DBSJ Japanese Journal
Vol. 17-J, Article No. 10, March 2019

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
Utilizing Block Partitioning and Hash-Join in L2AP Similarity-Join Method

吉村 龍之介♡ 大森 匡♢ 新谷 隆彦♠ 藤田 秀之♣

Ryunosuke YOSHIMURA Tadashi OHMORI Takahiko SHINTANI Hideyuki Fujita

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法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ことを目指す. そして， 特に **R-S** ジョインの場合にこの方針が有効なことを示し，その具体的算法 **L2AP/HJ** を提案する.

For big data analysis, a *similarity join* is known as a basic but cost-expensive operation over a large database. For efficient processing of the operation, C.Anastasiu et al. proposed *L2AP* in 2014 along an approach of *AllPairs* of R.Bayardo. This paper examines effects of pruning strategies of L2AP, and utilizes block-partitioning and hash-join techniques to accelerate L2AP especially in case of R-S similarity join.

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. その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の算法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により，データベース処理演算としての L2AP の能力を強化する.

L2AP 自体はコサイン類似度下の類似結合を行う算法であり，データベース上のセルフジョイン (自己結合) をコサイン類似度下で行っているといえる. 本論文では，まず，L2AP のコラム戦略，ノルム戦略と呼ぶ効率化戦略について，その各々の加速能力を調べる. そして，データ集合 D を部分集合 $D_1, D_2, ..., D_s$ に水平分割してコラム戦略を強化する修正を導入し，セルフジョインにおける計算対象候補のフィルタリング能力の強化を試みる.

♡ 学生会員 電気通信大学
yoshimura@hol.is.uec.ac.jp
♢ 正会員 電気通信大学
omori@is.uec.ac.jp
♠ 正会員 電気通信大学
shintani@is.uec.ac.jp
♣ 正会員 電気通信大学
fujita@is.uec.ac.jp

次に，本論文では，上記の結果に基づいて，R-S ジョイン形式の類似結合を行う場合に L2AP を適用する事を考える. そして，ハッシュ結合に基づいて R 側だけをインデックス化するように L2AP を変形し，コラム戦略の新たな強化策を導入して，そのフィルタリング能力を大幅に強化した改良方法 L2AP/HJ を提案する.

以下，2. で類似結合と L2AP の概要を述べ，3. で L2AP の各戦略を原論文 [1] に沿って解説した後，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 の強化改良手法として L2AP/HJ を提案・評価し，最後に 7. でまとめる.

2. 類似結合と L2AP

2.1 導入

次元数 m の実数ベクトル n 個からなる集合 D と，D のベクトル $\boldsymbol{x}, \boldsymbol{y}$ に関する類似度関数 $sim(\boldsymbol{x}, \boldsymbol{y})$，類似度閾値 t が与えられたとする. D 上の類似結合とは，$\boldsymbol{x}, \boldsymbol{y} \in D$，$sim(\boldsymbol{x}, \boldsymbol{y}) \geq t$ となるような全ての $(\boldsymbol{x}, \boldsymbol{y})$ とその類似度 $sim(\boldsymbol{x}, \boldsymbol{y})$ を求める問題である. 本論文では解法として L2AP[1] を扱い，類似度関数をコサイン類似度に限定する.

以後，記号として $\boldsymbol{x}$ の第 j 次元を x_j と書き，$\boldsymbol{x} = \langle x_1, x_2, ..., x_m \rangle$ と記す. $\boldsymbol{x}$ の L2 ノルム $\|\boldsymbol{x}\|$ を用いると，類似度 $cos(\boldsymbol{x}, \boldsymbol{y})$ は $cos(\boldsymbol{x}, \boldsymbol{y}) = \frac{\boldsymbol{x} \cdot \boldsymbol{y}}{\|\boldsymbol{x}\| \times \|\boldsymbol{y}\|}$ である. コサイン類似度に限ったため，一般性を失わずに以後，$\boldsymbol{x}$ を，その L2 ノルム $\|\boldsymbol{x}\| = 1$ に正規化して扱う. 即ち，$\boldsymbol{x}$ の第 j 要素を x_j と表記すると $cos(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x} \cdot \boldsymbol{y} = \sum_{j=1}^{m} x_j \times y_j$ である.

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. ($\boldsymbol{x}$ のサイズと呼ぶ.)

$\|\boldsymbol{x}\| = \sqrt{\sum_{j=1}^{m} x_j^2}$ 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 $(\boldsymbol{x}, x_j)$ を記録する逆引きインデックス. $\boldsymbol{x}$ は m 次元なので m 個のインデックス $I_1, I_2, ..., I_m$ が作られ，その順番 $j = 1, \ldots, m$ は I_j の持つ要素数の降順とする.

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

L2AP は，インデックス入れ子ループ結合に沿った類似結合の高速化算法である AllPairs の改良版である [1]. ここでは L2AP の全体の処理構造を述べ，高速化のためのフィルタリング戦略は 3. で述べる.

前提として，データベースの各レコードは m 次元ベクトルで表されており，その各次元は出現頻度に基づく IDF 値で重み付けされ，そこから L2 ノルム $\|\boldsymbol{x}\|$ による正規化を行なって 2.1 で述べたベクトル集合 D を作成した，と仮定する.

まず，D 上の閾値 t についての L2AP の全体の処理 L2AP(D, t) を Algorithm1 に示し，Algorithm1 から呼ばれる候補生成・照合部を構成する関数 Find_MatchesL2AP を Algorithm2 に示した. (共に [1] の記載と同じもの).

Algorithm1 では，前処理としてその 1 行目で D の全 $\boldsymbol{x}$ を rw_x の降順でソートし，2 行目で次元 $1, \ldots, m$ の順序を頻度の降順に直して，インデックス $I_j (j = 1 \ldots m)$ の順を固定する. (IDF 値の高い次元ほど I_m 側になる.) そして，4 行目から，$\boldsymbol{x}$ のソート順に沿って D の先頭から順に各 $\boldsymbol{x}$ について処理を行う (Algorithm1

の 4~18 行目の for ループ). Algorithm1 の 4-18 行目の for ループ内の処理は以下の通り：

まず，5 行目で，$\boldsymbol{x}$ を Find_MatchesL2AP 関数に送る．この関数は，この時点で既に $I_1 \sim I_m$ に索引付けされているベクトルのうち，$\boldsymbol{x}$ に対して $cos(\boldsymbol{x},\boldsymbol{y}) \geq t$ を満たすベクトル $\boldsymbol{y}$ を全て求める関数である．

この後の Algorithm1 の 8~17 行目は Index Construction（インデックス作成部）と呼ばれる．ここでは，$\boldsymbol{x}$ 以降に追加されるベクトルと $\boldsymbol{x}$ との間での正解を落とさないように，$\boldsymbol{x}$ を $I_1 \sim I_m$ に適切に索引付けを行う．このとき，$\boldsymbol{x}$ の要素のうち，$I_1 \sim I_m$ に索引付けされた要素からなるベクトルが $\boldsymbol{x}''_j$，索引付けされなかった部分が $\boldsymbol{x}'_j$ である．

AllPairs や L2AP は，rw_x で降順ソートした D の先頭から順にインデックスに $\boldsymbol{x}$ を索引づけするのと同時に類似結合を行う形をとっており，その中で $\boldsymbol{x}'_j$ を索引づけしなくても正解漏れ無く類似結合を行えるように効率化を行う．そのため，インデックスへの登録データ量をできるだけ減らすことと，Find_MatchesL2AP 関数内での照合処理 (候補生成部と候補検証部の 2 つから成る) をできるだけ効率良く行うこと，を目的に様々な技法を用いる．

Algorithm 1 L2AP(D, t)

```
 1: D の全 x を rw_x の降順でソート    //データ生成時
 2: D の次元 1,...,m を頻度の降順にする    //データ生成時
 3: O ← ∅; I_1, I_2, ..., I_m ← ∅; cŵ_1, cŵ_2, ..., cŵ_m ← ∅
 4: for each x ∈ D from 1 to n do
 5:     O ← O∪ Find_MatchesL2AP(x, I_1, ..., I_m, ps, cŵ, t)
 6:     b_1 ← 0; b_t ← 0; b_3 ← 0
 7:     //Index Construction
 8:     for each j = 1 to m s.t.x_j > 0 do
 9:         pscore ← min(b_1, b_3)
10:         b_1 ← b_1+x_j × min(cw_j, rw_x)  //b_1(k) の計算
11:         b_t ← b_t + x_j^2; b_3 ← √b_t  //b_3(k) の計算
12:         if min(b_1, b_3) ≥ t then
13:             ps[x] ← pscore if ps[x] = 0
14:             I_j ← I_j ∪ {(x, x_j, ||x'_j||)}
15:             x_j ← 0
16:         end if
17:     end for
18: end for
19: return O
```

3. L2AP における効率化技法

3.1 Index Construction (インデックス作成部)

ここでは L2AP で提案された効率化技法を，コラム戦略 (I_j の統計情報に基づくもの) とノルム戦略（$\boldsymbol{x}$ の L2 ノルムに基づくもの）に大別し，本論文に必要な範囲で説明する．

まず，Algorithm1 のインデックス作成部におけるコラム戦略による索引付け条件 (b_1) を説明する．

$\boldsymbol{x}$ をインデックスに索引付けするとき，D のソート順で $\boldsymbol{x}$ より後にくる $\boldsymbol{y}$ との照合を考える．このとき，類似度閾値 t を満たす組 $(\boldsymbol{x},\boldsymbol{y})$ の内積は $y_j \leq cw_j$ かつ $y_j \leq rw_x$ であるから，
$\boldsymbol{x}\cdot\boldsymbol{y} \leq \sum_{j=1}^{m} x_j \times min(rw_x, cw_j)$ が成立する．

一方，ソート順で $\boldsymbol{x}$ の後にくる $\boldsymbol{y}$ 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 $\boldsymbol{x}\cdot\boldsymbol{y} = \boldsymbol{x}'_{\boldsymbol{k}}\cdot\boldsymbol{y}'_{\boldsymbol{k}}$ であり，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．以上から，$B_1 = argmin_K(\sum_{j=1}^{K} x_j \times min(cw_j, rw_x) \geq t)$ として，$\boldsymbol{x}$ の要素 $x_{B_1} \sim x_m$ だけを索引付けしておけば，$cos(\boldsymbol{x},\boldsymbol{y}) \geq t$ となる $\boldsymbol{y}$ から見て必要な $\boldsymbol{x}$ の情報が少なくとも 1 次元はインデックスに登録されていることが保証できる．実装方法としては，$b_1(k) = \sum_{j=1}^{k} x_j \times min(rw_x, cw_j) \geq t$ となる最初の k から $\boldsymbol{x}$ の要素を索引付けする．(Algorithm1 の 10,12,14 行目).

次にノルム戦略の索引付け条件 (b_3) を説明する．

$\boldsymbol{x}\cdot\boldsymbol{y} = \boldsymbol{x}'_j\cdot\boldsymbol{y} + \boldsymbol{x}''_j\cdot\boldsymbol{y}$ なので，もし $\boldsymbol{x}''_j\cdot\boldsymbol{y} = 0$ ならば $\boldsymbol{x}\cdot\boldsymbol{y} = \boldsymbol{x}'_j\cdot\boldsymbol{y}$

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

```
 1: A ← ∅; M ← ∅
 2: sz ← t/rw_x
 3: rs_3 ← Σ_{j=1}^{m} x_j × cŵ_j; rs_t ← 1; rs_4 ← 1
 4: rw'_{x_j} ← max(x'_j), Σ'_{x_j} ← Σ_{i=1}^{j-1} x_i, ∀x_j > 0
 5: //Candidate Generation
 6: for each j = m to 1 s.t. x_j > 0 do
 7:     I_j ← I_j\{(y, y_j, ||y'_j||)}, ∀y s.t. |y| × rw_y < sz//省略
 8:     for each ((y, y_j, ||y'_j||) ∈ I_j do
 9:         if A[y] >0 or min(rs_3, rs_4) ≥ t then
10:             A[y] ← A[y] + x_j · y_j
11:             if A[y] + ||x'_j|| × ||y'_j|| < t then
12:                 A[y] ← 0
13:             end if
14:         end if
15:     end for
16:     rs_3 ← rs_3 − x_j × cŵ_j
17:     rs_t ← rs_t − x_j^2; rs_4 ← √rs_t
18: end for
19: //Candidate Verification
20: for each y s.t.A[y] > 0 do
21:     next y if A[y] + ps[y] < t
22:     next y if A[y] + min(rw_x × Σ'_y, rw'_y × Σ_x) < t
23:     find first p s.t. y_p ∈ y' ∧ y_p > 0 ∧ x_p > 0
24:     s ← A[y] + x_p × y_p
25:     next y if s + min(rw'_{x_p} × Σ'_{y_p}, rw'_y × Σ'_{x_p}) < t
26:     for each j < p s.t. y_j > 0 ∧ x_j > 0 do
27:         s ← s + x_j × y_j
28:         if s + ||x'_j|| × ||y'_j|| < t then
29:             next y
30:         end if
31:     end for
32:     if s ≥ t then
33:         M ← M ∪ {(x, y, s)}
34:     end if
35: end for
36: cŵ_j ← max(x_j, cŵ_j), ∀x_j > 0
37: return M
```

であり，さらに，シュワルツの不等式より

$\boldsymbol{x}\cdot\boldsymbol{y} = \boldsymbol{x}'_j\cdot\boldsymbol{y} \leq \|\boldsymbol{x}'_j\| \times \|\boldsymbol{y}\|$ となる; ここで入力ベクトルが単位ベクトルであるため，結局，

$\boldsymbol{x}\cdot\boldsymbol{y} = \boldsymbol{x}'_j\cdot\boldsymbol{y} \leq \|\boldsymbol{x}'_j\| \times \|\boldsymbol{y}\| = \|\boldsymbol{x}'_j\|$ となる．

ゆえに $B_3 = argmin_K(\sqrt{\sum_{j=1}^{K} x_j^2} \geq t)$ とした時，$\boldsymbol{x}$ の $x_{B_3} \sim x_m$ だけを I に索引付けする．(もし $\boldsymbol{x}''_{\boldsymbol{B_3}}$ と $\boldsymbol{y}$ の内積が 0 なら，$\|\boldsymbol{x}'_{\boldsymbol{B_3}}\| < t$ により，$\boldsymbol{y}$ にとって $\boldsymbol{x}$ は不要.) 実装方法としては，$b_3(k) = \sqrt{\sum_{j=1}^{k} x_j^2} \geq t$ となる最初の k から索引付けする．

まとめると Algorithm1 では，ベクトル $\boldsymbol{x}$ について，$b_1 \geq t$ と $b_3 \geq t$ を共に満たす要素以降のみを索引付けする．

3.2 Candidate Generation (候補生成部)

Find_MatchesL2AP($\boldsymbol{x}, I, t$) は，D のソート順で引数 $\boldsymbol{x}$ よりも前にいる $\boldsymbol{y}$ のうち，$cos(\boldsymbol{x},\boldsymbol{y}) \geq t$ となる $\boldsymbol{y}$ を正しく求める．そのため，そのような $\boldsymbol{y}$ の候補を正解漏れなく探す「候補生成」(Algorithm2 の 6-18 行目) を行った後，その候補の検証 (candidate verification, Algorithm2 の 20-35 行目) を行う．候補生成部は 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 フィルタのみ説明する．

3.2.1 remscore フィルタ

まずコラム戦略による remscore フィルタ 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 へのループ (Algorithm2 の 6-18 行目) 内の各 I_j の全登録エントリとの照合 (同 8-15 行目) の効率化に利用する．具体的には，I_j の登録データを使った照合が終わる度に $x_j \times c\hat{w}_j$ を $rs_3(m)$ から繰り返し減算する（同 16 行目)．また，同 10 行目の $A[y] \leftarrow A[y] + x_j \cdot y_j (j = m, ..., j_0)$ を使って，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 $A[y] = \sum_{j=j_0}^{m} x_j \cdot y_j$.

ここで，$j = m, ..., j_0$ までのループ (Algorithm2 の 6-18 行目) が回った時点で，$rs_3(j_0)$ は，まだ処理が行われていない $I_1 \sim I_{j_0-1}$ にある任意の y と入力 x との間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ような候補ベクトル $\boldsymbol{y}$ は類似度閾値を満たさない．よってこの時点以後の未知の新しい候補 $\boldsymbol{y}$ の生成とその $A[y]$ の計算は省略される．

次にノルム戦略による remscore rs_4 を説明する．上と同じく，Algorithm2 の 6-18 行目のループ処理が I_{j_0} についてまで回ったとき，まだ処理が行われていない $I_1 \sim I_{j_0-1}$ で $\boldsymbol{x}'_{\boldsymbol{j_0}}$ が取り得る L2 ノルム $\|\boldsymbol{x}'_{\boldsymbol{j_0}}\|$ を用いる．

$\boldsymbol{x} \cdot \boldsymbol{y} = \boldsymbol{x}' \cdot \boldsymbol{y} + \boldsymbol{x}'' \cdot \boldsymbol{y} = \boldsymbol{x}'_{\boldsymbol{j_0}} \cdot \boldsymbol{y} + \boldsymbol{x}''_{\boldsymbol{j_0}} \cdot \boldsymbol{y}$ であり，$I_m \sim I_{j_0}$ のループが終了した時点でもし $A[y] = 0$ なら，$\boldsymbol{x}''_{\boldsymbol{j_0}} \cdot \boldsymbol{y} = 0$ であるから，この時点以後のループで見つかる新しいベクトル $\boldsymbol{y}$ については $\boldsymbol{x} \cdot \boldsymbol{y} = \boldsymbol{x}'_{\boldsymbol{j_0}} \cdot \boldsymbol{y} \leq \|\boldsymbol{x}'_{\boldsymbol{j_0}}\| \times \|\boldsymbol{y}\| = \|\boldsymbol{x}'_{\boldsymbol{j_0}}\|$ が成り立つ．従って，候補ベクトル $\boldsymbol{y}$ の生成条件のスコア rs_4 として，引数 $\boldsymbol{x}$ に対し $rs_4(k) = \sqrt{1 - \sum_{j=k}^{m} x_j^2} (k = m, ..., 1)$ を用意し，$rs_4 \geq t$ となる k の範囲でのみ新しい候補 $\boldsymbol{y}$ の生成を行えば正解を落とすことはない．

L2AP の候補生成部は，以上 2 つの条件により，引数 $\boldsymbol{x}$ に対する候補ベクトル $\boldsymbol{y}$ を生成する．

3.2.2 Candidate Verification の前倒し

L2AP は，候補生成部の中の 11-12 行目で，候補検証のうち L2 ノルムを使ったフィルタの一部を前倒しして行って効率化している．具体的には，Algorithm2 の 6-18 行目のループにおいて $I_m \sim I_{j_0}$ までの範囲で計算した内積 $\boldsymbol{x} \cdot \boldsymbol{y}$ の値を $A[y] = \sum_{j=j_0}^{m} x_j \cdot y_j$ と表した時，$\boldsymbol{x} \cdot \boldsymbol{y} = \boldsymbol{x}'_{\boldsymbol{j_0}} \cdot \boldsymbol{y}'_{\boldsymbol{j_0}} + A[y]$ であり，$\boldsymbol{x}'_{\boldsymbol{j_0}} \cdot \boldsymbol{y}'_{\boldsymbol{j_0}} \leq \|\boldsymbol{x}'_{\boldsymbol{j_0}}\| \times \|\boldsymbol{y}'_{\boldsymbol{j_0}}\|$ であるから，$A[y] + \|\boldsymbol{x}'_{\boldsymbol{j_0}}\| \times \|\boldsymbol{y}'_{\boldsymbol{j_0}}\| \geq t$ を満たさない $\boldsymbol{y}$ であれば，そのような $\boldsymbol{y}$ を ($\boldsymbol{x} \cdot \boldsymbol{y} \geq t$ となる $\boldsymbol{y}$ の) 候補から除外できる．同 11-12 行目は，この $\boldsymbol{y}$ の除外を，$A[y] = 0$ にリセットすることで行なう．

3.3 Candidate Verification (候補検証)

Algorithm2 の 20-35 行目では，入力 $\boldsymbol{x}$ を受けて候補生成部で求めた候補 $\boldsymbol{y}$ について，$cos(\boldsymbol{x}, \boldsymbol{y})$ を求めて，真に $cos(\boldsymbol{x}, \boldsymbol{y}) \geq t$ であるかを検証する (candidate verification, 候補検証部)．様々なフィルタ条件で候補 $\boldsymbol{y}$ の絞り込みを行うが，本論文では変更なしでそのまま用いるため説明を省略する．

4. 水平分割の提案

4.1 実装について

L2AP の改良を論じるにあたり，まず原論文 [1] のアルゴリズムを独自に C 言語で実装した．ただし，Algorithm2 の 7 行目のフィルタ箇所は，計算削減効果よりもオーバヘッドが処理時間に影響したため，意図的に外した．以後の実装評価では，実装は全て C 言語，動作環境は CentOS7.2, CPU 2.4GHz, メモリ 16GB の PC である．データには疎ベクトル表現を用い，一部高速化のため密ベクトル処理を使うが，これらも独自作成である．

4.2 水平分割

L2AP 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．最初にこの状況下での L2AP の高速化を考える．

最も直接的な方法は，データセット D を複数ブロックに水平分割して，ブロックごとにコラム戦略のフィルタ機能を強化することである．すなわち，領域分割数を s として，データセット D のレコード集合を，そのレコードのソート順に，s 個の集合 $D_1, D_2, ..., D_p, ..., D_s$ に水平分割する．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戦略のフィルタ条件 b_1 を強化できる．すなわち，コラム戦略では，$b_1(k) = \sum_{j=1}^{k} x_j \times min(rw_x, cw_j) \geq t$ 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，入力レコード以降の領域の cw_j だけを考えれば十分である．

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，$b_1(k) = \sum_{j=1}^{k} x_j \times min(rw_x, max(cwp_j, ..., cws_j)) \geq t$ となる最初の k から $\boldsymbol{x}$ の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

各領域毎の cw_j を用いることで，Candidate Generation のコラム戦略の remscore rs_3 を強化できる．具体的には，コラム戦略では，$rs_3(k) = \sum_{j=1}^{k} x_j \times c\hat{w}_j$ 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．しかし，領域 D_p に属する候補ベクトル $\boldsymbol{y}$ と Algorithm2 の入力 $\boldsymbol{x}$ の内積の上界値を考えるのであれば，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，候補ベクトルが属す領域を 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 D_p ごとに用意し，$x_j \times cwp_j$ を繰り返し $rs_3(k, p)$ から減算して，この $rs_3(k, p)$ を D_p に関する $\boldsymbol{x}$ との内積の上界値として用いれば良い．

5. セルフジョインの実験

5.1 概要

DBLP のデータを用いて L2AP のコラム戦略，ノルム戦略の特性効果と水平分割の効果を試す．原典の L2AP はコラム戦略とノルム戦略を併用するため，その性能は両方の戦略の高速な側の性能で動作することになる．ここでは，各戦略の特性を検証するため，各々を単独で用いる．また，水平分割は本質的にコラム戦略の強化版であるため，コラム戦略だけを用いた L2AP に適用する．

以下に，各戦略に基づいた方式を示す．いずれも，Algorithm1 と Algorithm2 の枠組みに従うが，効率化のためのフィルタリング技法だけが異なる：

- ナイーブ: Algorithm1 と Algorithm2 の枠組みに沿うが 3. のフィルタリング機能を一切用いない方式．すなわち，インデックス作成部ではデータセット D の各ベクトル $\boldsymbol{x}$ についてその全要素をインデックスに登録する；また，候補生成部では，$\boldsymbol{x}$ の全ての非 0 要素のインデックスの全エントリと内積計算をし，$\boldsymbol{x} \cdot \boldsymbol{y} \geq t$ となる $\boldsymbol{y}$ を決定する．候補検証部は不要なため省く．処理性能は閾値 t には依存しない．
- コラム： Algorithm1, 2 に従うが，コラム戦略に基づくフィルタ b_1 と rs_3 だけを使う方式．(b_3 と rs_4 を用いない)．すなわち，索引生成時に b_1 のみ，候補生成部のインデックスとの照合時に rs_3 のみを用いる．
- ノルム: Algorithm1, 2 に従うが，ノルム戦略のフィルタ b_3 と rs_4 のみを使う方式．(b_1 と rs_3 は用いない)．すなわち，索引付け時に b_3 のみ，インデックスとの照合時に rs_4 のみを用いる．
- 水平分割： 上述したコラム方式 (b_1 と rs_3 だけを使う方式) において，さらにデータセット D の水平分割を適用した方法である．

なお，コラム，ノルム，水平分割の各方式では，Algorithm2 の 11-12 行目の前倒し検証，及び，20-35 行目の候補検証は変更なしで使っている．

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 年分の論文データ 240595 件を用いる．論文タイトルと著者名を単

表 1: 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

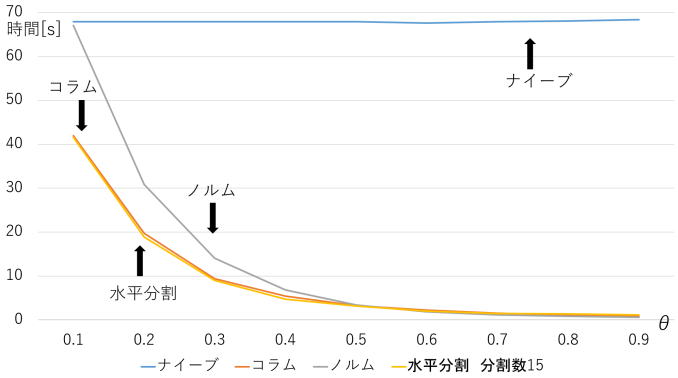

図 1: DBLP2 上の各方式の処理時間 (秒)

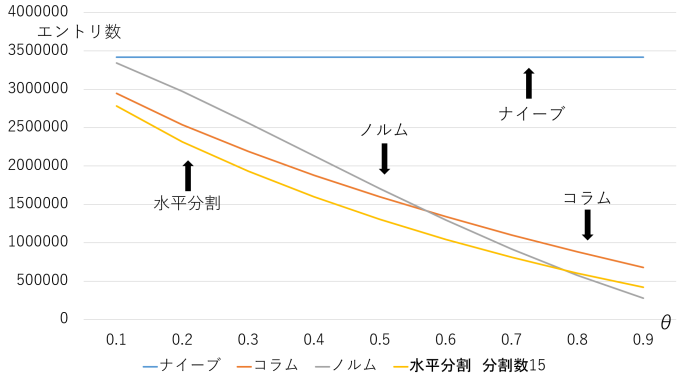

図 2: DBLP2 上の各方式のインデックスエントリ数

語トークン化して 1 トークンを 1 次元とし，単語の出現頻度（レコード数）による IDF 値で重さ付けして各レコードを疎ベクトル表現にした．このデータセットを以後 DBLP2 と呼ぶ．DBLP2 の全次元数は 326732, うち 2 レコード以上に出現する次元の総数は 154977，1 レコードを表すベクトルが持つ出現頻度 2 回以上の次元の数は平均で 16．実行に際し，出現頻度 1 の次元 (つまり，1 レコードにしか出現しないトークン) については，インデックス作成・登録は行なわない．

表 1 に，DBLP2 において類似度の閾値 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ため，$\theta = 0.2$ 以上の状況で動作特性を調べる．

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 (分割数 15) の各方式で実験を行った．D の rw_x によるソートと次元順の決定は前処理で行っており，以下の処理時間には含まない．

図 1 に，類似度の閾値 θ における各方式の処理時間を示す．ナイーブとコラム，ノルムの各方式は概ね原論文 [1] の記載とほぼ同様の振舞いを示した．一方，水平分割方式は，強化元のコラム方式より処理時間では若干良好な結果を示しており，全体では閾値が高い場合はノルム方式が最速，閾値が 0.5 以下では水平分割方式が最速となった．水平分割方式は処理時間でコラム方式より少ししか改善されなかったため，その原因を調べる．

図 2 に，DBLP2 上の各方式のインデックスの登録エントリ総数 (以下，エントリ数) を示す．処理時間同様に，閾値が高い場合，ノルム方式のエントリ数が最も少なく，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．水平分割方式が意図したインデックスのエントリ数は，$\theta = 0.3$ でコラム方式の場合から約 12%減であり，効果は確認できる．

次に，候補生成部の内積計算 $A[y]$ の更新回数 (Algorithm2 の 10 行目の実行回数) の総数を，各方式について図 3 に示す．インデックスエントリ数の場合と同じく，閾値が高い場合はノルム方式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くなった．(閾値 0.3 で水平分割のこのコストはコラム方式より 45%減．)

以上より，水平分割で狙ったインデックス量と内積計算コストの削減は機能している事が分かる．しかし，この削減はコラム方式からの計算処理時間の削減につながっていない．

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 $A[y]$ の更新回数を削減できた．一方で処理時間の削減量は少ない．以下，この原因を考える．

5.4.1 候補生成部のコスト

まず，候補生成部の計算コストについて考える．

図 3 に示したように，水平分割を行った結果，remscore フィルタが強化され，$A[y]$ の更新回数はコラム方式の場合よりある程度 (約 40%強) 削減されている．しかし，この削減はあくまで Algorithm2 の 10 行目の $A[y]$ の内積計算の削減でしかない．この削減を判定するために，Algorithm2 は，インデックス I_j の登録エントリの全スキャン (同 8-15 行目) を行っていることに注意する．すなわち，水平分割によって I_j のエントリ数自体がコラム方式の場合よりも若干減り，remscore フィルタによって I_j のスキャン終了時以後は新規の候補の生成が省かれても，その時点までに $A[y] > 0$ となった候補 $\boldsymbol{y}$ の $A[y]$ 計算は，残りの $I_{j-1}, \ldots, I_1$ のスキャンによって行う必要がある．このスキャン処理については，コラム方式でも水平分割でもインデックスエントリ数の程度しかコストは変わらない．つまり，L2AP は処理コストとして内積計算のコスト $A[y]$ を減らすことを重視しているが，上記の実験では，このインデックスの全スキャン処理の方が処理時間の支配項になったと考えられる．

5.4.2 候補検証部のコスト

次に，候補検証部のコストについて考える． 候補検証部ではベクトル $\boldsymbol{x}$ の索引付けしていない部分 $\boldsymbol{x}'$ を使ったフィルタ条件 (Algorithm2 の 21, 22 行目) を使って候補ベクトル $\boldsymbol{y}$ と $\boldsymbol{x}$ の内積計算コスト (同 27 行目) を削減する．

表 2 に，閾値 0.4 における各方式の候補生成部と候補検証部の処理コストを載せた．同表は，左から順に，全体の処理時間，インデックス作成部 (IC)，候補生成部 (CG)，候補検証部 (CV) の処理時間を示し，その次に Algorithm2 の 19 行目に残った検証すべき候補の組 (x, y) の総数，その次の 2 つが候補検証部の各フィルタを通過した候補ペア数である．(最後の欄がインデックスエントリ数．)

表 2 の「エントリ数」と「22 行目を通るペア数」に着目すると，水平分割方式ではインデックスエントリ数が最小で，この「ペア数」が最大になっている．一方，コラム方式やノルム方式は，インデックスエントリ数は多いが，「ペア数」はむしろ小さい．つまり，索引付けする要素数を減らすと，候補検証部で検査する候補ペア数が増える場合があるといえる．

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．6. では，5. の結果を踏ま

表 2: セルフジョインの各方式の処理コスト内訳 (閾値 0.4, DBLP2)

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

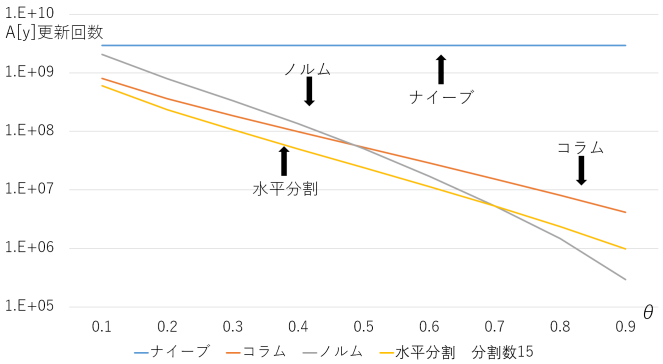

図 3: DBLP2 上の各方式の候補生成時 A[y] 更新回数

え，L2AP の持つ特性に着目して，R-S ジョイン形式の類似結合に適した L2AP の改良版として L2AP/HJ と呼ぶ方式を提案する．

ここで R-S ジョイン形式の類似結合とは，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，$\boldsymbol{x} \in R$，$\boldsymbol{y} \in S$，$sim(\boldsymbol{x},\boldsymbol{y}) \geq t$ 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． 以下，R-S ジョインと言う時は，この類似結合の意味で使い，類似度はコサイン類似度に限る．

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 Find_MatchesL2AP 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できないものとなる．これが 5. でわかった L2AP の特性である．よって，R-S ジョインの場合でも，インデックス登録量を大幅に減らす事で候補生成時のコストを減らす事ができる．

6.2.2 L2AP/MJ

L2AP で R-S ジョインを行う時，最も直接的な方法は，データセット $R \cup S = D$ として，D を rw_x で降順ソートして入力とし，セルフジョイン用の L2AP を D に適用することである．この時，インデックス作成部でのベクトルの登録は 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についてのみ該当する計算を行う．また各フィルタに用いる cw_j としては $cw_j = max\{x_j | \boldsymbol{x} \in R \cup S\}$, $c\hat{w}_j = max\{x_j | \boldsymbol{x} \in I_j\}$ として用いることになる．

上記の方式は，マージソートジョインの考えを直接 L2AP の場合に使ったものとも言える．以後，この方式を，L2AP/MJ と呼ぶ．

以下では，L2AP/MJ においても，コラム戦略のフィルタ b_1 と rs_3 だけを使う場合（L2AP/MJ コラム）と，ノルム戦略のフィルタ b_3, rs_4 だけを使う場合（L2AP/MJ ノルム）との 2 方式に分けて以後の評価に用いる：

L2AP/MJ コラム： データセットを $R \cup S = D$ として，D について R-S ジョインを L2AP の Algorithm1, 2 により実行する．このとき，セルフジョインの時と同じく，D 全体で決まる rw_x と cw_j を用いて索引付けフィルタ b_1 と候補生成フィルタ rs_3 を決めて用いる．(b_3 と rs_4 は用いない．) つまり，L2AP/MJ コラムで用いる $b_1(k)$ と $rs_3(k)$ は，$D = R \cup S$ 下のセルフジョイン時の L2AP のそれらと同じであり，内部動作も Algorithm1, 2 と同じ．ただし，Find_MatchesL2AP 関数内の候補生成は，相異なるリレーションに属すベクトルの組についてのみ行い，そうでない場合は内部処理をスキップするように修正する．

L2AP/MJ ノルム： L2AP/MJ でノルム戦略の各フィルタ b_3 と rs_4 のみを使い，b_1, rs_3 を使わない方式．このときの b_3 と rs_4 は，$D = R \cup S$ とおいた時のセルフジョイン用のノルム戦略の式定義のままで良い．なぜなら，ノルム戦略の場合，入力レコードと候補レコードの 2 つだけでフィルタが決まり，データセット $D = R \cup S$ 全体の情報には依存しないからである．候補生成部の動作は，L2AP/MJ コラムの場合と同様に，相異なるリレーションのベクトル間の照合のみを行うように修正しておく．

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 の枠組みで R-S ジョインを行う時，L2AP の特性から見てインデックス登録量自体を大幅に減らす事を重視すべきとなると，ハッシュ結合 [5],[6] が有利になるはずである．すなわち，ハッシュ結合なら，R だけのインデックスを作成し，S のベクトルからはインデックス作成せずに照合をするだけである．先述した L2AP/MJ が，マージソート結合のために $D = R \cup S$ 全体のインデックスを作成したのに比べ，この方針なら，インデックス登録量が大幅に少なくなり，L2AP/MJ よりも処理時間を削減できるはずである．

以下，ハッシュ結合に基づいて R-S ジョインを行う L2AP の改良方式を L2AP/HJ と呼び，その具体化を行う．

前提として，データセット R と S は，各々を rw_x で降順ソートを行っておき，R の後に S を連結する形式で $D = R \cup S$ を作って L2AP/HJ の入力とする．次元の出現頻度による降順の決定は D に対して行う．

L2AP/HJ の全体の処理の流れとしては，D のうち，R の入力については L2AP のインデックス作成部（の修正版）のみを行い候補生成と候補検証は行わなず，その後，S の入力についてはインデックス作成を行わずに，候補生成部（の修正版）と候補検証を行う，としておく．

以下，ベクトル集合 R の cw_j を cwR_j、S の cw_j を cwS_j と表記し，cwR_j と cwS_j を用いて，上記の L2AP/HJ の処理の中で用いる索引付け条件と候補生成時の remscore フィルタを強化する．強化の対象となるのは，コラム戦略下のフィルタ b_1 と rs_3 である．

6.3.1 Index Construction フィルタリングの強化

上述した L2AP/HJ の処理の枠組みでは，$\boldsymbol{x} \in R$ をインデックスづけするとき，S 側の cwS_j を用いることで，$\boldsymbol{x} \in R$ に関するコラム戦略のフィルタリング条件 b_1 を強化できる．

具体的には，L2AP/MJ コラムの時は，基本的な考えは $b_1(k) = \sum_{j=1}^{k} x_j \times cw_j \geq t$ となる最初の k から $\boldsymbol{x}(\in R \cup S)$ の索引付けを行うことだったが，この cw_j は $D = R \cup S$ 全体から決める必要があった．これに対し，L2AP/HJ では，$\boldsymbol{x}(\in R)$ と S のレコードとの内積の最大値だけを考えて $\boldsymbol{x}$ を索引づけすれば良い．つまり，$b_1(k)$ 式では，S の cw_j だけを使えば十分である．

よって，L2AP/HJ の (R の) インデックス作成部では，cwS_j を用いて $b_1\_HJ(k) = \sum_{j=1}^{k} x_j \times cwS_j$ を用意し，$b_1\_HJ(k) \geq t$ となる最初の k から $\boldsymbol{x} \in R$ の要素を索引付けすれば良い．

表 3: 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

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 におけるコラム戦略の remscore フィルタ rs_3 を強化できる．

元のセルフジョイン用のコラム戦略では，$rs_3(k) = \sum_{j=1}^{k} x_j \times c\hat{w}_j$ 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，R-S ジョインの場合，Algorithm2 の候補生成部では，R の全ベクトルがインデックスに登録された状態で $\boldsymbol{x} \in S$ から候補生成を行う状況に限定されている；この内積の上界値を考えるのであるから，インデックスに登録済みの R の cw_j の値と，$\boldsymbol{x} \in S$ の要素 x_j の積を考えて $rs_3(k)$ を作れば十分である．

従って，L2AP/HJ で候補生成を行うときは，cwR_j を用いて，新しい remscore フィルタ $rs_3(k) = \sum_{j=1}^{k} x_j \times cwR_j$ を用意し，$rs_3(m)$ から $x_j \times cwR_j$ を繰り返し減算して内積の上界を求めればよい．以後，この $rs_3(k)$ を $rs_3\_\mathrm{HJ}(k)$ と表記する．

以上に加えて，セルフジョインの場合と同様に，R の水平分割を適用して rs_3_HJ をさらに強化できる．R-S ジョインの場合，領域分割数を s とし，インデックスに登録する R のデータ集合を $R_1, R_2, ..., R_p, ..., R_s$ に水平分割する．そして，候補ベクトルの属す領域を $R_p (p = 1, 2, ..., s)$，領域 R_p の cw_j を $cwRp_j$ としたとき，$\boldsymbol{x} \in S$ に対して $rs_3\_\mathrm{HJ}(k, p) = \sum_{j=1}^{k} x_j \times cwRp_j$ を用意し，候補生成部で $x_j \times cwRp_j$ を繰り返し減算して内積の上界とすればよい．

6.4 L2AP/HJ のアルゴリズムの全体概要

まとめると，L2AP/HJ のアルゴリズムの Algorithm1, 2 からの変更は次の通りである：

前提：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．

(1). Algorithm1 の修正は，入力レコードが R 側の場合，Find_MatchesL2AP 関数を呼び出さず，Index Construction（インデックス作成部）のみを行う．S 側のレコード入力なら，インデックス作成部をスキップし，Find_MatchesL2AP 関数への入力のみを行うように修正する．式 $b_1(k)$ の代わりに，$b_1\_\mathrm{HJ}(k)$ を用いる．

(2) Find_MatchesL2AP 関数は $\boldsymbol{x} \in S$ の入力についてしか呼ばれない．Algorithm2 の候補生成部では，$rs_3(k)$ ではなく，代わりに $rs_3\_\mathrm{HJ}(k)$ を使う．

L2AP/HJ は，R のみをインデックス登録するため L2AP/MJ と比較し全体のインデックス登録量を少なくでき，メモリ量だけでなく，L2AP の内部処理のインデックススキャンを削減できる．また，ハッシュ結合の手順に沿うことで L2AP の索引づけ条件と remscore フィルタを L2AP/MJ の場合よりも強化できており，処理時間自体を大幅に削減できるはずである．

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+水平分割の各方式で実験を行った．

これら 4 つのうち，L2AP/MJ コラム，L2AP/MJ ノルムは，6.2.2. で述べた方式である．それ以外の方式は，以下：

- ナイーブ方式: $D = R \cup S$ にセルフジョイン時のナイーブ方式を適用したもの．ただし，候補生成部では，相異なるリレーションに属すレコード間の内積しか計算対象としないよう修正する．性能は閾値に依存しない．
- L2AP/HJ: 6.3 と 6.4 で述べた方式．ただし，元のコラム戦略からの速度向上を調べるため，改良されたコラム戦略だけを用いる．すなわち，R のインデックス作成には b_1_HJ だけを使い，S 側のベクトルと R 側のベクトルの候補生成部では rs_3_HJ だけを用いる．原理上はノルム戦略のフィルタ b_3, rs_4 も併用できるが，今回は用いていない．
- L2AP/HJ+水平分割: 上記の L2AP/HJ に 6.3.2 で述べた水平分割を適用したもの．従って，コラム戦略のみに基づく．

なお，ナイーブ以外の方式は全て，Algorithm2 の 11-12 行目の前倒し検証，20-35 行目の候補検証を変更なしで用いている．

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，これを，R として 243851 件，S として 243852 件に分けて用いる．（分け方は会議名の文字列順などであり，年度や rw_x には無関係．）論文タイトルと著者名を単語の出現頻度に基づいて IDF で重さ付けをし，疎ベクトルとした．このデータセットを DBLP/R-S と呼ぶ．DBLP/R-S の全次元数は 472104，うち出現頻度 2 以上の次元数が 235780，インデックス作成対象となる 1 レコード辺りの平均要素数は 16 である．

表 3 に DBLP/R-S における閾値を満たす類似ペア数を示す．閾値 $\theta = 0.2$ 以上で評価を行う．

6.5.3 DBLP/R-S 上の実験結果

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 に，各方式の処理時間を閾値 $\theta = 0.2 - 0.9$ で示す．この範囲の閾値では，L2AP/HJ+水平分割が最も速く，L2AP/HJ と合わせて，最速となった．特に，閾値が 0.2-0.4 と低い時，L2AP/HJ と同水平分割方式は，L2AP の直接的適用として従来方式と呼べる L2AP/MJ コラムに比べ，半分程度の処理時間となった．方式によって処理時間に大きな差が現れる結果となったので，要因を調べた．

図 5 に，DBLP/R-S 上の閾値 0.2-0.9 における各方式のインデックスエントリ総数を示す．また，表 4 に，閾値 0.4 における各方式の処理時間内訳と候補生成部の $A[y]$ 更新回数，インデックス照合ループ (Algorithm2 の 8-15 行目) の回数，を示す．

図 5 から，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大幅に少なくなった事が分かる．このエントリ数の差が，表 4 の候補生成部のループ回数 (#forloop(CG) の欄) の差に表れたと言える．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 ノルムが最も少ないが，0.5 以下では L2AP/HJ と同水平分割方式の 2 つが最も少なくなった．水平分割の効果としては，L2AP/HJ における $A[y]$ の更新回数を 4 割程度削減しており，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. おわりに

本論文では，逐次実行向けの高速な類似結合演算の実行算法として 2014 に Anastasiu らが提案した L2AP に注目し，L2AP で使われているコラム戦略・ノルム戦略に基づくフィルタリング機能の効果と処理コストの特性を検証した．そして，その結果に基づき，データベース演算として類似結合を行うときの L2AP の強化を模索した．

表 4: R-S ジョインの各方式の処理コスト内訳 (閾値 0.4, DBLP/R-S)

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	#A[y] 更新回数 (CG)	#forloop(CG)
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

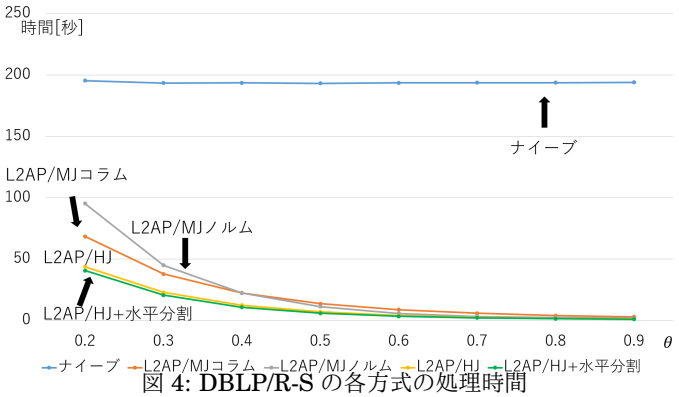

図 4: DBLP/R-S の各方式の処理時間

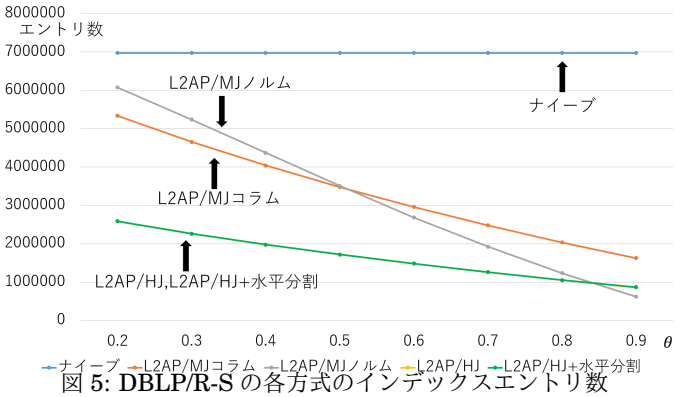

図 5: DBLP/R-S の各方式のインデックスエントリ数

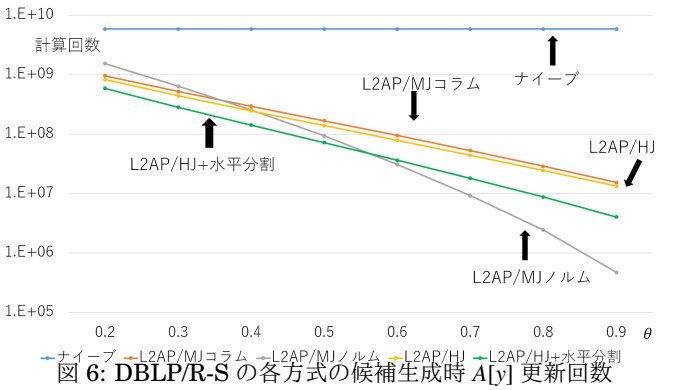

図 6: DBLP/R-S の各方式の候補生成時 $A[y]$ 更新回数

始めに，セルフジョインにおける L2AP の動作を検証し，データベース D を複数ブロックに水平分割することでコラム戦略のフィルタリング能力を強化する方法を提案した．しかし，DBLP 上の評価の結果，水平分割によって確かにコラム戦略下の効率化の狙いであるインデックス登録量，及び，候補生成部の内積計算量の削減を得たが，それに見合う処理時間の削減を得られなかった．そのため，L2AP の動作特性を調べ，候補生成部における内積計算コストの削減を判定するために全インデックス I_j のエントリのスキャンを行うこと，この処理のオーバヘッドが無視できないことに注目した．

次に，上記の性質に基づいて，本論文では，R-S ジョイン形式の類似結合を L2AP で行う場合を論じ，ハッシュ結合法に沿った L2AP の改良方式 L2AP/HJ を提案した．ハッシュ結合であれば，リレーション R 側のみのインデックス作成だけで済み，L2AP のネックであるインデックス登録量自体を減らすことができる．L2AP/HJ では，この考えに沿って，R 側のインデックス登録量とリレーション S 側のベクトルによる候補生成時の照合コストの両方を大幅に削減することを狙った．そして，コラム戦略に基づくインデックス作成部のフィルタと候補生成部の remscore フィルタの 2 つを，ハッシュ結合下の L2AP の動作向けに大幅に強化する方法を提案した．そして，水平分割による修正も含め，R-S ジョインで L2AP/HJ を評価し，従来方式に相当するマージソート結合型の L2AP/MJ と比較した．その結果，閾値が低い場合，従来の L2AP/MJ と比較して，L2AP/HJ は，インデックス登録量と候補生成部の内積計算コストを約 40-60%削減し，その結果，処理時間も半分程度に削減できた．

今後の課題としては，L2AP 内のインデックス構造の変更や，データベースの複数ブロック分割に伴う類似結合の並列化などがあげられる．

[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 Proc. 16th Int. Conf. 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 Proc. 13th ACM Symp. Theory Of Computing (STOC) ’98, pp.604-613,1998.

[5] 廣瀬, 大森, 新谷, ” 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”,ACM SIGMOD Conf.,pp.165-178,2009.

吉村龍之介 Ryunosuke YOSHIMURA
電気通信大学大学院情報理工学研究科修士課程修了 (2018)．現在，

日本アビオニクス（株）に勤務.

大森匡 **Tadashi OHMORI**
電気通信大学大学院情報理工学研究科教授. データ工学，データベースアルゴリズム，トランザクション処理等の研究に従事.

新谷隆彦 **Takahiko SHINTANI**
電気通信大学大学院情報理工学研究科准教授. データ工学，データマイニング，並列データ処理等の研究に従事.

藤田秀之 **Hideyuki FUJITA**
電気通信大学大学院情報理工学研究科准教授. データ工学，地図情報処理，空間データ等の研究に従事.