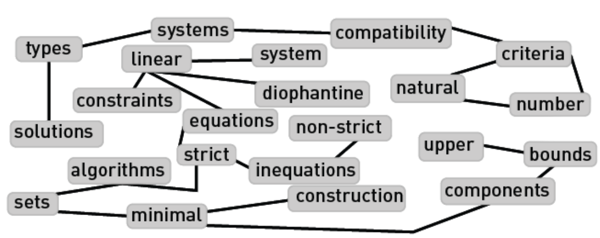
TextRank: Keyword Extraction

- Identify important words in a text
- Previous work
    - Mostly supervised learning
    - Genetic algorithms (Turney 1999), naïve Bayes (Frank 1999), rule induction (Hulth 2003)
- Keywords useful for:
    - Within other applications—information retrieval, text summarization, word-sense disambiguation
    - Terminology extraction
    - Automatic indexing

"TextRank: Bringing Order Into Texts," by Rada Mihalcea and Paul Tarau (EMNLP Conference 2004)

TextRank: An Example

Compatibility of systems of linear constraints over the set of natural numbers Criteria of compatibility of a system of linear Diophantine equations, strict inequations, and nonstrict inequations are considered. Upper bounds for components of a minimal set of solutions and algorithms of construction of minimal generation sets of solutions for all types of systems are given. These criteria and the corresponding algorithms for constructing a minimal supporting set of solutions can be used in solving all the considered types of systems and systems of mixed types.



TextRank

numbers (1.46)
inequations (1.45)
linear (1.29)
diophantine (1.28)
upper (0.99)
bounds (0.99)
strict (0.77)

Frequency

systems (4)
types (4)
solutions (3)
minimal (3)
linear (2)
inequations (2)
algorithms (2)

Keywords by TextRank: *linear constrains, linear diophantine equations natural numbers, non-strict inequations, strict inequations, upper bounds*

~ 100% match

Keywords by human annotators: *linear constraints, linear diophantine equations, non-strict inequations, set of natural numbers, strict inequations, upper bounds*

Augmented TextRank in Spark

The following is a Spark implementation of TextRank by Mihalcea, et al. The graph used in the algorithm is enriched by replacing the original authors' *Porter stemmer* approach with lemmatization from WordNet.

This algorithm generates a *graph* from a text document, linking together related words, then runs PageRank on that graph to determine the high-ranked keyphrases. Those keyphrases summarize the text document, similar to how an human editor would summarize for an academic paper.

See https://github.com/ceteri/spark-exercises/tree/master/exsto and also the earlier Hadoop implementation which leveraged *semantic relations* by extending the graph using hypernyms from WordNet as well.

First, we need to create *base RDDs* from the Parquet files that we stored in DBFS during the ETL phase...

```
val edge = sqlContext.parquetFile("/mnt/paco/exsto/graph/graf_edge.parquet")
edge.registerTempTable("edge")

val node = sqlContext.parquetFile("/mnt/paco/exsto/graph/graf_node.parquet")
node.registerTempTable("node")
```

```
edge: org.apache.spark.sql.DataFrame = [id: string, node0: bigint, node1: bigint]
node: org.apache.spark.sql.DataFrame = [id: string, node_id: bigint, raw: string, root: string, pos: string, keep: int, num: int]
Command took 11.77s
```

Let's pick one message as an example -- at scale we would parallelize this to run for all the messages.

```
val msg_id = "CA+B-+fyrBU1yGZAYJM_u=gnBVtzB=sXoBHkhmS-6L1n8K5Hhbw"
```

```
msg_id: String = CA+B-+fyrBU1yGZAYJM_u=gnBVtzB=sXoBHkhmS-6L1n8K5Hhbw
Command took 0.14s
```

Our use of GraphX requires some imports...

Graph Analytics

We compose a graph from the `node` and `edge` RDDs and run PageRank on it...

```
val g: Graph[String, Int] = Graph(nodes, edges)
val r = g.pageRank(0.0001).vertices
```

```
g: org.apache.spark.graphx.Graph[String,Int] = org.apache.spark.graphx.impl.GraphImpl@67140f
r: org.apache.spark.graphx.VertexRDD[Double] = VertexRDDImpl[1120] at RDD at VertexRDD.scala:57
Command took 21.02s
```

Save the resulting ranks for each word of interest...

```
case class Rank(id: Int, rank: Double, word: String)

val rank = r.join(nodes).map {
  case (node_id, (rank, word)) => Rank(node_id.toInt, rank, word)
}

rank.toDF().registerTempTable("rank")
```

```
defined class Rank
rank: org.apache.spark.rdd.RDD[Rank] = MapPartitionsRDD[1126] at map at <console>:48
Command took 1.42s
```