類似結合算法L2APのブロック分割とハッシュ結合による強化

吉村龍之介† 大森 匡† 新谷 隆彦† 藤田 秀之†

† 電気通信大学大学院情報理工学研究科 〒 182-8585 東京都調布市調布ケ丘 1-5-1

E-mail: †yoshimura@hol.is.uec.ac.jp, ††{omori,shintani,fujita}@is.uec.ac.jp

あらまし データベースにおける類似結合演算とは、与えられたデータベースから、類似したレコードの対を全て求める問題であり、データベースの重複除去やクラスタリングなどで用いられる基本的で高コストな結合演算の 1 つである. 本稿では、逐次実行向けの高速な類似結合として最近提案された L2AP に注目し、その特性を調べ、ブロック分割とハッシュ結合を導入する事で、データベース演算としての能力を強化する. 特に R-S ジョインの場合に、L2AP をブロック分割とハッシュ結合に基づき強化した手法、L2AP/HJ を提案する.

キーワード データベース, 類似結合, ハッシュ結合

1. はじめに

データベースにおける類似結合演算 [3] とは、与えられたデータベースから、類似したレコードの対を全て求める問題であり、データベースの重複除去やクラスタリングなどで用いられる基本的で高コストな結合演算の 1 つである.

計算アルゴリズムとしては、LSH[4] や Prefix filtering[3]、All_Pairs[2]、L2AP[1] などが知られている. 本稿では、逐次実行向けの高速な類似結合として最近提案された L2AP に注目し、その特性を調べ、ブロック分割とハッシュ結合を導入する事で、データベース演算としての能力を強化する.

具体的にはセルフジョインで議論されている L2AP について、まず、L2AP で使われているコラム戦略、ノルム戦略それぞれの加速能力を調べ、データ集合 D を部分集合 $D_1, D_2, ..., Ds$ に水平分割して、セルフジョインにおける候補フィルタリング能力の強化を試みる.

次に、その結果に基づいて、R-S ジョインの場合に L2AP を適用する事を考え、ハッシュ結合戦略に基づいて R 側だけをインデックス化するように L2AP を変形し、コラム戦略のフィルタリング能力を強化した手法 L2AP/HJ を提案する.

以下、本稿では 2. で類似結合の導入と L2AP の概要の説明を行い、3. で L2AP の各戦略を解説した後、4. でセルフジョインの場合の水平分割手法を述べて、5. で評価を行う. その結果に基づいて、6. で R-S ジョインの場合の L2AP の動作を取り上げ、改良手法として L2AP/HJ を提案・評価し、最後に 7. でまとめを述べる.

2. 類似結合と L2AP

2.1 導 入

類似結合とは次元数 m の実数ベクトル n 個からなる集合 D が与えられた時、類似度関数 $sim(\boldsymbol{x}, \boldsymbol{y})$、類似度閾値 t として、$\boldsymbol{x}, \boldsymbol{y} \in D$、$sim(\boldsymbol{x}, \boldsymbol{y}) \geqq t$ となるような全ベクトルペア $(\boldsymbol{x}, \boldsymbol{y})$ と類似度 $sim(\boldsymbol{x}, \boldsymbol{y})$ を計算する問題である.

本稿では解法として L2AP[1] を扱い、類似度関数をコサイン類似度とする. 記号として $\boldsymbol{x}$ の第 j 次元を x_j と書き、$\boldsymbol{x} = \langle x_1, x_2, ..., x_m \rangle$ とし、$\boldsymbol{x}$ の L2 ノルム $||\boldsymbol{x}||$ を用いると、類似度関数 $cos(\boldsymbol{x}, \boldsymbol{y})$ は $cos(\boldsymbol{x}, \boldsymbol{y}) = \frac{\boldsymbol{x} \cdot \boldsymbol{y}}{||\boldsymbol{x}|| \times ||\boldsymbol{y}||}$ である. 一般性を失わずに $\boldsymbol{x}$ の L2 ノルム $||\boldsymbol{x}|| = 1$ で正規化する. $\boldsymbol{x}$ の第 j 要素を x_j と表記すると
$cos(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x} \cdot \boldsymbol{y} = \sum_{j=1}^{m} x_j \times y_j$ となる.

2.2 記 号

記号として以下を用いる.

ベクトル $\boldsymbol{x}$ について第 j 要素 x_j から第 m 要素 x_m を 0 としたベクトルを $\boldsymbol{x'_j} = \langle x_1, x_2, ..., x_{j-1}, 0, ..., 0 \rangle$、第 1 要素 x_1 から 第 $j-1$ 要素 x_{j-1} を 0 としたベクトルを $\boldsymbol{x''_j} = \langle 0, 0, ..., x_j, ..., x_m \rangle$ と表記する. つまり $\boldsymbol{x} = \boldsymbol{x'_j} + \boldsymbol{x''_j}$ である.

$|\boldsymbol{x}|$: $\boldsymbol{x}$ の非ゼロ要素数.$\boldsymbol{x}$ のサイズ.

$||\boldsymbol{x}|| = \sqrt{\sum_{j=1}^{m} x_j^2}$ は正規化したため $||\boldsymbol{x}|| = 1$ だが、$||\boldsymbol{x'_j}||$ などを計算するときに L2 ノルムは用いられる.

$I_j : x_j \neq 0$ となる (x, x_j) を記録する逆引きインデックス. $\boldsymbol{x}$ は m 次元なので $I_1, I_2, ..., I_m$ と表される.

j を決めた時、D の全ベクトル $\boldsymbol{x}$ についての第 j 要素 x_j の最大値を cw_j と表記する. 即ち $cw_j = max\{x_j | \boldsymbol{x} \in D\}$ である.

また、ベクトル $\boldsymbol{x}$ について、要素 $x_1, ..., x_m$ のうち最大値を rw_x と表記する. 即ち $rw_x = max\{x_j | j = 1, ..., m\}$ である.

2.3 L2AP の処理概要

以下に L2AP の処理概要を示す.

まず前処理として Algorithm1 の 1 行目で D の全 $\boldsymbol{x}$ を rw_x の降順でソートを行う.

次に Algorithm1 の 5 行目で D の各ベクトル $\boldsymbol{x}$ を FindMatches 関数に送る. FindMatches 関数は、既に $I_1 \sim I_m$ に索引付けされているベクトルの中で $\boldsymbol{x}$ に対し、$cos(\boldsymbol{x}, \boldsymbol{y}) \geq t$ を満たすベクトル $\boldsymbol{y}$ を全て求める関数である.

最後に Algorithm1 の 8 行目以降の Index Construction で以後追加されるベクトルとの類似度計算を正しく行う事が出来るように、$\boldsymbol{x}$ の索引付けを行う. このとき索引付けされた要素が $\boldsymbol{x''_j}$、索引付けされなかった要素が $\boldsymbol{x'_j}$ である.

Algorithm 1 L2AP(D, t)

Reorder D rows in decreasing rw order//データ生成時
Reorder D's columns in decreasing frequency order//データ生成時
$O \leftarrow \emptyset$; $I_1, I_2, ..., I_m \leftarrow \emptyset$; $c\hat{w}_1, c\hat{w}_2, ..., c\hat{w}_m \leftarrow \emptyset$
for each $\boldsymbol{x} \in D$ from 1 to n **do**
  $O \leftarrow O \cup$ Find_MatchesL2AP($\boldsymbol{x}, I_1, ..., I_m, ps, c\hat{w}, t$)
  $b_1 \leftarrow 0$; $b_t \leftarrow 0$; $b_3 \leftarrow 0$
  //Index Construction
  for each $j = 1$ to m s.t.$x_j > 0$ **do**
    $pscore \leftarrow min(b_1, b_3)$
    $b_1 \leftarrow b_1 + x_j \times min(cw_j, rw_x)$ //$b_1(k)$ の計算
    $b_t \leftarrow b_t + x_j^2$; $b_3 \leftarrow \sqrt{b_t}$ //$b_3(k)$ の計算
    if $min(b_1, b_3) \geqq t$ **then**
      $ps[x] \leftarrow pscore$ **if** $ps[x] = 0$
      $I_j \leftarrow I_j \cup \{(x, x_j, ||\boldsymbol{x}'_{\boldsymbol{j}}||)\}$
      $x_j \leftarrow 0$
    end if
  end for
end for
return O

3. L2AP の戦略について

3.1 Index Construction

まずコラムベースの索引付け条件 (b_1) を説明する.

インデックスへの索引付けで $\boldsymbol{x}$ より後にくる $\boldsymbol{y}$ を考えたとき、類似度閾値 t を満たす組 $(\boldsymbol{x}, \boldsymbol{y})$ の内積は $y_j \leqq cw_j$ かつ $y_j \leqq rw_x$ であるので

$\boldsymbol{x} \cdot \boldsymbol{y} \leqq \sum_{j=1}^{m} x_j \times min(rw_x, cw_j)$ となる.

ソート順で $\boldsymbol{x}$ の後ろにくる $\boldsymbol{y}$ について、もし $\boldsymbol{x}''_{\boldsymbol{k}} = \langle 0, 0, ..., 0, x_k, ..., x_m \rangle$ と $\boldsymbol{y}''_{\boldsymbol{k}} = \langle 0, 0, ..., 0, y_k, ..., y_m \rangle$ の内積が 0 なら $\boldsymbol{x} \cdot \boldsymbol{y} = \boldsymbol{x}'_{\boldsymbol{k}} \cdot \boldsymbol{y}'_{\boldsymbol{k}}$ である.

さらにもし $\boldsymbol{x}'_{\boldsymbol{k}} \cdot \boldsymbol{y}'_{\boldsymbol{k}} < t$ なら $\boldsymbol{y}$ は不要である.

ゆえに $B_1 = argmin_K(\sum_{j=1}^{K} x_j \times min(cw_j, rw_x) \geqq t)$ とし、$\boldsymbol{x}$ の $x_{B_1} \sim x_m$ だけを索引付けすれば、十分である.

実装方法としては、$b_1(k) = \sum_{j=1}^{k} x_j \times min(rw_x, cw_j) \geqq t$ となる最初の k から索引付けする.

次にノルムベースの索引付け条件 (b_3) を説明する.

$\boldsymbol{x} \cdot \boldsymbol{y} = \boldsymbol{x}'_{\boldsymbol{j}} \cdot \boldsymbol{y} + \boldsymbol{x}''_{\boldsymbol{j}} \cdot \boldsymbol{y}$ なので、もし $\boldsymbol{x}''_{\boldsymbol{j}} \cdot \boldsymbol{y} = 0$ ならば

$\boldsymbol{x} \cdot \boldsymbol{y} = \boldsymbol{x}'_{\boldsymbol{j}} \cdot \boldsymbol{y}$.

さらに、シュワルツの不等式より

$\boldsymbol{x} \cdot \boldsymbol{y} = \boldsymbol{x}'_{\boldsymbol{j}} \cdot \boldsymbol{y} \leqq ||\boldsymbol{x}'_{\boldsymbol{j}}|| \times ||\boldsymbol{y}||$.

そして入力ベクトルが単位ベクトルである為

$\boldsymbol{x} \cdot \boldsymbol{y} = \boldsymbol{x}'_{\boldsymbol{j}} \cdot \boldsymbol{y} \leqq ||\boldsymbol{x}'_{\boldsymbol{j}}|| \times ||\boldsymbol{y}|| = ||\boldsymbol{x}'_{\boldsymbol{j}}||$ となる.

ゆえに $B_3 = argmin_K(\sqrt{\sum_{j=1}^{K} x_j^2} \geqq t)$ とした時、$\boldsymbol{x}$ の $x_{B_3} \sim x_m$ だけを I に索引付けする.

実装方法としては、$b_3(k) = \sqrt{\sum_{j=1}^{k} x_j^2} \geqq t$ となる最初の k から索引付けする.

まとめると L2AP では $b_1 \geqq t$ と $b_3 \geqq t$ を共に満たす要素以降を索引付けする.

3.2 Candidate Generation

FindMatches($\boldsymbol{x}, I, t$) は入力ベクトル $\boldsymbol{x}$ より前にいる $\boldsymbol{y}$ のうち $cos(\boldsymbol{x}, \boldsymbol{y}) \geqq t$ となる $\boldsymbol{y}$ を正しく求める. そのため候補生成を行った後、その候補の検証を行う. 候補生成は remscore フィルタとサイズフィルタから構成されるが本稿では remscore フィルタを説明する.

Algorithm 2 Find_MatchesL2AP($\boldsymbol{x}, I_1, ..., I_m, ps, c\hat{w}, t$)

$A \leftarrow \emptyset$; $M \leftarrow \emptyset$
$sz \leftarrow t / rw_x$
$rs_3 \leftarrow \sum_{j=1}^{m} x_j \times c\hat{w}_j$; $rs_t \leftarrow 1$; $rs_4 \leftarrow 1$
$rw'_{x_j} \leftarrow max(\boldsymbol{x}'_{\boldsymbol{j}})$, $\sum'_{x_j} \leftarrow \sum_{i=1}^{j-1} x_i$, $\forall x_j > 0$
//Candidate Generation
for each $j = m$ to 1 s.t. $x_j > 0$ **do**
  $I_j \leftarrow I_j \backslash \{(y, y_j, ||\boldsymbol{y}'_{\boldsymbol{j}}||)\}$, $\forall y$ s.t. $|\boldsymbol{y}| \times rw_y < sz$//省略
  for each $((y, y_j, ||\boldsymbol{y}'_{\boldsymbol{j}}||) \in I_j$ **do**
    if $A[y] > 0$ **or** $min(rs_3, rs_4) \geqq t$ **then**
      $A[y] \leftarrow A[y] + x_j \cdot y_j$
      if $A[y] + ||\boldsymbol{x}'_{\boldsymbol{j}}|| \times ||\boldsymbol{y}'_{\boldsymbol{j}}|| < t$ **then**
        $A[y] \leftarrow 0$
      end if
    end if
  end for
  $rs_3 \leftarrow rs_3 - x_j \times c\hat{w}_j$
  $rs_t \leftarrow rs_t - x_j^2$; $rs_4 \leftarrow \sqrt{rs_t}$
end for
//Candidate Verification
for each y s.t.$A[y] > 0$ **do**
  next y **if** $A[y] + ps[y] < t$
  next y **if** $A[y] + min(rw_x \times \sum'_y, rw'_y \times \sum_x) < t$
  find first p s.t. $y_p \in \boldsymbol{y}' \wedge y_p > 0 \wedge x_p > 0$
  $s \leftarrow A[y] + x_p \times y_p$
  next y **if** $s + min(rw'_{x_p} \times \sum'_{y_p}, rw'_y \times \sum'_{x_p}) < t$
  for each $j < p$ s.t. $y_j > 0 \wedge x_j > 0$ **do**
    $s \leftarrow s + x_j \times y_j$
    if $s + ||\boldsymbol{x}'_{\boldsymbol{j}}|| \times ||\boldsymbol{y}'_{\boldsymbol{j}}|| < t$ **then**
      next y
    end if
  end for
  if $s \geqq t$ **then**
    $M \leftarrow M \cup \{(x, y, s)\}$
  end if
end for
$c\hat{w}_j \leftarrow max(x_j, c\hat{w}_j), \forall x_j > 0$
return M

3.2.1 remscore フィルタ

まずコラムベースの remscore(rs_3) を説明する. ソート順で $\boldsymbol{x}$ より前、すなわち既に I に索引付けされている $\boldsymbol{y}$ に対して候補ペアの生成条件として、既に I に索引付けされたベクトル $\boldsymbol{x}$ の第 j 要素 x_j の最大値 $c\hat{w}_j$ を用いて、式

$rs_3(k) = \sum_{j=1}^{k} x_j \times c\hat{w}_j$

を用意し、I_m から I_1 へのスキャンの判断に使用する.

具体的には $x_j \times c\hat{w}_j$ を $rs_3(m)$ から繰り返し減算していく.

$A[y] \leftarrow A[y] + x_j \cdot y_j (j = m, ..., j_0)$ で I_m から I_{j_0} までの $\boldsymbol{x} \cdot \boldsymbol{y}$ を計算していく. すなわち $A[y] = \sum_{j=j_0}^{m} x_j \cdot y_j$.

$j = m, ..., j_0$ までループが回った時点で、$rs_3(j_0)$ はまだ処理が行われていない $I_1 \sim I_{j_0-1}$ で取り得る内積の上界である. 従って rs_3 が類似度閾値 t を満たさなくなった時点で、$I_1 \sim I_{j_0-1}$ にしか非ゼロ要素を持たない候補ベクトル $\boldsymbol{y}$ が類似度閾値を満たさない事は自明である. よって以後の候補生成を省略出来る.

次にノルムベースの remscore(rs_4) を説明する.

まだ処理が行われていない $I_1 \sim I_{j-1}$ で $\boldsymbol{x'_j}$ が取り得る L2 ノルム $||\boldsymbol{x'_j}||$ を用いる.

$\boldsymbol{x} \cdot \boldsymbol{y} = \boldsymbol{x'} \cdot \boldsymbol{y} + \boldsymbol{x''} \cdot \boldsymbol{y} = \boldsymbol{x'_{j_0}} \cdot \boldsymbol{y} + \boldsymbol{x''_{j_0}} \cdot \boldsymbol{y}$ であり, 今 $I_m \sim I_{j_0}$ のループが終了した時点で $A[y] = 0$ ならば、$\boldsymbol{x''_{j_0}} \cdot \boldsymbol{y} = 0$ である.

よって $\boldsymbol{x} \cdot \boldsymbol{y} = \boldsymbol{x'_{j_0}} \cdot \boldsymbol{y} \leqq ||\boldsymbol{x'_{j_0}}|| \times ||\boldsymbol{y}|| = ||\boldsymbol{x'_{j_0}}||$ である.

従って、候補ペアの生成条件のスコア rs_4 として $rs_4(k) = \sqrt{1 - \sum_{j=k}^{m} x_j^2 (k = m, ..., 1)}$ を用意し、$rs_4 \geqq t$ となる k の範囲でのみ $A[y]$ の計算を行う.

以上 2 つの条件より候補の生成を行う.

3.2.2 Candidate Verification の前倒し

L2AP では Candidate Generation で候補確認を行う事で効率化を図る.

$I_m \sim I_{j_0}$ までの範囲で計算した内積 $\boldsymbol{x} \cdot \boldsymbol{y}$ の値を $A[y] = \sum_{j=j_0}^{m} x_j \cdot y_j$ と表す.

この時 $\boldsymbol{x} \cdot \boldsymbol{y} = \boldsymbol{x'_{j_0}} \cdot \boldsymbol{y'_{j_0}} + A[y]$ である. $\boldsymbol{x'_{j_0}} \cdot \boldsymbol{y'_{j_0}} \leqq ||\boldsymbol{x'_{j_0}}|| \times ||\boldsymbol{y'_{j_0}}||$ を I_m から I_{j_0} のループ内で用いる事で、$A[y] + ||\boldsymbol{x'_{j_0}}|| \times ||\boldsymbol{y'_{j_0}}|| \geqq t$ を満たさない $\boldsymbol{y}$ を ($\boldsymbol{x} \cdot \boldsymbol{y} \geqq t$ となる $\boldsymbol{y}$ の) 候補から外し次の候補に移行出来る.

3.3 Candidate Verification

Candidate Generation で求めた候補について $cos(\boldsymbol{x}, \boldsymbol{y})$ を求めて、真に $cos(\boldsymbol{x}, \boldsymbol{y}) \geqq t$ であるかを検証する.

各フィルタによって生成された候補ペアの絞り込みを行うが、今回は変更を行わないので説明を省略する.

4. 水平分割の提案

4.1 実　　装

元論文 [1] のアルゴリズムを再現したが、サイズフィルタは計算削減効果よりオーバーヘッドの方が処理時間に影響するので意図的に外している.

またデータセットは疎ベクトル集合を想定しているが、入力ベクトルが疎ベクトルのままだと Candidate Verification での内積導出の際に、入力ベクトルの要素次元値の二分探索が必要となり、処理時間に影響が生じる. その為、Index Construction で入力ベクトルの密ベクトル変換を行って対応している.

4.2 水 平 分 割

L2AP ではデータベースのセルフジョインにおける類似結合演算を想定している.

よってデータセットの水平分割を行うことで、セルフジョインにおける類似結合演算の強化を図る.

具体的にはデータセット D のレコード集合に対して、領域分割数を s とした場合、ソート順に $D_1, D_2, ..., D_p, ..., D_s$ に分割する.

この際、領域 $D_p (p = 1, 2, ..., s)$ における第 j 次元の $\boldsymbol{x}$ の最大値を cwp_j とおく. この領域毎の cwp_j を用いて L2AP アルゴリズムの戦略強化を図る.

4.2.1 Index Construction のフィルタリング強化

各領域毎の cw_j を用いることで、Index Construction で索引付け時に用いるコラムベースの条件 b_1 を強化出来る.

コラムベースでは $b_1(k) = \sum_{j=1}^{k} x_j \times min(rw_x, cw_j) \geqq t$ となる最初の k から索引付けを行っているが、入力レコードより後から追加されるレコードとの積の最大値を考えるのであれば入力レコードより後ろの領域内の cw_j で十分である.

よって領域分割数を s、入力レコード $\boldsymbol{x}$ の所属領域を D_p とした場合、$b_1(k) = \sum_{j=1}^{k} x_j \times min(rw_x, max(cwp_j, ..., cws_j)) \geqq t$ となる最初の k から索引付けを行なえばよい.

4.2.2 Candidate Generation のフィルタリング強化

また各領域毎の cw_j を用いることで、Candidate Generation のコラムベースの remscore を強化出来る.

コラムベースでは $rs_3(k) = \sum_{j=1}^{k} x_j \times c\hat{w}_j$ を用意し、$x_j \times c\hat{w}_j$ を $rs_3(m)$ から繰り返し減算していくことで、まだ処理が行われていない $I_1 \sim I_{j_0-1}$ で取り得る内積の上界を求めていたが、内積の上界値を考えるのであれば候補 y を含む領域 D_p の cw_j である cwp_j と x_j の積を考えれば十分である.

よって領域分割数を s、候補レコードの領域を D_p、領域 D_p の cw_j を cwp_j とした場合、$rs_3(k, p) = \sum_{j=1}^{k} x_j \times cwp_j$ を用意し、$x_j \times cwp_j$ を繰り返し減算し内積の上界を求めればよい.

5. セルフジョインの実験

5.1 概　　要

DBLP のデータを用いてナイーブ、コラムベース、ノルムベース、水平分割方式での実験を行った.

L2AP では本来、コラム戦略とノルム戦略を併用しているが本実験では各戦略の特性を検証するために、コラムベースはコラム戦略、ノルムベースはノルム戦略を単独で用いた.

以下に各手法の概要を示す.

- ナイーブ:この方法はインデックスジョインに相当する手法であり、データセット D の全要素をインデックスに登録し、入力ベクトルとの共通項を全て計算し全ペアに対し、閾値判定を行っている. この為、性能は閾値に依存しない.
- コラム:この方法は索引付け時に b_1 のみ、インデックスとの照合時に rs_3 のみを用いたレコードの最大値と各次元の最大値を使用した手法である.
- ノルム:この方法は索引付け時に b_3 のみ、インデックスとの照合時に rs_4 のみを用いた L2 ノルムに基づいた手法である.
- 水平分割:この方法はコラムベースの索引付け条件とインデックスとの照合条件を、データセット D の水平分割によって強化した手法である.

5.2 DBLP2

検証用のデータセットとして DBLP の 2015 年 1 月〜12 月の 1 年分の論文 240595 件のデータを用意した. 論文タイトルと著者名を単語の出現頻度に基づいて重さ付けをした上で、ベクトル化し疎ベクトルとした.

このデータセットを以後 DBLP2 と呼ぶ.

表 1 にデータセットの詳細な情報を示す. DBLP2 の全次元数は 154977、1 レコード辺りの平均ビット長は 16 ビットとなっている.

また表 2 に DBLP2 において閾値 θ を満たすペア数を示す. 閾値 0.2 で類似ペア数が多すぎる状況となっている為、負荷テ

表 1　データセット

データセット	レコード数	次元数	概要	ビット長
DBLP2	240595	154977	論文名+著者名	平均 16 ビット

表 2　DBLP2 において閾値 θ を満たす類似ペア数

データセット	閾値 0.1	閾値 0.2	閾値 0.3	閾値 0.4	閾値 0.5	閾値 0.6	閾値 0.7	閾値 0.8	閾値 0.9
DBLP2	12548199	821975	255464	120933	64257	35144	19306	10999	7082

表 3　DBLP2 において閾値 0.4 の場合の処理結果

戦略	処理時間 [s]	IC[s]	CG[s]	CV[s]	$A[y] \neq 0$ となるペア数	21 行目を通るペア数	22 行目を通るペア数	27 行目の回数	エントリ数
ナイーブ	67.91	0.14	56.37	11.40	2656380908	-	-	-	3418303
コラム	5.40	0.27	3.93	1.20	18921192	16411463	13475364	328566	1881386
ノルム	6.79	0.26	6.16	0.37	18451036	17780952	4855781	215859	2133424
水平分割数 15	4.71	0.28	3.01	1.42	18820685	16913559	16298260	505217	1599411

ストを抜きで考えると、閾値 0.3 以上の状況で考えるのが妥当である.

実験ではデータセット内の次元に 1 つしか立っていない要素を処理に加えても無意味となるので、各レコードの要素のうち他レコードと 1 つ以上共通する要素のみで処理を行った.

5.3　DBLP2 における実験

DBLP2 を用いてナイーブ、コラム、ノルム、水平分割で実験を行った.

図 1 は戦略毎の処理時間を示した図となっている. ナイーブ、コラムベース、ノルムベース共に概ね元論文 [1] と同様の振舞いを示した. 一方提案手法の水平分割方式は、強化元のコラムベースより多少良好な結果を示した. 閾値が高い場合、ノルムベースが最速で、閾値が 0.5 以下では水平分割方式が最速な状態となった. しかしながら処理時間において劇的な優位性を示していない為、より処理状況を詳しく追っていくこととする.

図 2 は DBLP2 の戦略毎のインデックスエントリ数を示した図となっている. 処理時間同様に閾値が高い場合ノルムベースのエントリ数が最も少く、閾値 0.7 以下では水平分割方式が最もエントリ数が少ない結果となった. エントリ数を減らすという強化の結果が確認できる結果となった.

次に候補生成部の振る舞いを検証するため候補生成部の A[y] の更新回数を計測した. その図が図 3 である. A[y] の更新回数同様に閾値が高い場合はノルムベースが最も候補生成部のコストが少なく、閾値 0.7 以下で水平分割方式のコストが最も少ない状況となった.

水平分割で狙った候補生成部のコスト削減効果は現れている

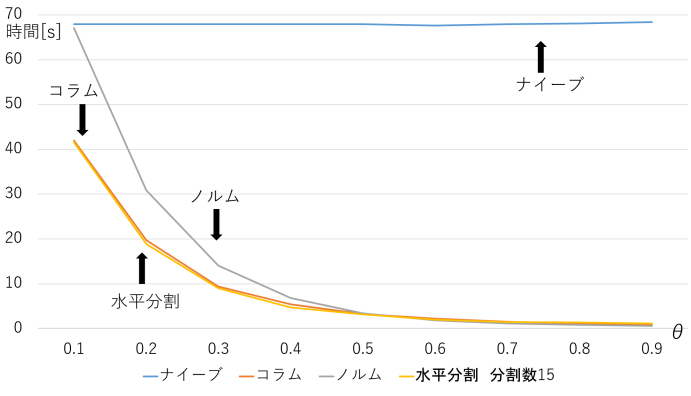

図 1　DBLP2 の戦略毎の処理時間

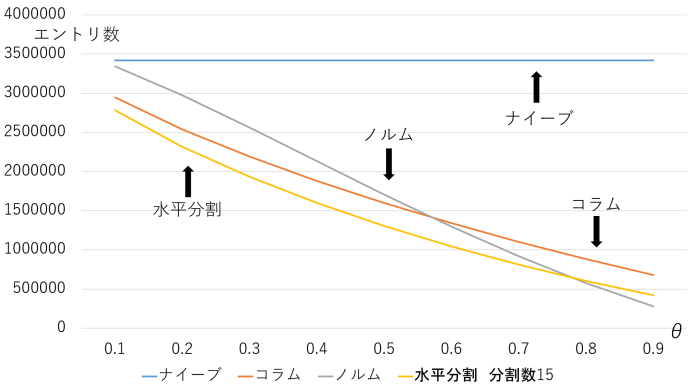

図 2　DBLP2 の戦略毎のインデックスエントリ数

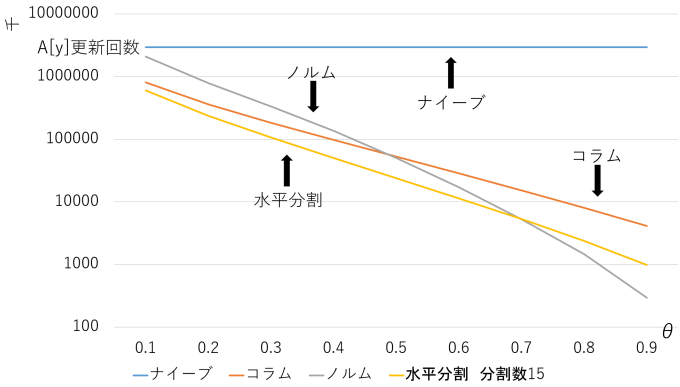

図 3　DBLP2 の戦略毎の候補生成時 A[y] 更新回数

事が分かる. しかしながら候補生成部のコスト削減量と比較して処理時間の削減は小さいものとなっている.

5.4　DBLP2 における実験結果考察

DBLP2 の実験の結果、水平分割により、候補生成部の A[y] の更新回数を削減出来る事が確認できた. 一方で処理時間の削減量は少ないという状況が判明した. そこで原因を検証するため 2 つの側面から検討を行った.

5.4.1　候補生成部のコスト

1 つ目は候補生成部のコストについてである. 水平分割方式の場合、領域の数だけ remscore を導出する必要がある為、オーバーヘッドが増加したと考えられる. 初期値導出を前倒しで行った結果、分割数 15 で 0.5 秒程度のオーバーヘッドが生じていた. その為、特に閾値が高い場合、候補生成部にかかる計算時間が少ないためこのオーバーヘッドの分、水平分割方式の処理

時間結果が悪くなっていた事が分かった.

また候補生成部において A[y] の更新をスキップする為には必ずインデックスへの照合が必要となるという点も影響しているのではないかと考えられる.

水平分割を行った結果、A[y] の更新回数を削減したとしても、インデックスへの照合というループを必ず挟んでしまう為に、処理オーバーヘッドが削れない分だけ処理時間への影響が小さくなってしまったと考えられる.

5.4.2 候補検証部のコスト

2 つ目は候補検証部のコストについてである. 候補検証部では索引付けしていない領域に依存したフィルタが存在する. その為、索引付け数が少ないとこのフィルタの能力が下がり候補検証部での候補ペアの削減能力が低下する事が考えられる.

実際に候補検証部の動作を確認するために閾値 0.4 の時の候補ペア数の推移を調べた.

表 3 は DBLP2 における閾値 0.4 の場合の処理結果である. 左部は全体の処理時間と Index Construction(IC)、Candidate Generation(CG)、Candidate Verification(CV) の処理時間を示しており、右部では CV のフィルタ毎の候補ペア数とインデックスエントリ数を示している. 表 3 のエントリ数と 22 行目を通るペア数に着目するとエントリ数が最も少ない水平分割で通るペア数が最大で、エントリ数がナイーブの次に多いノルムでは通ペア数が最も少なくなっている. その結果 CV の処理時間は水平分割の場合が最も長く、ノルムの場合が最も短くなっている. つまり候補生成時に削ったコストの一部が候補検証のコストに流れている為、トータルの時間では大きな差異が表れなかったと考えられる.

6. R-S ジョインに適した L2AP/HJ の提案

6.1 R-S ジョインと L2AP

前節では L2AP が想定している、データベースのセルフジョインにおける、類似結合の評価・実験を行った.

それを踏まえ、L2AP の持つ特性に着目し、データベース演算としての能力を強化する為に、R-S ジョインに適した L2AP/HJ を提案する. ここで R-S ジョインとは 2 つの実数ベクトル集合 R と S が与えられた時、類似度関数 $sim(\boldsymbol{x},\boldsymbol{y})$、類似度閾値 t として、$\boldsymbol{x} \in R$、$\boldsymbol{y} \in S$、$sim(\boldsymbol{x},\boldsymbol{y}) \geqq t$ となるような全ベクトルペア $(\boldsymbol{x},\boldsymbol{y})$ と類似度 $sim(\boldsymbol{x},\boldsymbol{y})$ を計算する問題である.

6.2 L2AP における処理オーバーヘッドなど特性の着目点

6.2.1 L2AP の動作特性

L2AP では FindMatchesL2AP 関数で既にインデックスに登録されているレコードと入力レコードの照合を行い、照合後に入力レコードのインデックス登録を行う. インデックスとの照合時には内積計算条件を設ける事で、内積計算コストの削減を行っているが、その際、必ずインデックスへの照合を行ってから内部計算省略の判定を行っている. その為、インデックス登録量が多い場合、インデックス照合によるオーバーヘッドが無視出来ない物となる. よってインデックス登録量を減らす事で候補生成時のコストを減らす事が可能となる.

6.2.2 L2AP/MJ

L2AP でデータセット $R \cup S = D$ として R-S ジョインを実行する場合が最も初歩的な手法である. この時、インデックス登録は R、S 両データセットに関して行うが、候補生成、候補検証に関しては R のレコードと S のレコードとの組み合わせの場合のみ処理を行う. また各フィルタに用いる cw_j としては $cw_j = max\{x_j|\boldsymbol{x} \in R \cup S\}$、$c\hat{w}_j = max\{x_j|\boldsymbol{x} \in I\}$ として用いる. 以後、この手法を L2AP/MJ と呼ぶ. また L2AP のコラム戦略での動作を L2AP/MJ コラム、ノルム戦略での動作を L2AP/MJ ノルムと表記する.

L2AP/MJ コラムのとき、データセットを $R \cup S = D$ として R-S ジョインを実行する場合、セルフジョイン時と同様 rw_x と cw_j によって索引付けフィルタ b_1 と候補生成フィルタ rs_3 を決定出来る. よって $b_1(k)$ と $rs_3(k)$ はセルフジョインでの L2AP の動作時から変更はない.

また L2AP/MJ ノルムの各フィルタはセルフジョインで L2AP を動作させる場合におけるノルム戦略のフィルタと戦略的には変わらない. これはノルム戦略の場合、入力レコードと候補レコードによってフィルタ能力が決定してしまう為、データセットの範囲に関わらない為である. よって b_3 と rs_4 においてもセルフジョインでの L2AP の動作時から変更はない.

6.3 R-S 結合におけるハッシュ結合による L2AP,L2AP/HJ の提案

L2AP において、インデックス登録量を減らす事を考えた時、L2AP を R-S ジョインに動作させる場合を考える. R-S ジョインでの動作でハッシュ結合 [5],[6] を使うことにすると、R のみのインデックスを用意すればよい. その為、$R \cup S$ 全体のインデックスを用意するマージソート結合での動作である L2AP/MJ と比べ、インデックス登録量が大幅に少なくなるので、候補生成時のインデックス照合量を大きく減らす事が出来るはずであるので、ハッシュ結合によるこの手法を以後 L2AP/HJ と表記する事とする.

以後 R の cw_j を cwR_j、S の cw_j を cwS_j と表記する.

これら cwR_j と cwS_j を用いることで L2AP の索引付け条件と候補生成時の remscore を強化する事が出来る.

6.3.1 Index Construction フィルタリングの強化

L2AP/HJ では cwS_j を用いることで、Index Construction で索引付け時に用いるコラムベースの条件 b_1 を強化出来る.

L2AP/MJ コラムでは $b_1(k) = \sum_{j=1}^{k} x_j \times cw_j \geqq t$ となる最初の k から索引付けを行うが、L2AP/HJ では S のレコードとの積の最大値を考えるのであれば S の cw_j で十分である.

よって cwS_j を用いて、$b_1\_\mathrm{HJ}(k) = \sum_{j=1}^{k} x_j \times cwS_j \geqq t$ となる最初の k から索引付けを行なえばよい.

6.3.2 Candidate Generation フィルタリングの強化

L2AP/HJ では R の cw_j を用いることで、Candidate Generation のコラムベースの remscore を強化出来る.

元々のコラムベースでは $rs_3(k) = \sum_{j=1}^{k} x_j \times c\hat{w}_j$ を用意し、$x_j \times c\hat{w}_j$ を $rs_3(m)$ から繰り返し減算していくことで、まだ処理が行われていない $I_1 \sim I_{j_0-1}$ で取り得る内積の上界を求めていた. 一方で R-S ジョインの場合、内積の上界値を考えるの

であればインデックスに登録する R の cw_j と x_j の積を考えれば十分である.

よって L2AP/HJ では cwR_j を用いて、$rs_3(k) = \sum_{j=1}^{k} x_j \times cwR_j$ を用意し、$x_j \times cwR_j$ を繰り返し減算し内積の上界を求めればよい. 以後この $rs_3(k)$ を rs_3_HJ(k) と表記する.

加えて、セルフジョインの場合と同様に水平分割を行うことで更なる強化が出来る. R-S ジョインの場合、領域分割数を s とした場合、インデックスに登録する R のデータ集合を $R_1, R_2, ..., R_p, ..., R_s$ として水平分割する.

よって候補レコードの領域を $R_p(p = 1, 2, ..., s)$、領域 R_p の cw_j を $cwRp_j$ とした場合、rs_3_HJ$(k, p) = \sum_{j=1}^{k} x_j \times cwRp_j$ を用意し、$x_j \times cwRp_j$ を繰り返し減算し内積の上界を求めればよい.

6.4 L2AP/HJ のアルゴリズムの全体概要

L2AP/HJ では R のみをインデックスに登録する為に L2AP/MJ と比較し全体のインデックス登録量を少なくする事が出来る為、インデックス構築に必要なメモリ使用量を大幅に少なくする事が出来る.

加えて候補生成時のインデックス照合量が少なくて済む為、アルゴリズムのオーバーヘッドを削減する事が出来る.

また前述したように索引付け条件と、候補生成時の remscore 条件を直接強化することも出来る為、類似結合演算コストそのものを削減する事が出来る.

アルゴリズムの変更点としては入力レコードが R 側の場合、FindMatches 関数への入力を省略し、Index Construction のみを行い、S 側のレコードの場合、FindMatches 関数への入力のみを行うように動作させる.

また R と S は各々、rw_x でソートを行っておき、R の入力後に S を入力する.

各戦略の変更については前述したように b_1 と rs_3 の強化を行う.

6.5 評 価 実 験

6.5.1 概　　要

DBLP のデータを用いて、R-S ジョインにおけるナイーブ、L2AP/MJ コラム、L2AP/MJ ノルム、L2AP/HJ、L2AP/HJ 水平分割での実験を行った. 以下に各手法の概要を示す.

- ナイーブ:この方法はインデックスジョインに相当する手法であり、データセットの全要素をインデックスに登録し、R と S のレコード間の結合を全て計算し、全ペアに対して閾値判定を行っている. この為、性能は閾値に依存しない.
- L2AP/MJ コラム:この方法はデータセットの全要素の索引付け時に b_1、$S(R)$ 側の入力レコードとインデックスに登録されている $R(S)$ 側のレコードとの照合時に rs_3 を用いた手法である.
- L2AP/MJ ノルム:この方法はデータセットの全要素の索引付け時に b_3、$S(R)$ 側の入力レコードとインデックスに登録されている $R(S)$ 側のレコードとの照合時に rs_4 を用いた L2 ノルムに基づいた手法である. 動作を R-S ジョイン対応させているだけなのでセルフジョイン時と戦略は同じである.
- L2AP/HJ:この方法はデータセット R の索引付け時に b_1_HJ、S 側の入力レコードとインデックスに登録されている R 側のレコードとの照合時に rs_3_HJ を用いた手法である.
- L2AP/HJ 水平分割:この方法は L2AP/HJ をデータセット R の水平分割により、インデックスとの照合条件を強化した手法である.

6.5.2 DBLP/R-S

検証用のデータセットとして DBLP の 2014 年 1 月〜2015 年 12 月の 2 年分の論文 487703 件のデータを用意した. データセット R は 243851 件、データセット S は 243852 件とした. 論文タイトルと著者名を単語の出現頻度に基づいて重さ付けをした上で、ベクトル化し疎ベクトルとした.

このデータセットを以後 DBLP/R-S と呼ぶ.

表 4 に DBLP/R-S の詳細な情報を示す.DBLP/R-S の全次元数は 235780、1 レコード辺りの平均ビット長は 16 ビットとなっている.

表 5 に DBLP/R-S における閾値を満たす類似ペア数を示す. 閾値 0.1 は類似ペア数が多すぎる点や、各戦略においてナイーブと同等の数字を示したので除外した.

6.5.3 DBLP/R-S における実験

DBLP/R-S を用いてナイーブ、L2AP/MJ コラム、L2AP/MJ ノルム、L2AP/HJ、L2AP/HJ 水平分割で実験を行った.

図 4 は戦略毎の処理時間を示した図となっている. どの場合においても L2AP/HJ 水平分割が最も処理時間が短い結果となっている. 特に閾値が低い場合、従来の L2AP/MJ と比較し半分程度の処理時間を示した. セルフジョインでの水平分割方式の場合と異なり、処理時間に大きな差が現れる結果となったので、要因を考えていく事とする.

図 5 は DBLP/R-S の戦略毎のインデックスエントリ数を示した図となっている.

L2AP/HJ では R 側のレコードのみをインデックスに登録している為、L2AP/MJ 方式と比較しエントリ数が少なくなっている事が分かる. このエントリ数の差が表 6 の候補生成時の Forloop 回数の差に表れていると考えられる. L2AP/HJ 方式でのエントリ数の削減とそれに伴うインデックス照合オーバヘッドの削減が確認できる結果となった.

次に候補生成部の振る舞いを検証する為、候補生成部の A[y] 更新回数を計測した. その図が図 6 である. 閾値 0.6 以上では L2AP/MJ ノルムが最も回数が少なく、0.5 以下では L2AP/HJ 水平分割が最も回数が少ない結果となっている. 処理時間とは異なる傾向が表れているが、オーバーヘッドの削減による結果であると考えられる.

また水平分割の効果として L2AP/HJ における A[y] の更新回数を 4 割程度削減しており、あらためて水平分割による候補生成部のフィルタ強化を確認できた.

7. ま と め

本稿では、逐次実行向けの高速な類似結合である L2AP に注目し、L2AP の各戦略の特性を検証し、その上でデータベース演算としての強化の余地を模索した.

表 4　データセット

データセット	レコード数	次元数	概要	ビット長	R のレコード数	S のレコード数
DBLP/R-S	487703	235780	論文名+著者名	平均 16 ビット	243851	243852

表 5　DBLP/R-S において閾値 θ を満たす類似ペア数

データセット	閾値 0.2	閾値 0.3	閾値 0.4	閾値 0.5	閾値 0.6	閾値 0.7	閾値 0.8	閾値 0.9
DBLP/R-S	1284373	352944	156785	80256	44044	24964	15094	9900

表 6　DBLP/R-S において閾値 0.4 の場合の処理結果

戦略	処理時間 [s]	IC[s]	CG[s]	CV[s]	$A[y] \neq 0$ となるペア数	27 行目の回数	エントリ数	候補生成時 A[y] 更新回数	候補生成時 Forloop 回数
ナイーブ	193.57	0.82	167.41	25.34	5312068208	-	6975241	586411041	11844119852
L2AP/MJ コラム	22.46	0.67	20.37	1.42	34795272	359452	4040721	293577376	954022292
L2AP/MJ ノルム	22.27	0.67	20.75	0.85	34352001	33011232	4369141	255620139	1156926035
L2AP/HJ	12.38	0.22	9.05	3.11	35148078	425535	1978497	245414610	337435318
L2AP/HJ 分割数 4	10.73	0.35	7.77	2.61	34575471	425472	1978497	141592480	337435318

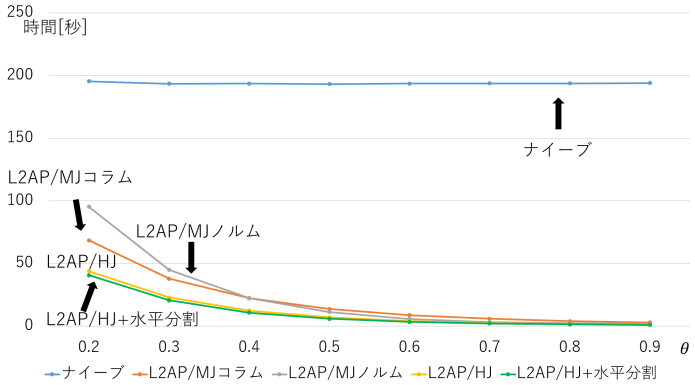

図 4　DBLP/R-S の戦略毎の処理時間

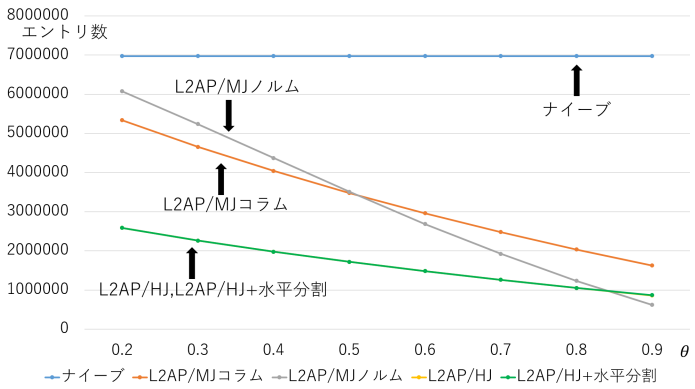

図 5　DBLP/R-S の戦略毎のインデックスエントリ数

具体的には、初めにセルフジョインにおける L2AP の動作を検証し、提案手法である水平分割方式によって強化を図った．しかし結果としてインデックス登録量と候補生成部の計算量の削減に見合う、処理時間の削減を確認出来なかった為、L2AP の動作特性を検証した．その際に L2AP の動作特性に着目し、その特性に伴うオーバヘッドの問題の解決を検討した．

そして R-S ジョインで L2AP を動作させる際の、L2AP の R-S ジョイン対応である L2AP/HJ を考え、セルフジョインで提案した水平分割も加え動作を確認した．結果として閾値が低い場合、従来の L2AP と比較しインデックス登録量と候補生成部の計算量を半分程度に削減し、その結果として処理時間も半分程度に削減する事が出来た．

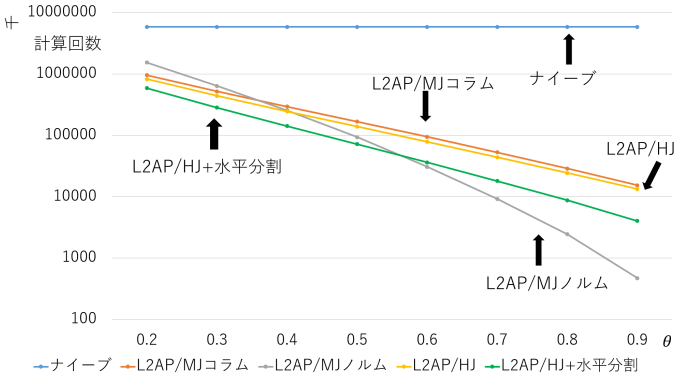

図 6　DBLP/R-S の戦略毎の候補生成時 A[y] 更新回数

文　献

[1] C.Anastasiu,G.Karypis. ”L2AP:Fast Cosine Similarity Search With Prefix L-2 Norm Bounds”, IEEE Int.Conf.Data Engineering, pp.784-795,2014.

[2] J.Bayardo,Y.Ma,R.Srikant. ”Scaling Up All Pairs Similarity Search”,in Proceedings of the 16th International Conference on World Wide Web,WWW’07, pp.131-140,2007.

[3] S.Chauduri,V.Ganti,R.Kaushik. ”A Primitive Operator for Similarity Joins in Data Cleaning”, IEEE Int.Conf.Data Engineering, pp.5-16,2006.

[4] P.Indyk and R. Motwani, “Approximate nearest neighbors:towards removing the curse of dimensionality,” in Proceedings of the thirtieth annual ACM symposium on Theory of computing, ser. STOC ’98, pp.604-613,1998.

[5] 廣瀬繁雄, 大森匡, 新谷隆彦. ”Map/Reduce におけるバケット再グループ化を用いたハイブリッドハッシュ結合アルゴリズム”.DBSJ Journal,vol.12,(No.1),pp.61-66,2013.

[6] A.Pavlo,E.Paulson,A.Rasin,D.J.Abadi,D.J.Dewitt,S.Madden, and M.Stonebraker.”A comparison of approaches to large-scale data analysis”,SIGMOD,pp.165-178,2009.