

Моделирование БД

Вершины

В вершинах графа будет храниться выбранный фрагмент текста в нормализованном виде.

Анализируемый текст - "Война и мир", Л.Н. Толстой.

Текст делится на параграфы. Параграфы формируют главы, главы - части.

Производится моделирование для случаев разбиения на параграфы, главы и части

Выборка значений из текста

Символов в параграфах

$$S_{par} := [1015 \ 412 \ 456 \ 231 \ 353 \ 245 \ 176 \ 286 \ 171 \ 420 \ 75 \ 1420 \ 189 \ 429 \ 537]^T$$

Символов в главах

$$S_{chp} := [12550 \ 7519 \ 11371 \ 16585 \ 7438 \ 22404 \ 8428 \ 4589 \ 6643 \ 3787 \ 8037 \ 8391]^T$$

Символов в частях

$$S_{blk} := [267558 \ 214656 \ 121649 \ 182496 \ 176234 \ 113721 \ 162835]^T$$

Средние значения

$$s_{par_m} := \text{floor} \left(\text{mean} (S_{par}) \right) = 4.27 \cdot 10^2$$

$$s_{chp_m} := \text{floor} \left(\text{mean} (S_{chp}) \right) = 9.81 \cdot 10^3$$

$$s_{blk_m} := \text{floor} \left(\text{mean} (S_{blk}) \right) = 1.77 \cdot 10^5$$

$$s_{total} := \sum S_{blk} = 1.24 \cdot 10^6 \quad - \text{ всего символов}$$

$$n_{par} := \text{floor} \left(\frac{s_{total}}{s_{par_m}} \right) = 2.9 \cdot 10^3 \quad - \text{ всего параграфов}$$

$$n_{chp} := \text{floor} \left(\frac{s_{total}}{s_{chp_m}} \right) = 1.3 \cdot 10^2 \quad - \text{ всего глав}$$

$$n_{blk} := \text{length} (S_{blk}) = 7 \quad - \text{ всего частей}$$

Некоторые константы

$$v_{sym} := 4 \quad - \text{ вес символа (здесь и далее - байт)}$$

$$v_{float} := 4 \quad - \text{ вес float}$$

$$v_{node_neo4j} := 15 \quad - \text{ вес вершины в neo4j (из документации)}$$

$$v_{edge_neo4j} := 33 \quad - \text{ вес связи в neo4j}$$

$$v_{prop_neo4j} := 41 \quad - \text{ вес свойства в neo4j}$$

Объем вершин в памяти

Далее размеры зависят от длины фрагмента.

$$v_{node_one} (s_m) := s_m \cdot v_{sym} + v_{node_neo4j} + v_{prop_neo4j} \quad - \text{ вес одной вершины}$$

$$v_{node_total} (n, s_m) := v_{node_one} (s_m) \cdot n \quad - \text{ вес всех вершин}$$

Ребра

В ребрах хранится информация о связях между фрагментами. На данный момент планируется реализовать два алгоритма вычисления связей:

- Пересечение словарей
- Пересечение имен собственных

В этом случае в ребре будет храниться коэффициент от 0 до 1 и 5 слов, по которым обнаружено пересечение.

В случае наличия достаточного времени в программу будут добавлены и другие алгоритмы.

Некоторые предположения

$s_{word_m} := 6$ - среднее количество символов в значащем слове

Объем ребра в памяти

$$v_{edge_one} := v_{float} + s_{word_m} \cdot v_{sym} + v_{edge_neo4j} + v_{prop_neo4j} \cdot 2 = 1.43 \cdot 10^2$$

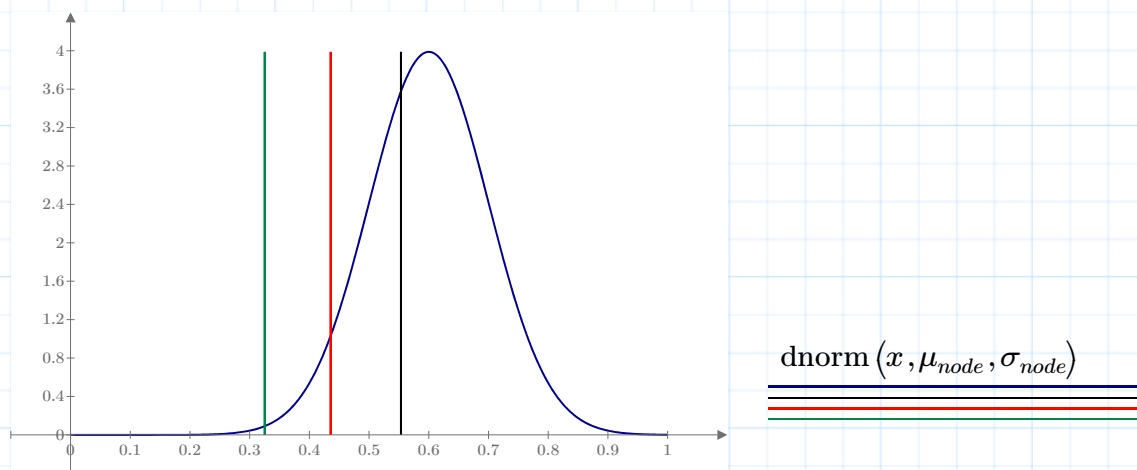
Моделирование

Характер связей

Предположим, что вероятность связи от одного фрагмента к другому подчинена закону нормального распределения.

Проверим затраты памяти без отсечения связей или с отсечением по 68% (условно 1σ), 95% (2σ) и 99.7% (3σ) от левого хвоста

$$\mu_{node} := 0.6 \quad \sigma_{node} := 0.1 \quad \sigma_0 := 0 \quad \sigma_1 := 1 - 0.68 \quad \sigma_2 := 1 - 0.95 \quad \sigma_3 := 1 - 0.997$$



x

$$qnorm(\sigma_1, \mu_{node}, \sigma_{node})$$

$$qnorm(\sigma_2, \mu_{node}, \sigma_{node})$$

$$qnorm(\sigma_3, \mu_{node}, \sigma_{node})$$

Таким образом, вес всех ребер:

$$v_{edge_total}(n, \sigma) := \text{floor} \left(n \cdot n \cdot (1 - \sigma) \cdot v_{edge_one} \right)$$

И общий вес:

$$v_{total}(s, n, \sigma) := v_{node_total}(n, s) + v_{edge_total}(n, \sigma)$$

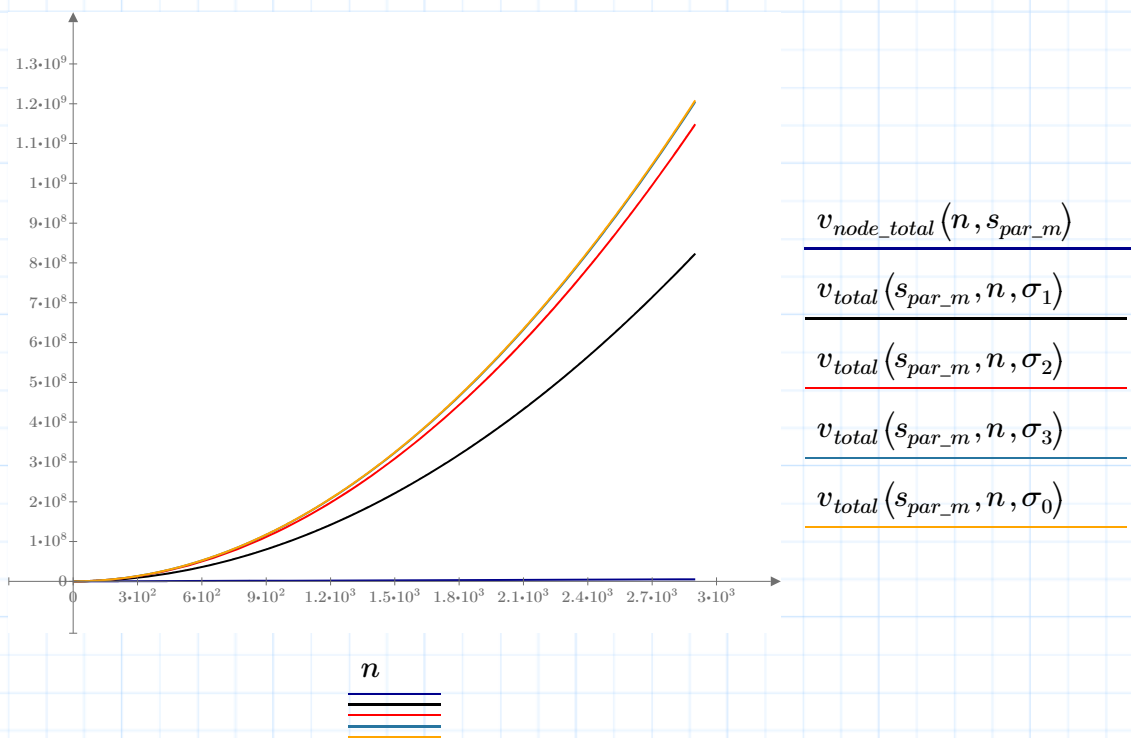
$$\Sigma := [\sigma_1 \ \sigma_2 \ \sigma_3]^T$$

$$P_{saving}(s, n) := \text{for } i \in 0 \dots \text{length}(\Sigma) - 1 \quad \text{return } r \quad \text{- ДОЛЯ ЭКОНОМИИ ОТ ОТКАЗА ОТ ЧАСТИ СВЯЗЕЙ}$$

$$\left\| r_i \leftarrow 1 - \frac{v_{total}(s, n, \Sigma_i)}{v_{total}(s, n, \sigma_0)} \right\|$$

$$p_{node}(s, n, \sigma) := \frac{v_{node_total}(n, s)}{v_{total}(s, n, \sigma)} \quad \text{- ДОЛЯ ВЕРШИН}$$

Моделирование для параграфов



$$v_{total}(s_{par_m}, n_{par}, \sigma_0) = 1.21 \cdot 10^9 \quad \text{- общий вес при разбиении на параграфы}$$

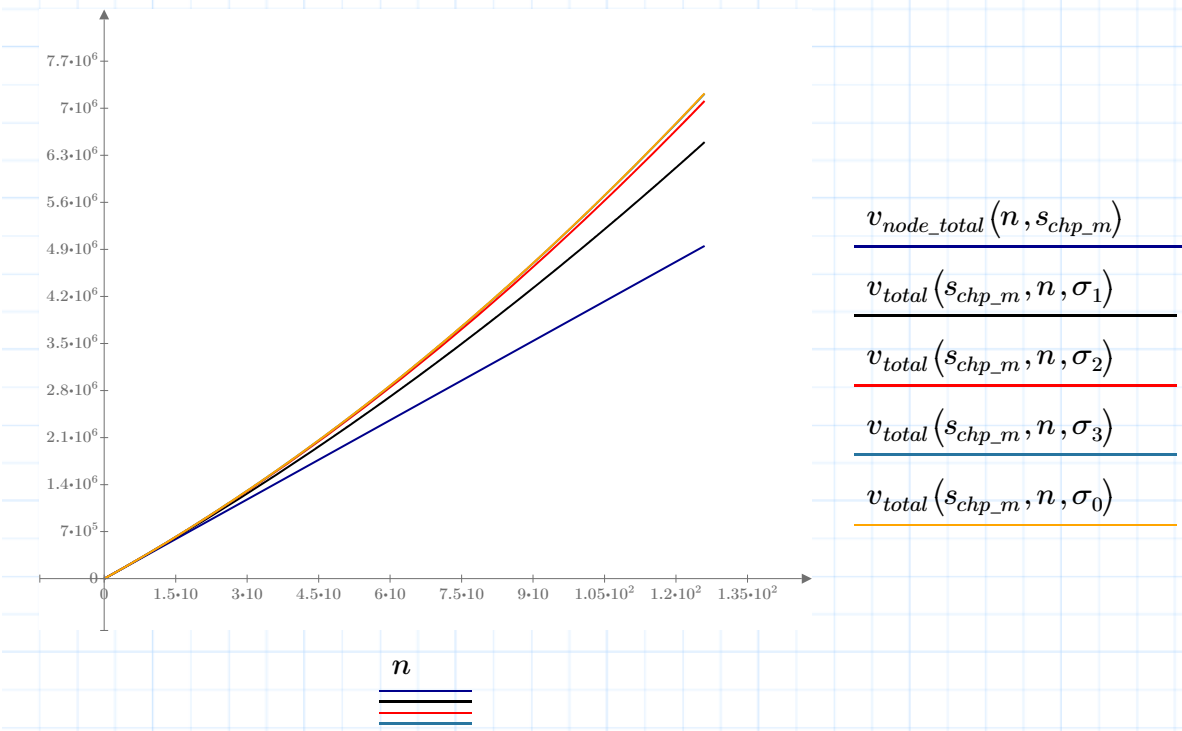
$$P_{saving}(s_{par_m}, n_{par}) = \begin{bmatrix} 31.86\% \\ 4.98\% \\ 0.3\% \end{bmatrix} \quad \text{- экономия места при отказе от части связей (32%, 5%, 0.3%)}$$

$$p_{node}(s_{par_m}, n_{par}, \sigma_0) = 0.42\% \quad \text{- доля вершин в общем объеме}$$

Как видно, в случае разбиения на параграфы ребра составляют подавляющую часть объема. В этом случае отсекание части связей приводит к почти пропорциональному

ЭКОНОМИЮ МЕСТА.

Моделирование для глав



$$v_{total}(s_{chp_m}, n_{chp}, \sigma_0) = 7.22 \cdot 10^6 \quad - \text{общий вес при разбиении на главы}$$

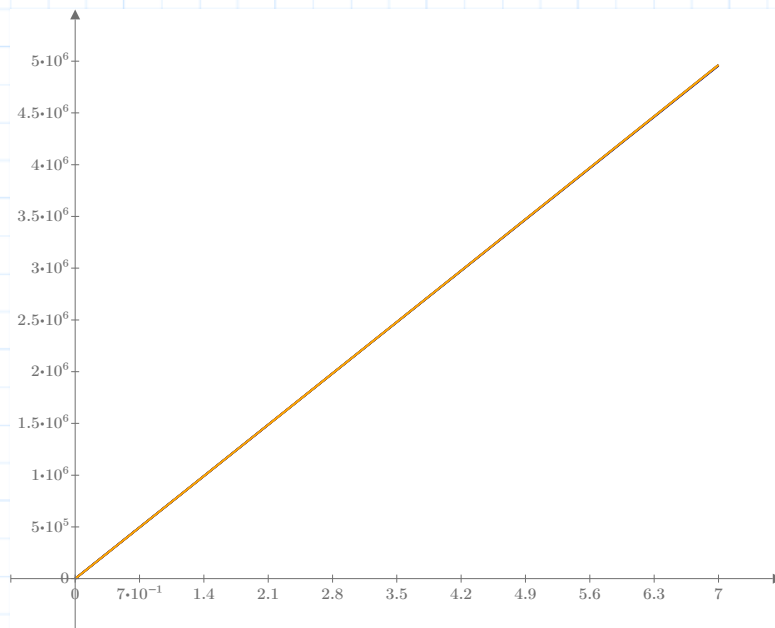
$$P_{saving}(s_{chp_m}, n_{chp}) = \begin{bmatrix} 10.06\% \\ 1.57\% \\ 0.09\% \end{bmatrix} \quad - \text{экономия места при отказе от части связей (32\%, 5\%, 0.3\%)}$$

$$p_{node}(s_{chp_m}, n_{chp}, \sigma_0) = 68.56\% \quad - \text{доля вершин в общем объеме}$$

$$\frac{v_{total}(s_{chp_m}, n_{chp}, \sigma_0)}{v_{total}(s_{par_m}, n_{par}, \sigma_0)} = 0.6\% \quad - \text{отношение объемов БД при разбиении на главы и на параграфы}$$

В случае разбиения на главы объем БД гораздо меньше. В этом случае текст составляет более двух третей объема, и экономия места при отсечении вершин в несколько раз ниже

Моделирование для частей



$$\underline{v_{node_total}(n, s_{blk_m})}$$

$$\underline{v_{total}(s_{blk_m}, n, \sigma_1)}$$

$$\underline{v_{total}(s_{blk_m}, n, \sigma_2)}$$

$$\underline{v_{total}(s_{blk_m}, n, \sigma_3)}$$

$$\underline{v_{total}(s_{blk_m}, n, \sigma_0)}$$

n

$$v_{total}(s_{blk_m}, n_{blk}, \sigma_0) = 4.96 \cdot 10^6 \quad \text{- общий вес при разбиении на части}$$

$$P_{saving}(s_{blk_m}, n_{blk}) = \begin{bmatrix} 0.05\% \\ 0.01\% \\ 0 \end{bmatrix} \quad \text{- экономия места при отказе от части связей (32%, 5%, 0.3%)}$$

$$p_{node}(s_{blk_m}, n_{blk}, \sigma_0) = 99.86\% \quad \text{- доля вершин в общем объеме}$$

$$\frac{v_{total}(s_{blk_m}, n_{blk}, \sigma_0)}{v_{total}(s_{chp_m}, n_{chp}, \sigma_0)} = 68.73\% \quad \text{- отношение объемов БД при разбиении на части и на главы}$$

Подавляющее большинство объема занимает текст. Отсечение вершин не имеет смысла

Сравнение с SQL

Для анализа выберем БД MySQL

```
CREATE DATABASE nosql_graph_text CHARACTER SET utf8 COLLATE  
utf8_unicode_ci;
```

Пусть фрагменты хранятся в следующей таблице:

```
CREATE TABLE texts (  
  id INT AUTO_INCREMENT PRIMARY_KEY,  
  text LONGTEXT  
)
```

Пусть связи хранятся в таблицах такого вида:

```
CREATE TABLE link_1 (  
  id INT AUTO_INCREMENT PRIMARY_KEY,  
  text_1_id INT,  
  text_2_id INT,  
  intersection FLOAT,  
  word_1 NVARCHAR(20),  
  word_2 NVARCHAR(20),  
  word_3 NVARCHAR(20),  
  word_4 NVARCHAR(20),  
  word_5 NVARCHAR(20),  
  
  CONSTRAINT text1_index_constraint  
  INDEX ind_text1(text_1_id),  
  
  CONSTRAINT text2_index_constraint,  
  INDEX ind_text2(text_2_id),  
  
  CONSTRAINT text1_key_constraint  
  FOREIGN_KEY fk_text1(text_1_id)  
  REFERENCES texts(id)  
  ON_UPDATE CASCADE  
  ON_DELETE CASCADE,  
  
  CONSTRAINT text2_key_constraint  
  FOREIGN_KEY fk_text2(text_2_id)  
  REFERENCES texts(id)  
  ON_UPDATE CASCADE  
  ON_DELETE CASCADE  
)
```

Вычислим объем тех же данных в MySQL.

Некоторые константы:

$$v_{nvarchar_sql}(n) := n \cdot 4 + 1$$
$$v_{int_sql} := 4$$
$$v_{longtext_sql}(n) := n \cdot 4 + 4$$
$$v_{float_sql} := 4$$

Индексы

При использовании движка InnoDB первичный ключ не занимает лишнего места.
Для использования внешних ключей нужны индексы.

Размер индекса обычно близок к результату следующей формулы:

$$v_{index_sql}(n, v_{field_pk}, v_{field}) := n \cdot (v_{field_pk} + v_{field}) \cdot 3$$

Размер таблиц в памяти

$$v_{texts_sql}(n, s) := n \cdot (v_{int_sql} + v_{longtext_sql}(s))$$

$$v_{links_index_sql}(n, \sigma) := v_{index_sql}(n \cdot n \cdot (1 - \sigma), v_{int_sql}, v_{int_sql})$$

$$v_{links_fields_sql}(n, \sigma) := n \cdot n \cdot (1 - \sigma) \cdot (v_{int_sql} \cdot 3 + v_{nvarchar_sql}(20) \cdot 5 + v_{float_sql})$$

$$v_{links_sql}(n, \sigma) := \text{floor}(v_{links_index_sql}(n, \sigma) + v_{links_fields_sql}(n, \sigma))$$

$$v_{total_sql}(s, n, \sigma) := v_{texts_sql}(n, s) + v_{links_sql}(n, \sigma)$$

Сравнение

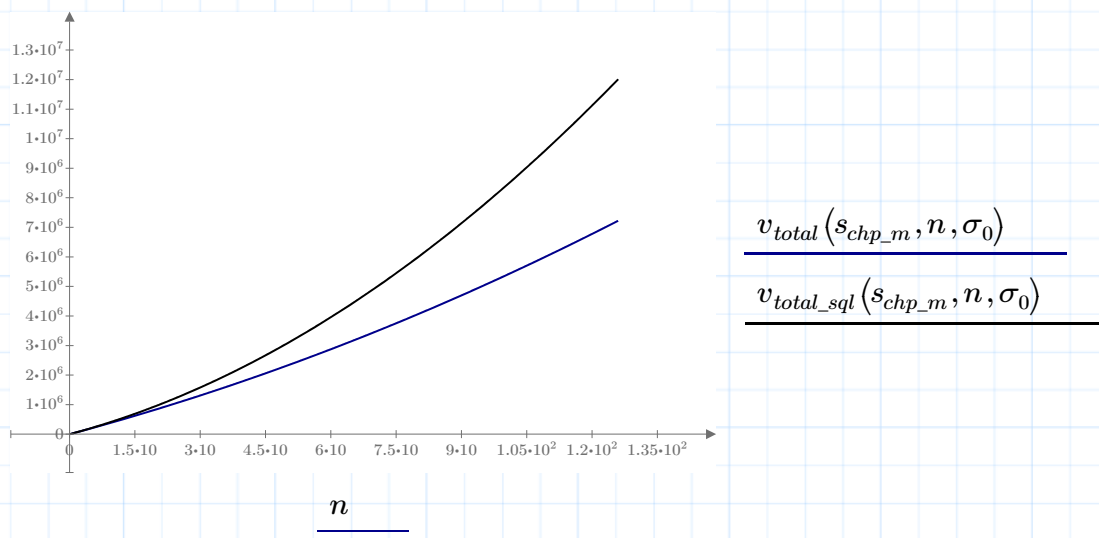
$$\frac{v_{total}(s_{blk_m}, n_{blk}, \sigma_0)}{v_{total_sql}(s_{blk_m}, n_{blk}, \sigma_0)} = 99.71\% \quad - \text{сравнение для разбиения по частям}$$

$$\frac{v_{total}(s_{chp_m}, n_{chp}, \sigma_0)}{v_{total_sql}(s_{chp_m}, n_{chp}, \sigma_0)} = 60.13\% \quad - \text{сравнение для разбиения по главам}$$

$$\frac{v_{total}(s_{par_m}, n_{par}, \sigma_0)}{v_{total_sql}(s_{par_m}, n_{par}, \sigma_0)} = 32.23\% \quad - \text{сравнение для разбиения по параграфам}$$

Получается, что при разбиении по частям объем примерно одинаков; при разбиении по главам и параграфам - несколько выигрывает NoSQL.

График зависимости



Моделирование запросов

Получить тексты с процентами связи > N

Cypher

```
MATCH (t1:Text)-[rel:ALG1]-(t2:Text)
WHERE rel.intersection > ${N}
RETURN id(t1), id(t2), rel.intersection
```

SQL

```
CREATE procedure get_intersect_more(IN N float)
BEGIN
    SELECT text1_id, text2_id, intersection FROM link_1
    WHERE intersection > N
END;
```

Добавление вершины

Cypher

```
CREATE (t:Text {text: ${TEXT}})
```

SQL

```
INSERT INTO texts
VALUES ${TEXT}
```

Кратчайший путь

Cypher

```
MATCH (start:Text{id: ${start_id}}), (end:Text{id: ${end_id}})
CALL algo.shortestPath.stream(start, end, 'cost' , {
    nodeQuery: 'MATCH (n:Text) RETURN id(n) AS id',
    relationshipQuery: 'MATCH (t1:Text)-[rel:${REL_TYPE}]- (t2:Text)
        WHERE rel.intersection > ${N}
        RETURN id(t1) AS source, id(t2) as target, 1 as weight'
})
YIELD nodeId, cost
RETURN nodeId, cost
```

SQL

На SQL нужно во многом реализовывать вне языка запросов. Например, для Алгоритма Дейкстры нужны следующие процедуры

```
CREATE procedure get_weight(IN id1 INT, IN id2 INT, IN max FLOAT)
BEGIN
```

```
    SELECT intersection FROM links
    WHERE links.text_1_id = id1 AND links.text_2_id = id2 AND links.intersection >=
max
END
```

```
CREATE procedure get_adjascent_nodes(IN id, IN max FLOAT)
BEGIN
```

```
    SELECT text_1_id AS text_id
    FROM links
    WHERE links.text_2_id = id
UNION
    SELECT text_2_id AS text_id
    FROM links
    WHERE links.text_1_id = id
```


END

Вышеприведённые запросы имеют шанс получить сложность $O(m)$ (правда, создание индексов поможет сгладить этот недостаток). В neo4j, в отличие от SQL-варианта, для поиска веса связи между вершинами не нужно итерироваться по всем связям графа.

Поэтому neo4j может быть эффективнее SQL в $O(m/n)$ раз. Рационально предположить, что это верно и для нижеприведённых алгоритмов

PageRank

Cypher

CALL algo.pageRank.stream('Text', 'ALG1', {iterations:20, dampingFactor:0.85})

YIELD nodeId, score

RETURN algo.getNodeById(nodeId).id **AS** page,score

ORDER BY score **DESC**

Компоненты связности

Cypher

CALL algo.unionFind.stream('Text', 'ALG1', {
 weightProperty:'intersection',

 defaultValue:0.0,

 threshold: \${N},

 concurrency: 1

})

YIELD nodeId,setId

RETURN algo.getNodeById(nodeId).id **AS** fragId, setId

ORDER BY setId, fragId