v1 an, " +
  "      SUM(v1.outdegree * v2.outdegree * v3.outdegree) AS ac " +
  "FROM   a " +
  "GROUP BY v1").registerTempTable("aa")

present.sqlContext.sql(
  "SELECT v1 pn, " +
  "      SUM(v1.outdegree * v2.outdegree * v3.outdegree) AS pc " +
  "FROM   p " +
  "GROUP BY v1").registerTempTable("pa")

absent.sqlContext.sql("SELECT an, " +
  "      ac * pc/(ac+pc) AS score " +
  "FROM   aa " +
  "JOIN   pa " +
  "ON     an=pn " +
  "ORDER BY score DESC").show
```

```
+-----+-----+
|              an|              score|
+-----+-----+
| [7662,<Plato>,102]| 3.822406412297308E7|
|[10648,<Aristotle...|3.2961326121938106E7|
|[4959,<Immanuel_K...|2.6445857520978764E7|
|[2961,<Georg_Wilh...|2.1092802441273782E7|
|[9304,<Baruch_Spi...|1.4513392385496272E7|
|[12217,<René_Desc...|1.2407118036818413E7|
|[12660,<Johann_Wo...|1.0109121178397963E7|
|[11895,<Jean-Jacq...| 9081581.748842742|
|[11615,<Gottfried...| 7146037.710399863|
|[2025,<Friedrich_...| 6897244.1896990575|
|[1082,<William_Sh...| 4168778.144288711|
|[11034,<Adam_Smit...| 4100936.5022027283|
|[1121,<John_Locke...| 3868447.819527024|
|[1566,<Heraclitus...| 3616900.3025887734|
|[3746,<Karl_Marx>...| 3575419.671920321|
|[10954,<Søren_Kie...| 3143375.914849735|
|[7322,<David_Hume...| 3122089.3473657905|
|[8540,<Arthur_Sch...| 2978239.727690162|
|[3186,<Ibn_Tufail...| 2234249.031615453|
|[8267,<Epicurus>,24]| 1812594.4073720106|
+-----+-----+
only showing top 20 rows
```

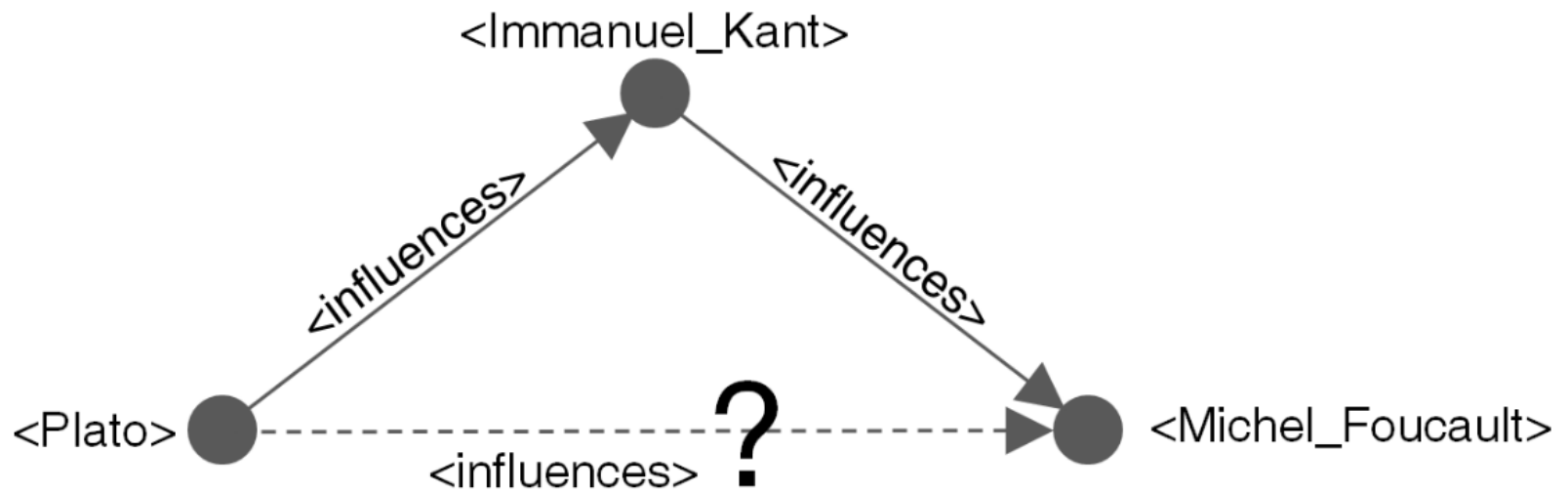
For Plato as A, score the C's



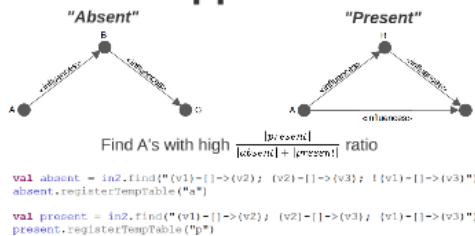
```
absent.sqlContext.sql(
  "SELECT v1.attr, " +
  "      v3.attr, " +
  "      SUM(v1.outdegree * v2.outdegree * v3.outdegree) AS score " +
  "FROM   a " +
  "      "WHERE v1.id=7662 " +
  "      "GROUP BY v1.attr, v3.attr " +
  "      "ORDER BY score DESC").collect

res24: Array[org.apache.spark.sql.Row] =
Array([<Plato>,<Karl_Marx>,7139388],
[<Plato>,<Jean-Paul_Sartre>,3143640], [<Plato>,<Michel_Foucault>,2871606],
[<Plato>,<Gilles_Deleuze>,2689128], [<Plato>,<Henri_Bergson>,2179128],
[<Plato>,<Maurice_Merleau-Ponty>,2088450]...
```

3-vertex <influences>



Approach



Outdegree in SQL

```
graph "c1r1ucrcu3" graph LR; A((A)) --> B((B)); B --> C((C));
val in = readRDD[(id, "graphSetInfluences.txt")]
in.registerTempTable("i")
in.vertices.registerTempTable("v")

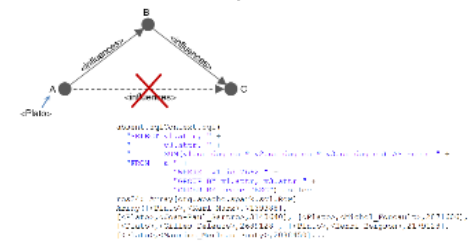
val in2 = GraphFrame(ir.vertices.sqlContext.sql(
  "SELECT v.id, "
  "FIRST(v.str) AS str, "
  "COUNT(*) AS outdegree "
  "FROM "
  "v "
  "JOIN i ON v.id=i.v1 "
  "GROUP BY v.id, str, outdegree
), ir.edges)
```

Score the A's

```
absent.registerTempTable("a")
absent.sqlContext.sql(
  "SELECT v.id, "
  "COUNT(*) AS outdegree, "
  "COUNT(*) AS in_degree "
  "FROM "
  "a "
  "JOIN i ON v.id=i.v1 "
  "GROUP BY v.id, outdegree, in_degree
), ir.edges)

absent.registerTempTable("pa")
absent.sqlContext.sql(
  "SELECT v.id, "
  "COUNT(*) AS outdegree, "
  "COUNT(*) AS in_degree "
  "FROM "
  "pa "
  "JOIN i ON v.id=i.v1 "
  "GROUP BY v.id, outdegree, in_degree
), ir.edges)
```

For Plato as A, score the C's



3-vertex general linear pattern


```

# Create an instance of the class
p1 = Person("John", 30, "Male")

# Call the say_hello method
p1.say_hello()

# Call the get_age method
p1.get_age()

# Call the get_gender method
p1.get_gender()

```

4-vertex general Y pattern

Reading YAGO

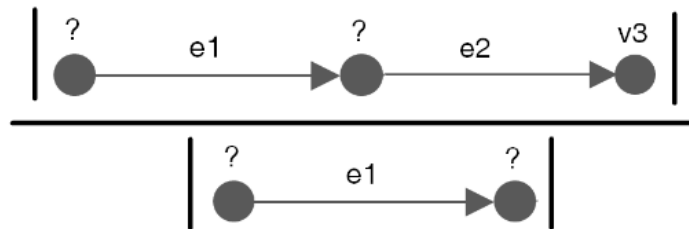
```
var yago = readRdfDf(sc, "/home/ec2-user/yago/yagoFacts.tsv")

def mergeIntoYago(filename:String):Unit = {
  val r = readRdfDf(sc, filename)
  r.vertices.cache
  r.edges.cache
  val g = mergeGraphs(yago, r)
  val g2 = GraphFrame(g.vertices, g.edges.distinct)
  g2.vertices.cache
  g2.vertices.count
  g2.edges.cache
  g2.edges.count
  yago.vertices.unpersist
  yago.edges.unpersist
  r.vertices.unpersist
  r.edges.unpersist
  yago = g2
}

mergeIntoYago("/home/ec2-user/yago/yagoLabels.tsv")
mergeIntoYago("/home/ec2-user/yago/yagoTaxonomy.tsv")
```

Query

```
yago.find("()-[e1]->(v2); (v2)-[e2]->(v3)")  
  .distinct  
  .selectExpr("e1.attr AS e1attr", "e2.attr AS e2attr",  
              "v3.id AS v3id", "v3.attr AS v3attr")  
  .groupBy("e1attr", "e2attr", "v3id", "v3attr")  
  .count  
  .join(yago.edges.groupBy("attr")  
        .count  
        .selectExpr("attr", "count AS e1count"),  
        $"e1attr" === $"attr")  
  .selectExpr("e1attr", "e2attr", "v3attr", "count/e1count AS ratio")  
  .orderBy($"ratio".desc)  
  .show(40)
```



Results



e1attr	e2attr	v3attr	ratio
<hasAcademicAdvisor>	<hasGender>	<male>	0.9746811345897582
<influences>	<hasGender>	<male>	0.8912262249101587
<hasGender>	skos:prefLabel	"male"@eng	0.8428451872279868
<hasGender>	rdfs:label	"male"@eng	0.8428451872279868
<isCitizenOf>	<participatedIn>	<War_on_Terror>	0.835011170256778
<dealsWith>	<participatedIn>	<War_on_Terror>	0.772226304188097
<isCitizenOf>	<participatedIn>	<Operation_Enduri...	0.7614529608075682
<isCitizenOf>	<participatedIn>	<War_in_Afghanist...	0.7572055051438343
<isCitizenOf>	<participatedIn>	<Operation_Enduri...	0.7488484982210332
<isCitizenOf>	<dealsWith>	<China>	0.742863447058499
<isCitizenOf>	<participatedIn>	<Korean_War>	0.7089389635104946
<isCitizenOf>	<participatedIn>	<Unified_Task_Force>	0.6842540750751579
<dealsWith>	<participatedIn>	<Operation_Enduri...	0.6796473181484203
<hasNeighbor>	<dealsWith>	<China>	0.6774193548387096

Discovered bug in YAGO 3.02 (not in YAGO 3.00 or forthcoming YAGO 3.03)

<isCitizenOf>	<participatedIn>	<Setting_Accounts>	0.620443400308033
<dealsWith>	<dealsWith>	<United_States>	0.6223365172667157
<dealsWith>	<dealsWith>	<China>	0.6193975018368847
<dealsWith>	<participatedIn>	<Operation_Enduri...	0.6186627479794269
<isCitizenOf>	<participatedIn>	<Operation_Desert...	0.6018424028463469
<isCitizenOf>	<participatedIn>	<Operation_Active...	0.587031469785145
<isCitizenOf>	<participatedIn>	<Operation_Eagle'...	0.586865984499545
<isCitizenOf>	<participatedIn>	<Operation_Enduri...	0.5862316242380782
<isCitizenOf>	<participatedIn>	<Iraqi_no-fly_zones>	0.5859282345478115
<isCitizenOf>	<participatedIn>	<Colombian_confli...	0.5689935736547426
<isCitizenOf>	<participatedIn>	<Operation_Provid...	0.5657390297046088
<isCitizenOf>	<participatedIn>	<European_theatre...	0.5540447361888738
<isCitizenOf>	<participatedIn>	<Western_Front_(w...	0.5523071406900737
<hasOfficialLangu...	rdfs:label	"Joseph"@eng	0.5515832482124617
<isCitizenOf>	<participatedIn>	<Helmand_province...	0.5459635380754061
<isCitizenOf>	<participatedIn>	<People's_General>	0.5350139283448714
<isCitizenOf>	<participatedIn>	<Congo_Crisis>	0.5346277960118047
<hasOfficialLangu...	rdfs:label	"Francis"@eng	0.5342185903983657

← What?

← What?

```

# Create an instance of the class
p1 = Person("John", 30, "Male")

# Call the say_hello method
p1.say_hello()

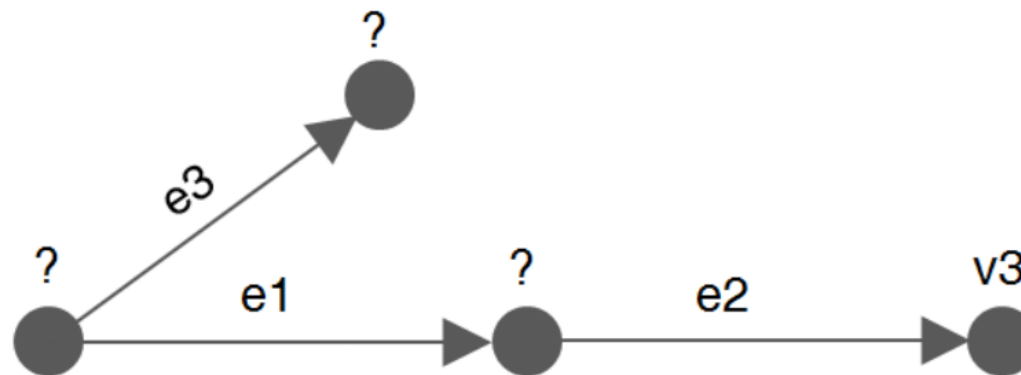
# Call the get_age method
p1.get_age()

# Call the get_gender method
p1.get_gender()

```

4-vertex general Y pattern

4-vertex general Y pattern



```
y1 = yago.find("(v1)-[e3]->(v4); (v1)-[e1]->(v2); (v2)-[e2]->(v3)")
    .distinct
    .selectExpr("e3.attr AS e3attr", "e1.attr AS e1attr", "e2.attr AS e2attr",
                "v3.id AS v3id", "v3.attr AS v3attr")
    .groupBy("e3attr", "e1attr", "e2attr", "v3id", "v3attr")
    .count
    .cache
```

Out of Memory

AWS 5 nodes of r3.8xlarge, each 244GB RAM, 32 vCPU, 320GB SSD

3-vertex <hasChild> <isMarriedTo>

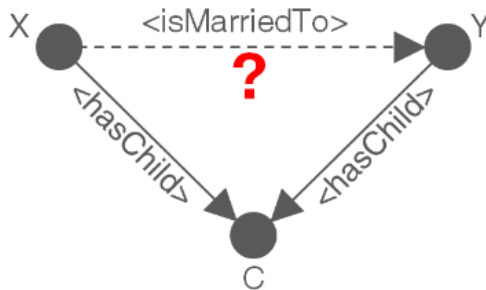
	parent1	parent2	child
0	0	0	0
1	0	1	1
2	1	0	1
3	1	1	0
4	0	0	0
5	0	1	1
6	1	0	1
7	1	1	0
8	0	0	0
9	0	1	1
10	1	0	1
11	1	1	0
12	0	0	0
13	0	1	1
14	1	0	1
15	1	1	0
16	0	0	0
17	0	1	1
18	1	0	1
19	1	1	0
20	0	0	0
21	0	1	1
22	1	0	1
23	1	1	0
24	0	0	0
25	0	1	1
26	1	0	1
27	1	1	0
28	0	0	0
29	0	1	1
30	1	0	1
31	1	1	0
32	0	0	0
33	0	1	1
34	1	0	1
35	1	1	0
36	0	0	0
37	0	1	1
38	1	0	1
39	1	1	0
40	0	0	0
41	0	1	1
42	1	0	1
43	1	1	0
44	0	0	0
45	0	1	1
46	1	0	1
47	1	1	0
48	0	0	0
49	0	1	1
50	1	0	1
51	1	1	0
52	0	0	0
53	0	1	1
54	1	0	1
55	1	1	0
56	0	0	0
57	0	1	1
58	1	0	1
59	1	1	0
60	0	0	0
61	0	1	1
62	1	0	1
63	1	1	0
64	0	0	0
65	0	1	1
66	1	0	1
67	1	1	0
68	0	0	0
69	0	1	1
70	1	0	1
71	1	1	0
72	0	0	0
73	0	1	1
74	1	0	1
75	1	1	0
76	0	0	0
77	0	1	1
78	1	0	1
79	1	1	0
80	0	0	0
81	0	1	1
82	1	0	1
83	1	1	0
84	0	0	0
85	0	1	1
86	1	0	1
87	1	1	0
88	0	0	0
89	0	1	1
90	1	0	1
91	1	1	0
92	0	0	0
93	0	1	1
94	1	0	1
95	1	1	0
96	0	0	0
97	0	1	1
98	1	0	1
99	1	1	0

```
val y2 = filterEdges(yago, ($'attr' === '#hasChild' || '$'attr' === '#isMarriedTo' || '$'attr' === '#hasGender'))

val df = y2.find((parent1)-[>](child); (parent2)-[>](child); (parent1)-[>](p1gender); (parent2)-[>](p2gender'))
    .filter('$p1gender.attr' === 'male' && '$p2gender.attr' === 'female') &&
    ('$e1.attr' === '#hasChild') && ('$e2.attr' === '#hasChild')
    .selectExpr('parent1.attr AS parent1', 'parent2.attr AS parent2', 'child.attr AS child')
```

主要成分	主要成分
1. 维生素A	2. 维生素B ₁
3. 维生素C	4. 维生素D
5. 维生素E	6. 维生素K
7. 维生素P	8. 维生素U
9. 维生素F	10. 维生素G
11. 维生素H	12. 维生素I
13. 维生素J	14. 维生素L
15. 维生素M	16. 维生素N
17. 维生素O	18. 维生素P
19. 维生素Q	20. 维生素R
21. 维生素S	22. 维生素T
23. 维生素U	24. 维生素V
25. 维生素W	26. 维生素X
27. 维生素Y	28. 维生素Z
29. 维生素AA	30. 维生素BB
31. 维生素CC	32. 维生素DD
33. 维生素EE	34. 维生素FF
35. 维生素GG	36. 维生素HH
37. 维生素II	38. 维生素JJ
39. 维生素KK	40. 维生素LL
41. 维生素MM	42. 维生素NN
43. 维生素OO	44. 维生素PP
45. 维生素QQ	46. 维生素RR
47. 维生素SS	48. 维生素TT
49. 维生素UU	50. 维生素VV
51. 维生素WW	52. 维生素XX
53. 维生素YY	54. 维生素ZZ
55. 维生素AAA	56. 维生素BBB
57. 维生素CCC	58. 维生素DDD
59. 维生素EEE	60. 维生素FFF
61. 维生素GGG	62. 维生素HHH
63. 维生素III	64. 维生素JJJ
65. 维生素KKK	66. 维生素LLL
67. 维生素MMM	68. 维生素NNN
69. 维生素OOO	70. 维生素PPP
71. 维生素QQQ	72. 维生素RRR
73. 维生素SSS	74. 维生素TTT
75. 维生素UUU	76. 维生素VVV
77. 维生素WWW	78. 维生素XXX
79. 维生素YYY	80. 维生素ZZZ
81. 维生素AAA	82. 维生素BBB
83. 维生素CCC	84. 维生素DDD
85. 维生素EEE	86. 维生素FFF
87. 维生素GGG	88. 维生素HHH
89. 维生素III	90. 维生素JJJ
91. 维生素KKK	92. 维生素LLL
93. 维生素MMM	94. 维生素NNN
95. 维生素OOO	96. 维生素PPP
97. 维生素QQQ	98. 维生素RRR
99. 维生素SSS	100. 维生素TTT

Initial Results



```

var yago = readRdfDf(sc, "/yago3/yagoFacts.tsv")

def removeSingletons(g:GraphFrame) =
  GraphFrame(g.vertices.sqlContext.createDataFrame(
    g.triplets.select("src").map(_.getStruct(0))
      .union(g.triplets.select("dst").map(_.getStruct(0)))
      .distinct,
    g.vertices.schema),
    g.edges)

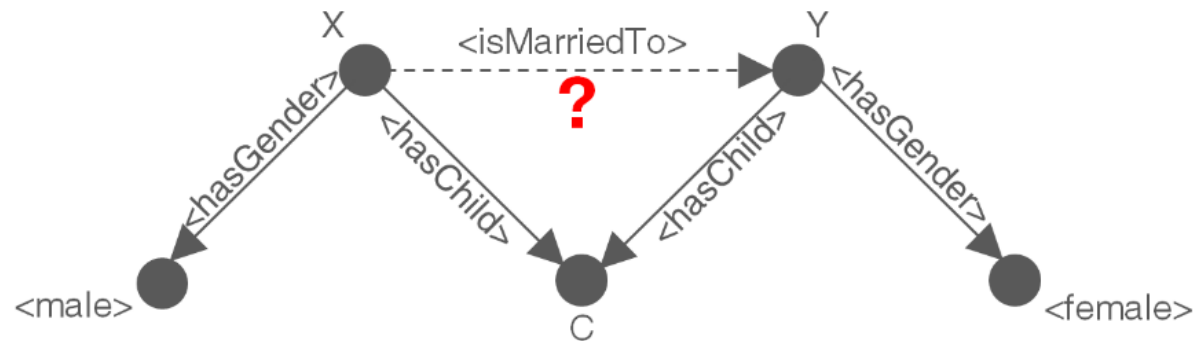
def filterEdges(g:GraphFrame, condition:Column) =
  removeSingletons(GraphFrame(g.vertices, g.edges.where(condition)))

val y2 = filterEdges(yago, $"attr" === "<hasChild>" || $"attr" === "<isMarriedTo>")

val df = y2.find("(parent1)-[e1]->(child); (parent2)-[e2]->(child); !(parent1)-[]->(parent2)")
  .filter(($"parent1" !== $"parent2") && ($"e1.attr" === "<hasChild>") &&
    ($"e2.attr" === "<hasChild>"))
  .selectExpr("parent1.attr AS parent1", "parent2.attr AS parent2",
    "child.attr AS child")
  
```

parent1	parent2	child
<Arcadius>	<Aelia Flaccilla>	<Honorius_(emperor)>
<Victor_Emanuel_...	<Victor_Emanuel_...	<Maria_Pia_of_Savoy>
<George_VI_of_the...	<George_VI>	<Elizabeth_II>
<Otto_I,_Duke_of_...	<Otto_I_Wittelsba...	<Louis_I,_Duke_of_...
<Jack_Nicholas_Pr...	<Rhoda_Pritzker>	<Nicholas_J._Prit...
<Augusta_Leigh>	<Lord_Byron>	<Elizabeth_Medora...
<Mehmed_the_Conqu...	<Mehmed_II>	<Bayezid_II>
<Jane_Stanhope,_C...	<Charles_Stanhope...	<Charles_Stanhope...
<Eleanor_Beaucham...	<Edmund_Beaufort,...>	<Edmund_Beaufort,...>
<Empress_Song_Fujin>	<Li_Bian>	<Li_Jing_(Souther...
<Guy_XIV_de_Laval>	<Isabella_of_Brit...	<Jeanne_de_Laval>
<Ronald_Reagan>	<Dorothy_Walker_B...	<George_H._W._Bush>
<Benedict_Swingat...	<Elizabeth_Calvert>	<Eleanor_Calvert>
<Hoapili>	<Kekuanaoa>	<Kamehameha_V>
<Leopold_II_of_Be...	<Prince_Philippe,...>	<Albert_I_of_Belg...
<Henry_the_Fowler>	<Henry_I_the_Fowler>	<Gerberga_of_Saxony>
<Ly_Hu?_Tông>	<Tr?n_Th?_Dung>	<Ly_Chieu_Hoàng>
<Argia_(mythology)>	<Aristodemus>	<Procles>
<William_I_of_Eng...	<William_the_Conq...	<Robert_Curthose>
<Emperor_Mommu>	<Emperor_Monmu>	<Emperor_Shomu>

Refinement



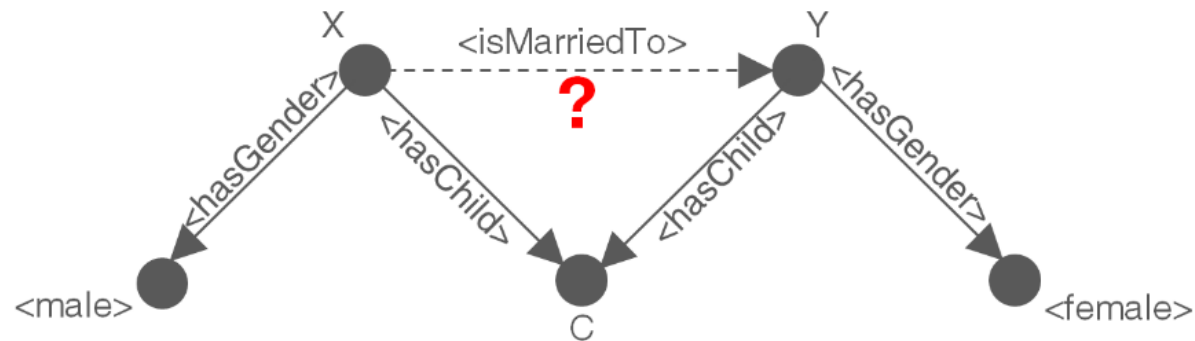
```
val y2 = filterEdges(yago, ("attr" === "<hasChild>" || "attr" === "<isMarriedTo>" || "attr" === "<hasGender>"))
```

```
val df = y2.find("(parent1)-[e1]->(child); (parent2)-[e2]->(child); (parent1)-[]->(p1gender); (parent2)-[]->(p2gender)")
    .filter(("p1gender.attr" === "<male>") && ("p2gender.attr" === "<female>") &&
        ("e1.attr" === "<hasChild>") && ("e2.attr" === "<hasChild>"))
    .selectExpr("parent1.attr AS parent1", "parent2.attr AS parent2", "child.attr AS child")
```

parent1	parent2	child
<Pinedjon_2>	<Duthachon-Henri...>	<Pausannes_2>
<Pinedjon_2>	<Duthachon-Henri...>	<Henkheerre>
<Joan_III_of_Fr...>	<Joan_III_of_Fr...>	<Joan_III_of_Fr...>
<Charles_III_of_N...>	<Eleanor_of_Casti...>	<Blanche_2_of_Nov...>
<John_de_Foix_1s...>	<Margaret_de_la_P...>	<Gaston_de_Foix>
<Prince_Tomohito...>	<Princess_Tomohit...>	<Princess_Akiko_o...>
<Prince_Tomohito...>	<Princess_Tomohit...>	<Princess_Yoko_of...>
<Dizy_Osbournes>	<Sharon_Osbournes>	<Jack_Osbournes>
<Orzy_Osbournes>	<Sharon_Osbournes>	<Kelly_Osbournes>
<Robert_Sidney_1...>	<Barbara_Sidney_...>	<Robert_Sidney_2...>
<Bruce_Sudano>	<Donna_Sunco>	<Brooklyn_Sudano>
<Phillip_V_of_France>	<Joan_II_Countess...>	<Blanche_of_Franc...>
<Phillip_V_of_France>	<Joan_II_Countess...>	<Isabella_of_Fran...>
<Edmund_I>	<Elififu_of_Shaft...>	<Edwige>
<Edmund_I>	<Elififu_of_Shaft...>	<Edgar_the_Peaceful>
<Ludovico_III_don...>	<Barbara_of_Brand...>	<Federico_I_donza...>
<Hotes_Hendemen...>	<Mary_Perkins>	<Bailey_Hardeman>
<Castro_Margrav...>	<Susanna_of_Bavaria>	<Albert_Archibald...>
<Edward_the_Elder>	<Elififu_of_Shaft...>	<Edwige>

parent1	parent2	child
<Pinedjem_I>	<Duathathor-Henut...>	<Psusennes_I>
<Pinedjem_I>	<Duathathor-Henut...>	<Menkheperre>
<Abdülmeceid_I>	<Gülüstü_Kadin_Ef...>	<Mehmed_VI>
<Charles_III_of_N...>	<Eleanor_of_Casti...>	<Blanche_I_of_Nav...>
<John_de_Foix,_1s...>	<Margaret_de_la_P...>	<Gaston_de_Foix,...>
<Prince_Tomohito_...>	<Princess_Tomohit...>	<Princess_Akiko_o...>
<Prince_Tomohito_...>	<Princess_Tomohit...>	<Princess_Yoko_of...>
<Ozzy_Osbourne>	<Sharon_Osbourne>	<Jack_Osbourne>
<Ozzy_Osbourne>	<Sharon_Osbourne>	<Kelly_Osbourne>
<Robert_Sidney,_1...>	<Barbara_Sidney,...>	<Robert_Sidney,_2...>
<Bruce_Sudano>	<Donna_Summer>	<Brooklyn_Sudano>
<Philip_V_of_France>	<Joan_II,_Countes...>	<Blanche_of_Franc...>
<Philip_V_of_France>	<Joan_II,_Countes...>	<Isabella_of_Fran...>
<Philip_V_of_France>	<Joan_II,_Countes...>	<Margaret_I,_Coun...>
<Edmund_I>	<Ælfgifu_of_Shaft...>	<Eadwig>
<Edmund_I>	<Ælfgifu_of_Shaft...>	<Edgar_the_Peaceful>
<Ludovico_III_Gon...>	<Barbara_of_Brand...>	<Federico_I_Gonza...>
<Thomas_Hardeman,...>	<Mary_Perkins>	<Bailey_Hardeman>
<Casimir,_Margrav...>	<Susanna_of_Bavaria>	<Albert_Alcibiade...>
<Edward_the_Elder>	<Ælfflæd,_wife_of...>	<Eadgyth>

Refinement



```
val y2 = filterEdges(yago, ("attr" === "<hasChild>" || "attr" === "<isMarriedTo>" || "attr" === "<hasGender>"))
```

```
val df = y2.find("(parent1)-[e1]->(child); (parent2)-[e2]->(child); (parent1)-[]->(p1gender); (parent2)-[]->(p2gender)")
    .filter(("p1gender.attr" === "<male>") && ("p2gender.attr" === "<female>") &&
        ("e1.attr" === "<hasChild>") && ("e2.attr" === "<hasChild>"))
    .selectExpr("parent1.attr AS parent1", "parent2.attr AS parent2", "child.attr AS child")
```

parent1	parent2	child
<Pinedon_2>	<Duathashor-Hamut...>	<Pausarnes_2>
<Pinedon_2>	<Duathashor-Hamut...>	<Penkheer>
<Ondineid_2>	<Ondineid_2>	<Ondineid_2>
<Charles_III_of_N...>	<Eleanor_of_Casti...>	<Blanche_2_of_Nov...>
<John_de_Foix_1s...>	<Margaret_de_la_P...>	<Gaston_de_Foix...>
<Prince_Tomohito...>	<Princess_Tomohit...>	<Princess_Akiko_O...>
<Prince_Tomohito...>	<Princess_Tomohit...>	<Princess_Yoko_of...>
<Ozzy_Osbourne>	<Sharon_Osbourne>	<Jack_Osbourne>
<Ozzy_Osbourne>	<Sharon_Osbourne>	<Kelly_Osbourne>
<Robert_Sidney_1...>	<Barbara_Sidney_1...>	<Robert_Sidney_2...>
<Bruce_Sudano>	<Donna_Sunco>	<Brooklyn_Sudano>
<Phillip_V_of_France>	<Joan_II_Countess...>	<Blanche_of_Franc...>
<Phillip_V_of_France>	<Joan_II_Countess...>	<Isabella_of_Fran...>
<Edmund_I>	<Elififu_of_Shaft...>	<Edwige>
<Edmund_I>	<Elififu_of_Shaft...>	<Edgar_the_Peaceful>
<Ludovico_III_don...>	<Barbara_of_Brand...>	<Federico_I_donza...>
<Hotes_Henderson...>	<Mary_Perkins>	<Bailey_Hardeman>
<Castro_Margrav...>	<Susanna_of_Pawart...>	<Albert_Archibald...>
<Edward_the_Elder>	<Elififu_of_Shaft...>	<Edwige>

[illegible]

Diagram illustrating a query graph for a join query. It shows three nodes: X (labeled <male>), Y (labeled <female>), and C. X has outgoing edges labeled <hasGender> to <male> and <hasChild> to C. Y has outgoing edges labeled <hasGender> to <female> and <hasChild> to C. A dashed arrow labeled <isMarriedTo> points from X to Y, with a red question mark below it.

```
val y2 = filterEdges(yago, ($'attr' === '#hasChild' || '$'attr' === '#isMarriedTo' || '$'attr' === '#hasGender'))
val df = y2.find((parent1)-[>](child); (parent2)-[>](child); (parent1)-[>](p1gender); (parent2)-[>](p2gender'))
    .filter('$p1gender.attr' === 'male' && '$p2gender.attr' === 'female') &&
    ('$e1.attr' === '#hasChild') && ('$e2.attr' === '#hasChild')
selectExpr('parent1.attr AS parent1', 'parent2.attr AS parent2', 'child.attr AS child')
```

主要成分	主要成分	主要成分
维生素A	维生素B ₁	维生素C
维生素D	维生素E	维生素K
维生素P	维生素B ₂	维生素B ₆
维生素B ₁₂	维生素B ₅	维生素B ₇
维生素B ₉	维生素B ₃	维生素B ₄
维生素B ₈	维生素B ₁₀	维生素B ₁₁
维生素B ₁₃	维生素B ₁₄	维生素B ₁₅
维生素B ₁₆	维生素B ₁₇	维生素B ₁₈
维生素B ₁₉	维生素B ₂₀	维生素B ₂₁
维生素B ₂₂	维生素B ₂₃	维生素B ₂₄
维生素B ₂₅	维生素B ₂₆	维生素B ₂₇
维生素B ₂₈	维生素B ₂₉	维生素B ₃₀
维生素B ₃₁	维生素B ₃₂	维生素B ₃₃
维生素B ₃₄	维生素B ₃₅	维生素B ₃₆
维生素B ₃₇	维生素B ₃₈	维生素B ₃₉
维生素B ₄₀	维生素B ₄₁	维生素B ₄₂
维生素B ₄₃	维生素B ₄₄	维生素B ₄₅
维生素B ₄₆	维生素B ₄₇	维生素B ₄₈
维生素B ₄₉	维生素B ₅₀	维生素B ₅₁
维生素B ₅₂	维生素B ₅₃	维生素B ₅₄
维生素B ₅₅	维生素B ₅₆	维生素B ₅₇
维生素B ₅₈	维生素B ₅₉	维生素B ₆₀
维生素B ₆₁	维生素B ₆₂	维生素B ₆₃
维生素B ₆₄	维生素B ₆₅	维生素B ₆₆
维生素B ₆₇	维生素B ₆₈	维生素B ₆₉
维生素B ₇₀	维生素B ₇₁	维生素B ₇₂
维生素B ₇₃	维生素B ₇₄	维生素B ₇₅
维生素B ₇₆	维生素B ₇₇	维生素B ₇₈
维生素B ₇₉	维生素B ₈₀	维生素B ₈₁
维生素B ₈₂	维生素B ₈₃	维生素B ₈₄
维生素B ₈₅	维生素B ₈₆	维生素B ₈₇
维生素B ₈₈	维生素B ₈₉	维生素B ₉₀
维生素B ₉₁	维生素B ₉₂	维生素B ₉₃
维生素B ₉₄	维生素B ₉₅	维生素B ₉₆
维生素B ₉₇	维生素B ₉₈	维生素B ₉₉
维生素B ₁₀₀	维生素B ₁₀₁	维生素B ₁₀₂
维生素B ₁₀₃	维生素B ₁₀₄	维生素B ₁₀₅
维生素B ₁₀₆	维生素B ₁₀₇	维生素B ₁₀₈
维生素B ₁₀₉	维生素B ₁₁₀	维生素B ₁₁₁
维生素B ₁₁₂	维生素B ₁₁₃	维生素B ₁₁₄
维生素B ₁₁₅	维生素B ₁₁₆	维生素B ₁₁₇
维生素B ₁₁₈	维生素B ₁₁₉	维生素B ₁₂₀
维生素B ₁₂₁	维生素B ₁₂₂	维生素B ₁₂₃
维生素B ₁₂₄	维生素B ₁₂₅	维生素B ₁₂₆
维生素B ₁₂₇	维生素B ₁₂₈	维生素B ₁₂₉
维生素B ₁₃₀	维生素B ₁₃₁	维生素B ₁₃₂
维生素B ₁₃₃	维生素B ₁₃₄	维生素B ₁₃₅
维生素B ₁₃₆	维生素B ₁₃₇	维生素B ₁₃₈
维生素B ₁₃₉	维生素B ₁₄₀	维生素B ₁₄₁
维生素B ₁₄₂	维生素B ₁₄₃	维生素B ₁₄₄
维生素B ₁₄₅	维生素B ₁₄₆	维生素B ₁₄₇
维生素B ₁₄₈	维生素B ₁₄₉	维生素B ₁₅₀
维生素B ₁₅₁	维生素B ₁₅₂	维生素B ₁₅₃
维生素B ₁₅₄	维生素B ₁₅₅	维生素B ₁₅₆
维生素B ₁₅₇	维生素B ₁₅₈	维生素B ₁₅₉
维生素B ₁₆₀	维生素B ₁₆₁	维生素B ₁₆₂
维生素B ₁₆₃	维生素B ₁₆₄	维生素B ₁₆₅
维生素B ₁₆₆	维生素B ₁₆₇	维生素B ₁₆₈
维生素B ₁₆₉	维生素B ₁₇₀	维生素B ₁₇₁
维生素B ₁₇₂	维生素B ₁₇₃	维生素B ₁₇₄
维生素B ₁₇₅	维生素B ₁₇₆	维生素B ₁₇₇
维生素B ₁₇₈	维生素B ₁₇₉	

The image shows a sequence of slides from a presentation. The slides are as follows:

- 3-vertex <influences>**: A directed graph with three nodes. The top node is labeled 'influences'. Below it are two nodes, one labeled 'is married to' and the other 'is a friend of'. Arrows point from the top node to both bottom nodes. A question mark is placed between the two bottom nodes.
- 3-vertex general linear pattern**: A directed graph with three nodes. The top node is labeled 'influences'. Below it are two nodes, one labeled 'is married to' and the other 'is a friend of'. Arrows point from the top node to both bottom nodes. A question mark is placed between the two bottom nodes.
- 4-vertex general Y pattern**: A directed graph with four nodes. The top node is labeled 'influences'. Below it are three nodes, one labeled 'is married to', one labeled 'is a friend of', and one labeled 'is a refugee'. Arrows point from the top node to all three bottom nodes.
- Out of Memory**: A slide with the text 'Out of Memory' in large red letters.
- 3-vertex <hasChild> <isMarriedTo>**: A directed graph with three nodes. The top node is labeled 'hasChild'. Below it are two nodes, one labeled 'is married to' and the other 'is a friend of'. Arrows point from the top node to both bottom nodes.
- Initial Friends**: A slide with the text 'Initial Friends' in large red letters.
- Refugees**: A slide with the text 'Refugees' in large red letters.

Graph Processing vs. Graph Database

OLAP-like

OLTP-like

**Parallel
algorithms**

**Parallel
query**

**Single-anchor
query**

Transactions

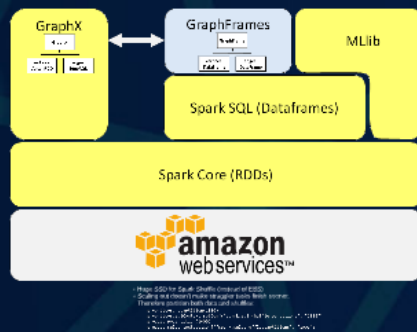


GraphFrames

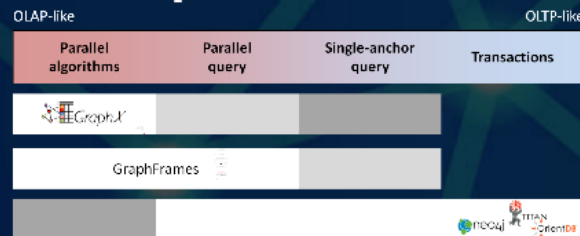


Spark

Stack



Graph Processing vs. Graph Database



History

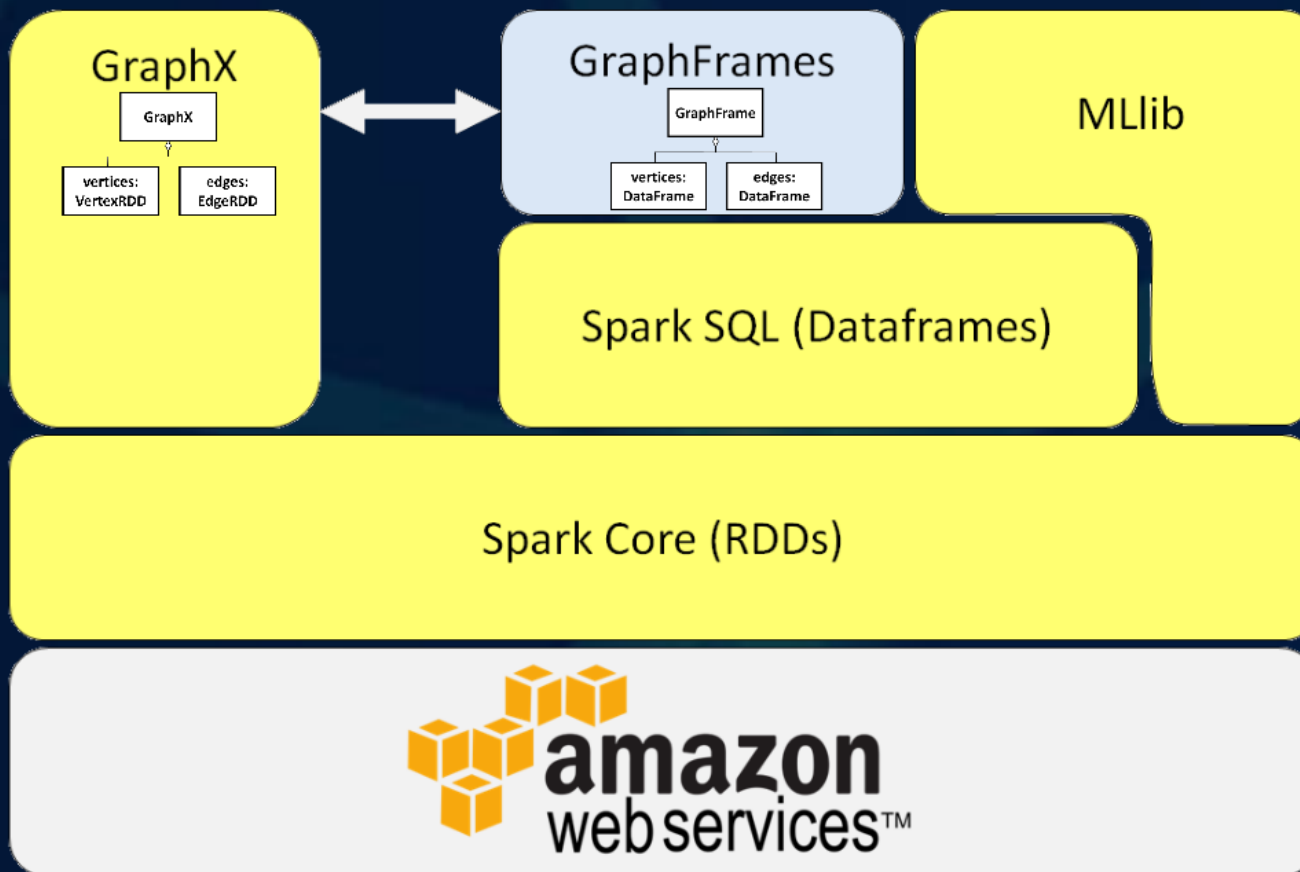
Timeline



Features & Performance

Bagel	GraphX	GraphFrames
- Features - Parallel - SQL-like language - SQL-like query - SQL-like execution - Performance - Flexibility	- Features - Parallel - SQL-like language - SQL-like query - SQL-like execution - Performance - Flexibility	- Features - Parallel - SQL-like language - SQL-like query - SQL-like execution - Performance - Flexibility

Stack



- Huge SSD for Spark Shuffle (instead of EBS)
 - Scaling out doesn't make straggler tasks finish sooner.
- Therefore partition both data and shuffles:

```
g.vertices.repartition(1000)
g.vertices.sqlContext.setConf("spark.sql.shuffle.partitions", "1000")
g.edges.repartition(1000)
g.edges.sqlContext.setConf("spark.sql.shuffle.partitions", "1000")
```

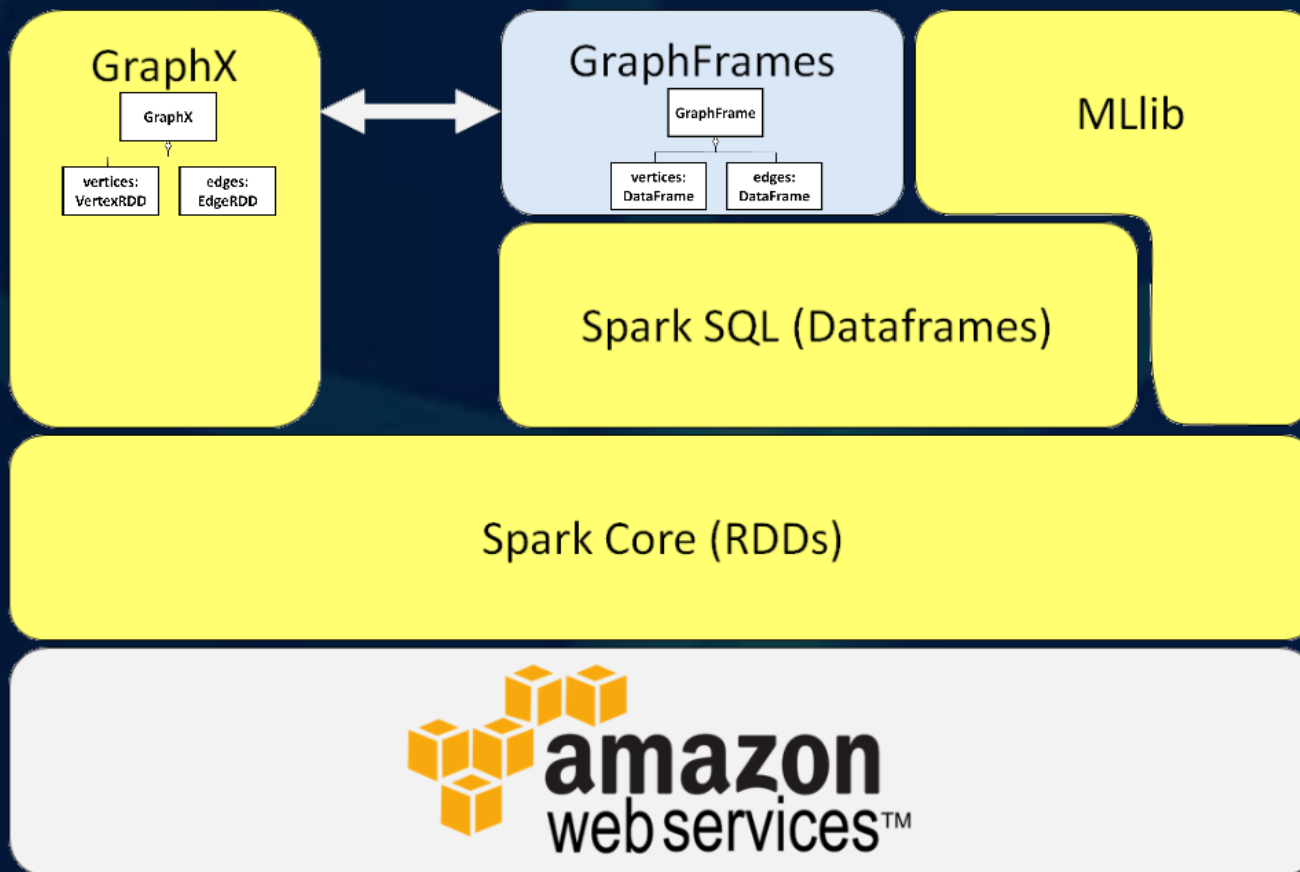


- Huge SSD for Spark Shuffle (instead of EBS)
- Scaling out doesn't make straggler tasks finish sooner.

Therefore partition both data and shuffles:

```
g.vertices.repartition(1000)
g.vertices.sqlContext.setConf("spark.sql.shuffle.partitions", "1000")
g.edges.repartition(1000)
g.edges.sqlContext.setConf("spark.sql.shuffle.partitions", "1000")
```

Stack



- Huge SSD for Spark Shuffle (instead of EBS)
 - Scaling out doesn't make straggler tasks finish sooner.
- Therefore partition both data and shuffles:

```
g.vertices.repartition(1000)
g.vertices.sqlContext.setConf("spark.sql.shuffle.partitions", "1000")
g.edges.repartition(1000)
g.edges.sqlContext.setConf("spark.sql.shuffle.partitions", "1000")
```


Finding Graph Isomorphisms in GraphX and GraphFrames

Michael Malak

Source code:
<https://github.com/michaelmalak/spark-graphx-is-action>
<https://github.com/michaelmalak>

ORACLE



SPARK SUMMIT 2016
MAY 10-11, 2016 SAN FRANCISCO, CA



History

Timeline



Isomorphisms

Definition



Detection



Finding



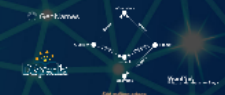
Summary

- Many frameworks (e.g., Spark, GraphX) support graph processing
- Some frameworks (e.g., Gremlin, Cypher) support graph database
- Graph processing is faster than graph database
- Graph database is more scalable than graph processing
- Graph processing is more flexible than graph database
- Graph database is more secure than graph processing
- Graph processing is more reliable than graph database



Yet Another Great Ontology (YAGO)

How It's Distributed



Summary

- Mini-isomorphisms (aka rule mining) good for:
 - Data quality/data cleansing
 - Recommendations
 - Opportunities
 - Fraud alerts
 - Product recommendations
- SVD++ in GraphX useful for tiniest of rules
- Cypher subset in GraphFrames for larger patterns
 - For larger patterns, better to pre-filter

Finding Graph Isomorphisms in GraphX and GraphFrames

Michael Malak

Source code:
<https://github.com/michaelmalak/spark-graphx-is-action>
<https://github.com/michaelmalak>

ORACLE



SPARK SUMMIT 2016
MAY 10-11, 2016 SAN FRANCISCO



History

Timeline



Isomorphisms

Definition



Detection



Finding



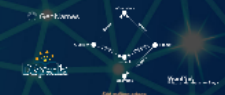
Summary

- Many frameworks (e.g., Neo4j, GraphX) support graph processing
- Some frameworks (e.g., Neo4j, GraphX) support graph storage
- Some frameworks (e.g., Neo4j, GraphX) support graph visualization
- Some frameworks (e.g., Neo4j, GraphX) support graph analysis
- Some frameworks (e.g., Neo4j, GraphX) support graph integration
- Some frameworks (e.g., Neo4j, GraphX) support graph security
- Some frameworks (e.g., Neo4j, GraphX) support graph management



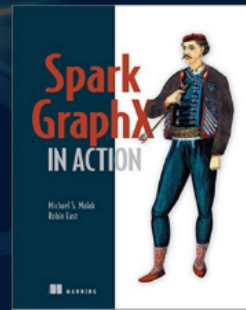
Yet Another Great Ontology (YAGO)

How It's Distributed



Finding Graph Isomorphisms in GraphX and GraphFrames

Michael Malak



Source code:

<https://www.manning.com/books/spark-graphx-in-action>

<https://gist.github.com/michaelmalak>

ORACLE®

Powered by
Spark



**Spark
summit**

SPARK SUMMIT 2016
DATA SCIENCE AND ENGINEERING AT SCALE
JUNE 6-8, 2016 SAN FRANCISCO