Моделирование БД

Вершины

В вершинах графа будет хранится 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

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

Символов в главах

$$S_{chp} := \begin{bmatrix} 12550 & 7519 & 11371 & 16585 & 7438 & 22404 & 8428 & 4589 & 6643 & 3787 & 8037 & 8391 \end{bmatrix}^{T}$$

Символов в частях

$$S_{blk} := \begin{bmatrix} 267558 & 214656 & 121649 & 182496 & 176234 & 113721 & 162835 \end{bmatrix}^{T}$$

Средние значения

$$s_{par\_m} := \mathrm{floor}\left(\mathrm{mean}\left(S_{par}\right)\right) = 4.27 \cdot 10^{2}$$
$$s_{chp\_m} := \mathrm{floor}\left(\mathrm{mean}\left(S_{chp}\right)\right) = 9.81 \cdot 10^{3}$$
$$s_{blk\_m} := \mathrm{floor}\left(\mathrm{mean}\left(S_{blk}\right)\right) = 1.77 \cdot 10^{5}$$

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

$n_{par} := \mathrm{floor}\left(\frac{s_{total}}{s_{par\_m}}\right) = 2.9 \cdot 10^{3}$ — *- всего параграфов*

$n_{chp} := \mathrm{floor}\left(\frac{s_{total}}{s_{chp\_m}}\right) = 1.3 \cdot 10^{2}$ — *- всего глав*

$n_{blk} := \mathrm{length}\left(S_{blk}\right) = 7$ — *- всего частей*

Некоторые константы

$v_{sym} := 4$ — *- вес символа (здесь и далее - байт)*
$v_{float} := 4$ — *- вес float*
$v_{node\_neo4j} := 15$ — *- вес вершины в neo4j (из документации)*
$v_{edge\_neo4j} := 33$ — *- вес связи в neo4j*
$v_{prop\_neo4j} := 41$ — *- вес свойства в neo4j*

Объем вершин в памяти

Далее размеры зависят о длины фрагмента.

$v_{node\_one}\left(s_m\right) := s_m \cdot v_{sym} + v_{node\_neo4j} + v_{prop\_neo4j}$ — *- вес одной вершины*
$v_{node\_total}\left(n, s_m\right) := v_{node\_one}\left(s_m\right) \cdot n$ — *- 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 хранится 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

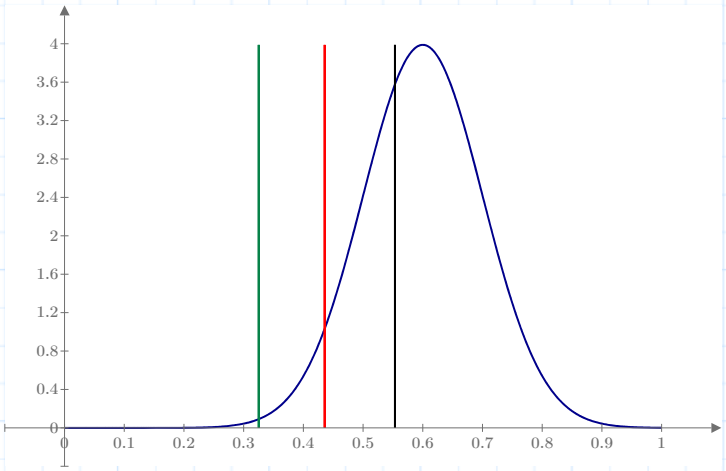
$v_{edge\_one} := v_{float} + s_{word\_m} \cdot v_{sym} + v_{edge\_neo4j} + v_{prop\_neo4j} \cdot 2 = 1.43 \cdot 10^2$

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

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



$\mathrm{dnorm}\left(x, \mu_{node}, \sigma_{node}\right)$

x

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

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

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

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:

$$v_{total}(s,n,\sigma) := v_{node\_total}(n,s) + v_{edge\_total}(n,\sigma)$$

$$\Sigma := \begin{bmatrix} \sigma_1 & \sigma_2 & \sigma_3 \end{bmatrix}^{\mathrm{T}}$$

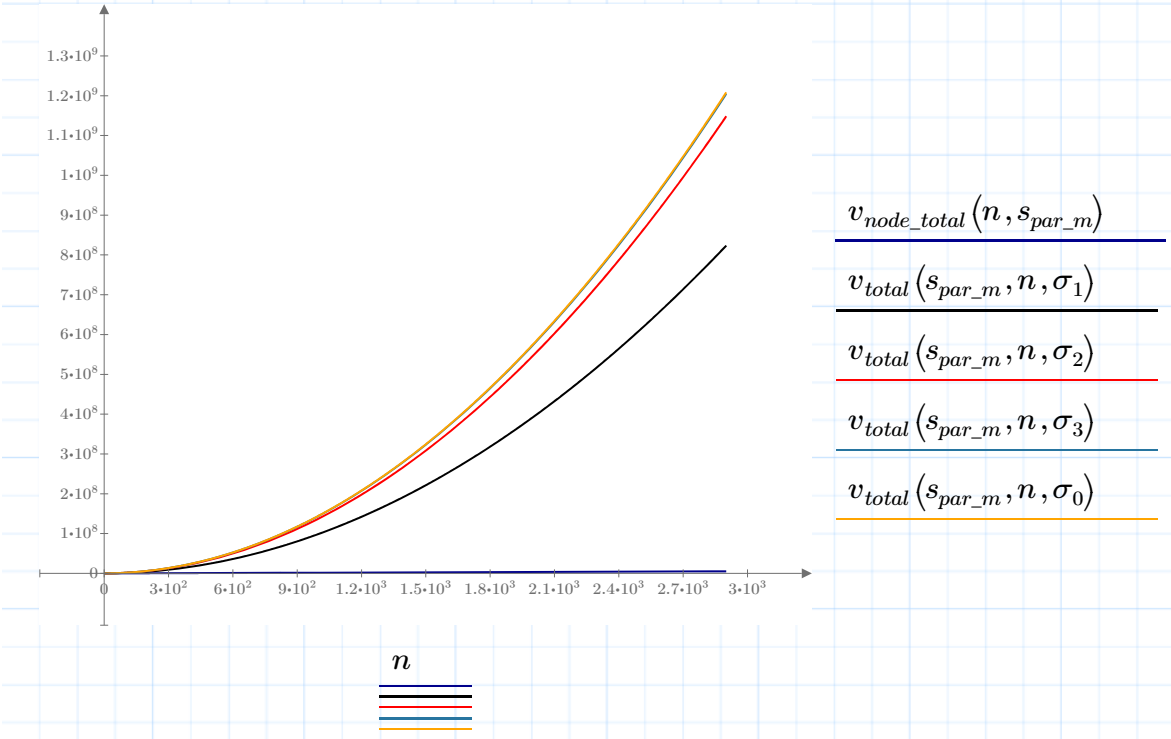
$$P_{saving}(s,n) := \left| \begin{array}{l} \text{for } i \in 0..\text{length}(\Sigma)-1 \\ \quad r_i \leftarrow 1 - \dfrac{v_{total}(s,n,\Sigma_i)}{v_{total}(s,n,\sigma_0)} \\ \text{return } r \end{array} \right.$$

- доля экономии от отказа от части связей

$$p_{node}(s,n,\sigma) := \frac{v_{node\_total}(n,s)}{v_{total}(s,n,\sigma)}$$

- доля вершин

Моделирование для параграфов



$$v_{total}(s_{par\_m}, n_{par}, \sigma_0) = 1.21 \cdot 10^9$$

- общий вес при разбиении на параграфы

$$P_{saving}(s_{par\_m}, n_{par}) = \begin{bmatrix} 31.86\% \\ 4.98\% \\ 0.3\% \end{bmatrix}$$

- экономия места при отказе от части связей (32%, 5%, 0.3%)

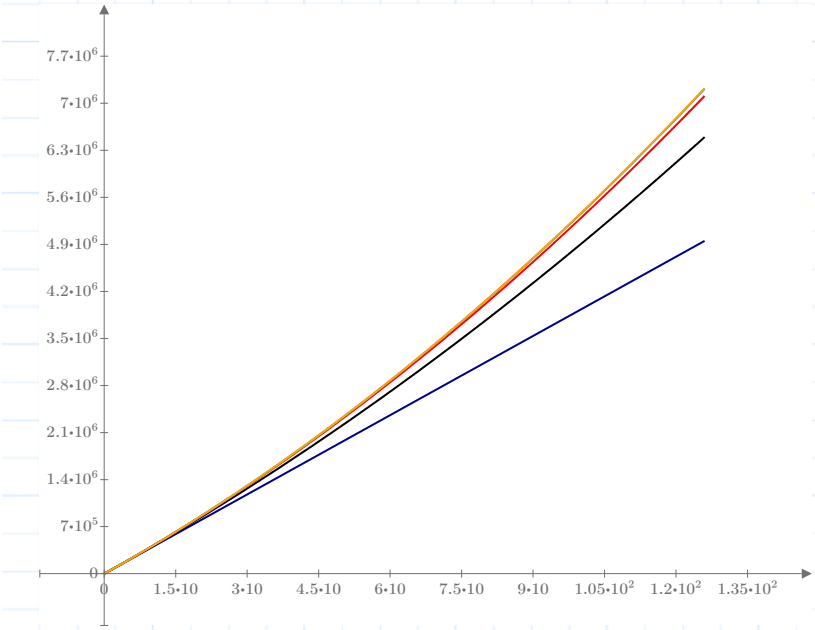
$$p_{node}(s_{par\_m}, n_{par}, \sigma_0) = 0.42\%$$

- 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



$v_{total}\left(s_{chp\_m}, n_{chp}, \sigma_0\right) = 7.22 \cdot 10^6$ ***- общий вес при разбиении на главы***

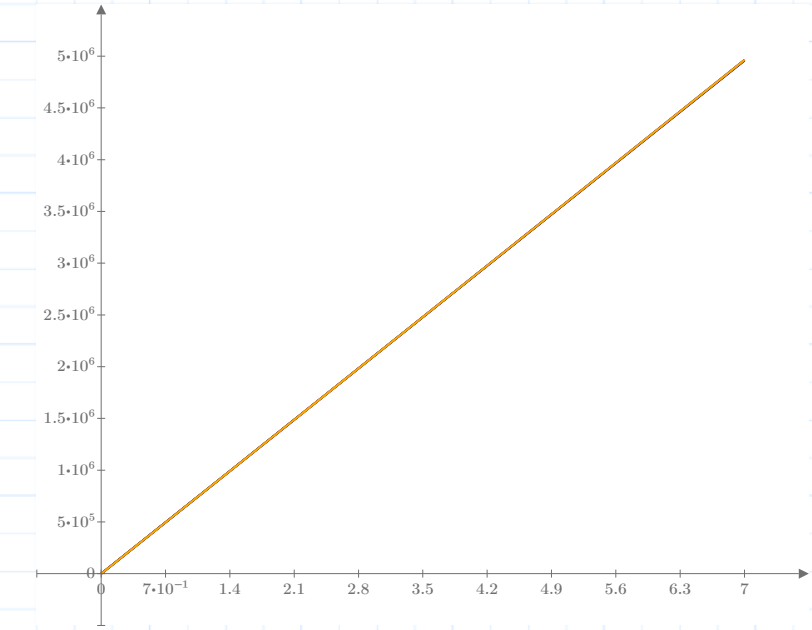
$P_{saving}\left(s_{chp\_m}, n_{chp}\right) = \begin{bmatrix} 10.06\% \\ 1.57\% \\ 0.09\% \end{bmatrix}$ ***- экономия места при отказе от части связей (32%, 5%, 0.3%)***

$p_{node}\left(s_{chp\_m}, n_{chp}, \sigma_0\right) = 68.56\%$ ***- доля вершин в общем объеме***

$\frac{v_{total}\left(s_{chp\_m}, n_{chp}, \sigma_0\right)}{v_{total}\left(s_{par\_m}, n_{par}, \sigma_0\right)} = 0.6\%$ ***- 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



$v_{node\_total}(n, s_{blk\_m})$

$v_{total}(s_{blk\_m}, n, \sigma_1)$

$v_{total}(s_{blk\_m}, n, \sigma_2)$

$v_{total}(s_{blk\_m}, n, \sigma_3)$

$v_{total}(s_{blk\_m}, n, \sigma_0)$

n

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

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

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

$\frac{v_{total}(s_{blk\_m}, n_{blk}, \sigma_0)}{v_{total}(s_{chp\_m}, n_{chp}, \sigma_0)} = 68.73\%$ ***- 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:

$$v_{index\_sql}\left(n, v_{field\_pk}, v_{field}\right) := n \cdot \left(v_{field\_pk} + v_{field}\right) \cdot 3$$

Размер таблиц в памяти

$$v_{texts\_sql}\left(n, s\right) := n \cdot \left(v_{int\_sql} + v_{longtext\_sql}\left(s\right)\right)$$

$$v_{links\_index\_sql}\left(n, \sigma\right) := v_{index\_sql}\left(n \cdot n \cdot \left(1-\sigma\right), v_{int\_sql}, v_{int\_sql}\right)$$
$$v_{links\_fields\_sql}\left(n, \sigma\right) := n \cdot n \cdot \left(1-\sigma\right) \cdot \left(v_{int\_sql} \cdot 3 + v_{nvarchar\_sql}\left(20\right) \cdot 5 + v_{float\_sql}\right)$$
$$v_{links\_sql}\left(n, \sigma\right) := \text{floor}\left(v_{links\_index\_sql}\left(n, \sigma\right) + v_{links\_fields\_sql}\left(n, \sigma\right)\right)$$

$$v_{total\_sql}\left(s, n, \sigma\right) := v_{texts\_sql}\left(n, s\right) + v_{links\_sql}\left(n, \sigma\right)$$

Сравнение

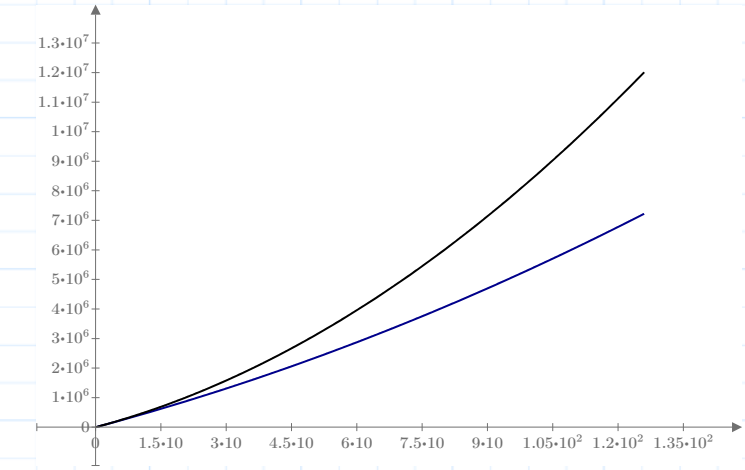
$$\frac{v_{total}\left(s_{blk\_m}, n_{blk}, \sigma_0\right)}{v_{total\_sql}\left(s_{blk\_m}, n_{blk}, \sigma_0\right)} = 99.71\%$$ - *сравнение для разбиения по частям*

$$\frac{v_{total}\left(s_{chp\_m}, n_{chp}, \sigma_0\right)}{v_{total\_sql}\left(s_{chp\_m}, n_{chp}, \sigma_0\right)} = 60.13\%$$ - *сравнение для разбиения по главам*

$$\frac{v_{total}\left(s_{par\_m}, n_{par}, \sigma_0\right)}{v_{total\_sql}\left(s_{par\_m}, n_{par}, \sigma_0\right)} = 32.23\%$$ - *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



n

$$\frac{v_{total}\left(s_{chp\_m}, n, \sigma_0\right)}{v_{total\_sql}\left(s_{chp\_m}, n, \sigma_0\right)}$$

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 реализоваывать 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 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
```

Выводы
По результатам сравнения эффективности neo4j и MySQL получены следующие результаты:

- Увеличение объема при увеличении чиста текстов - o(n^2).
- Отсечение части связей приводит к уменьшению места на o(σ), при росте o(n^2) это не имеет смысла
- neo4j тратит немного меньше места на хранение данных за счет динамического выделения памяти и отсутствия индексов

$$\frac{v_{total}\left(s_{chp\_m}, n_{chp}, \sigma_0\right)}{v_{total\_sql}\left(s_{chp\_m}, n_{chp}, \sigma_0\right)} = 60.13\%$$

- Разбиение большого текста на небольшие фрагменты нерационально и приводит к резкому возрастанию объема БД.
- При реализации алгоритмов на SQL эфективность может быть ниже в o(m/n) раз. Это существенная разница, т.к. для больших текстов m >> n .
  Кроме того, периодически необходимы трудоемкие операции вроде UNION.