Vol.2009-MPS-76 No.47
Vol.2009-BIO-19 No.47
2009/12/18

超高次元データの l 近傍検索への試み

ユスフ　ムカルラマー[†1]　　渡辺　知恵美[†1]　　瀬々 潤[†1]

遺伝子や画像のピクセルなどの超高次元データの類似検索などは幅広い分野で応用できる．本研究は与えられた超高次元の点データに対して類似した上位 l 個のデータを検索する手法を提案する．本研究では多様体型データに焦点を当てる．多様体型データは近傍グラフで表すことによって，点間の本来の類似関係を保つことができる[5]．近傍グラフを用いたマイニング手法や最近傍発見手法は数多く存在するが，近傍グラフ上でクエリ点に類似した上位 l 個を求める手法はいままでなかった．そこで，本研究では，クエリ点に類似した上位 l 個のデータを k 近傍グラフ上で求めることを提案する．クエリ点の k 近傍を求め，そしてそれらの点から一時的に辺を張る．次に，クエリ点をスタート点として，グラフ上での隣接関係を用いて，類似した上位 l 個のものを検索する．

l Nearest Neighbors for a High-dimensional Data

Mukarramah Yusuf,[†1] Chiemi Watanabe[†1]
and Jun Sese[†1]

Similarity search for high-dimensional data such as gene expressions and image pixels has a vast application in many fields. In this paper, we propose a method for finding similar top l points for a query point, from a point database. In this research, we specially focus to manifold data. By using neighborhood graph to present the manifold data, we can save the intrinsic similarity between two arbitrary points[5]. There are many researches that using neighborhood graph as a method for mining and finding nearest neighbor, but finding nearest l neighbors on a neighborhood graph has not been researched yet. There, in searching top l points, we propose to use neighborhood graph. First, we search k nearest neigbors of the query point, then add a temporary edge from the query point to them. Last, we then find the top l nodes by using adjacency relationship between nodes.

1. はじめに

近年，遺伝子発現量の検索エンジンの要求が高まっている．例えば，医療では，遺伝子発現量の検索エンジンは医師の診断を助けることができる．具体的な例としては，例えばある薬に対する副作用が出やすい患者Aがいたとする．医師が別の患者Bを診療する際に，患者Bが患者Aと似た遺伝子情報を持っているということが分かれば，その薬に対する副作用が出やすいのではないかと予測できる．このように，医師が患者から採取した遺伝子発現量を入力すると，過去のどの患者の遺伝子情報に近いかを検索する遺伝子検索エンジンの実装が必要とされている．

また，遺伝子発現量の検索エンジンは医療だけではなく，例えばビール，ワイン，日本酒などの酒類や生分解性プラスチック分解菌など，生活や環境に役立つ菌類の種分けと改善にも役立つ．

遺伝子発現量の検索エンジンを開発するにあたり，いくつかの問題を考える必要がある．例えば医療では，検索対象となる患者の数は膨大になることが予想されるため，大規模なデータからの検索が求められている．更に，人の遺伝子は約3万あり，遺伝子の数だけ次元があるため，各患者の情報は超高次元データとなる．超高次元空間では次元の呪いにより，各データ間の距離がほぼ等しく見える現象が知られており，検索精度が低いと予測される．診断において検索精度が低いことは致命的となるので，この点も考慮する必要がある．

先行研究[8]では，これらの問題を考慮した遺伝子発現量の検索エンジンのプロトタイプがwebアプリケーションとして開発された．しかし，検索エンジンが，より実用的なものになるため，素早い検索機能が求められている．そこで，本研究は，このアプリケーションを高速化するための手法を考える．

今まで，高次元データの検索について，様々な研究が行われた．R-Treeなどの高次元データの検索を高速化するための索引は沢山あるが，データの次元が１０以上になると，点が同程度の距離にあると判断され，検索が失敗してしまう．つまり次元の呪いに陥ってしまう．

また，近年では，遺伝子のクラスタリングを行う際には，遺伝子の近接性に注目することが多い[2]．遺伝子データを線形多様体と見なし，その多様体を低次元の空間にマッピングし，クラスタリングを行っている．多様体のクラスタリングに関する研究が数多く存在する[3][4]．

†1 お茶の水女子大学
Ochanomizu University

Vol.2009-MPS-76 No.47
Vol.2009-BIO-19 No.47
2009/12/18

それらの研究は，多様体状のデータのクラスタリングを行うが，多様体上での遺伝子発現量検索も要求されることがあると考える．我々は，先行研究のベースとなる検索エンジンの高速化において，そのような要求にも応じ，超高次元データ及び多様体に対応できる索引を検討する．遺伝子発現量の検索エンジンの索引づけする際に，まず，遺伝子データが多様体型ではなく，超高次元データであるということのみを考慮する場合は，M-Tree を採用することとした．M-Tree は絶対的な距離ではなく，相対的な距離を用いた索引である．点データを超高次元空間にプロットするのではなく，点間の相対的な距離を求め，M-Tree を構築する．そうすることで，次元の呪いを避ける．また，発現量検索に使う関数は，用途に合う距離関数に設定できるので，M-Tree は適切であると考える．

そして，遺伝子発現量データが多様体と見なされる場合，k 近傍グラフを用いた手法を提案する．具体的に，まず，データの k 近傍グラフを用意しておく．そして，問い合わせがあった際，クエリ点の k 近傍を求め，それらの点へ一時的に辺を張り，クエリ点を k 近傍グラフに組み込む．次に，クエリ点をスタート点として，グラフ上での隣接関係を用いて，類似した上位 l 個のものを検索する．

本稿は以下のような構成となっている：第２節は超高次元データを扱う時の問題，第３節は関連研究，第４節は超高次元データに対する高速な検索の試み，第５節は提案手法の評価である。

2. 超高次元データを扱う時の問題

本節は，超高次元データを扱う時の問題について説明する．上で述べたように，遺伝子発現量データの索引を構築するときは，２つの問題を考えなければいけない．一つ目は，遺伝子発現量データは超高次元データである．遺伝子発現データとは，各遺伝子がどのような時期または組織・条件で発現しているかを表した数値データのことである．人の場合，遺伝子の数が 3 万あるので，それぞれの発現量を属性であると考える場合，ある一人のデータが 3 万個の属性を持つタプルとなる．超高次元データを扱うときに生じる問題として，類似性を一般的なユークリッド距離で表すと，次元の呪いに陥ってしまうということである．

２つ目の問題は，多様体型データを考える必要がある．遺伝子データを扱う時，常に細胞間の相関を見る．同時期に発現している遺伝子のグループは，同一の機能を持っている可能性が高いと知られている．同時期に発現しているかを測る指標を相関係数と呼ぶ．遺伝子間の相関関係を表すと，遺伝子データのネットワークは多様体になることがある．この多様体上での検索の方が有意義である．例えば，図 1 に示すような多様体型データの一種であるロールケーキデータでは２点 $p1$,$p2$ とクエリ点 q がユークリッド距離の場合は，$d(p2,q) < d(p1,q)$ という関係になるが，多様体上での距離の場合は，$d(p1,q) < d(p2,q)$ になる．多様体上での検索とは，この多様体上での距離で近さを表すということである．つまり，$p1$ の方が q に類似している．

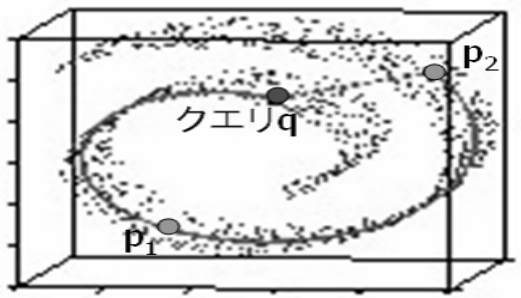

図 1 Tenenbaum のロールケーキデータ
Fig. 1 Tenenbaum's "Swiss Roll" data

3. 関 連 研 究

本節では関連する研究と提案手法で利用する技術を説明する．

3.1 M-Tree

M-Tree[1] は膨大な超高次元データを管理・アクセスするための索引である．ユークリッド空間やベクトル空間上では表現できないもののために提案された．いわゆる距離空間 (非負性，対称性，三角不等式を満たす空間) の索引である．M-tree の以前に，FastMap,VP-tree,MVP-tree,GNAT などの距離空間索引は提案されたが，全て静的な索引であるため，新しいデータの挿入や要らないデータの削除ができない．

M-tree の葉ノードはオブジェクトを，中間ノードはルーティングオブジェクトと呼ばれるものを保存する．ルーティングオブジェクトは部分木のポインターを保持する．部分木に含まれている全てのノードはルーティングオブジェクトのカバー半径内に位置する．

他の動的な索引と同じように，M-tree はボトムアップ型で成長していく．成長過程では，子ノードの数が予め決めた数となれば，ノードが定義された方針に従って分割する．エントリの挿入操作の他，削除操作も行える．また，M-tree で効率の良いレンジクエリや k 近傍クエリができる．

Vol.2009-MPS-76 No.47
Vol.2009-BIO-19 No.47
2009/12/18

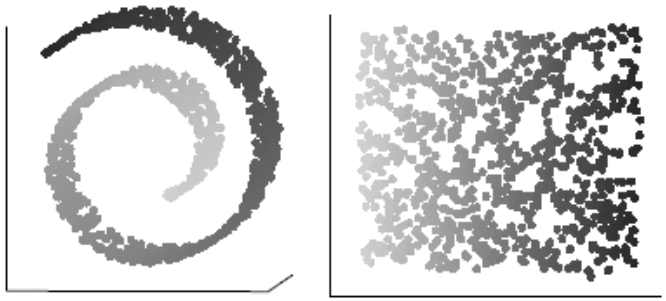

図 2 Tenenbaum のロールケーキデータとその２次元に圧縮した結果 ([6] から)
Fig. 2 Tenenbaum's "Swiss Roll" data and it's two dimensional embedding(from[6])

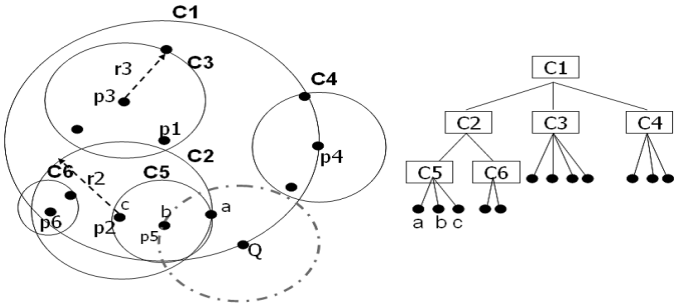

図 3 M-Tree の例
Fig. 3 An example of M-Tree

M-Tree の構築 M-Tree は挿入操作の繰り返しで構築される．挿入操作は木を下っていき，挿入する新しいオブジェクトのために最適な葉ノードを見つける．葉ノードがいっぱいになったら，分割を実行する．最適な葉ノードとは，基本的に，新しいオブジェクトを挿入することによって，部分木のカバー半径が変化しないものとする．このような部分木が複数存在する場合，中心が挿入するオブジェクトから最も近いものを選ぶ．そして，このような部分木が存在しなかったら，新しいオブジェクトを挿入することによって，部分木のカバー半径の変化が最も小さいものとする．

3.2 Isomap

Isomap アルゴリズム[5] は超高次元データの非線形な次元圧縮のために提案された．このアルゴリズムは，異なる環境で書かれた手書きの所有者は同一人物である，と判断ができる非線形な自由度を発見する他，ロールケーキ状のような特別な多様体の次元圧縮に成功した (図 2).

次元圧縮手順は１）近傍グラフを構築する２）各点の他の全ての点との最短距離を求める３）２）の結果で望ましい空間上での新しい位置を求める

この研究では，データを近傍グラフで表すことで２点間の本質的な類似性を保つことができる，と主張している．

3.3 k 近傍グラフ

k 近傍グラフ[9] は形状検索における最近傍検索のために提案された．特に，既存の形状検索用の索引 (VP-Tree,AESA,BK-Tree,GH-tree,GNAT など)[7] よりも，効率的な索引付け（距離計算を最小にする）を目指す．

k 近傍グラフの基本的なアイディアはドロネーグラフにおける最近傍検索である．ユークリッド空間上で作られるドロネーグラフの元となるボロノイ図を，超高次元距離空間用に考た．つまり，ドロネーグラフの近似である．

k 近傍グラフには各ノードが最も近い k 個のノードと辺で結ぶ．グラフが無向グラフである場合，各のノードの次数は k である．

4. 超高次元データに対する高速な検索の試み

4.1 M-Tree 索引の利用

図 3 は距離空間上にある点とその分布に対応する M-Tree を示す．以下，k 近傍を求める例を説明する．この例では，クエリ点 Q の k 近傍を求め，$k = 2$ とする．

検索はルートのスキャンから始まる．そのとき，k 番目の近傍への距離（k 近傍が存在する範囲）は無限大である．そして，ルートの各子ノード C_2，C_3，C_4 を $Q2$ への距離順にスキャンする．まず，C_2 をスキャンする．C_2 のクエリ点への距離 $d(C_2, Q)$ を計算し，無限大より小さいので，それを k 近傍が存在する範囲として保存しておく．そして，C_2 の子ノード C_5,C_6 を優先順位付きキューに格納する．C_3，C_4 に対しても同様である．この時点で，k 番目の近傍への距離の上界は $d(C_4, Q)$ である．

次に，キューに格納されているノードの中で，Q から最も近い C_5 をキューから取り出し，C_5 の子ノード a，b，c をスキャンする．まず，a をスキャンする．a は k 近傍が存在する範囲内（$d(C_4, Q)$）にあるので，k 近傍の候補として登録し，k 番目の近傍への距離の上界を $d(C_2, Q)$ に更新する．そして，キューに格納されている $d(C_2, Q)$ より遠いノードを削除

Vol.2009-MPS-76 No.47
Vol.2009-BIO-19 No.47
2009/12/18

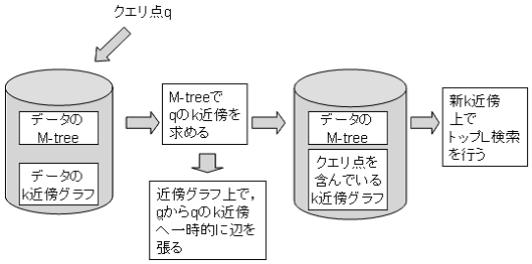

図 4 クエリの処理の流れ
Fig. 4 Processing the top l query

する．

この操作を繰り返すと，k 近傍として a，b を求められる．この例では，1 つのノードが 1 つのページであると考える場合，検索の I/O コストは 3 である．

4.2 M-Tree と k 近傍グラフを合わせた方法

4.1 節の例で述べたように，M-Tree で素早い l 近傍検索が実現できる．しかし，M-Tree のみで検索を行う場合，図 1 のような，グラフ上での距離の方が有意義なデータに対して，遺伝子発現量検索が失敗してしまう．そこで，我々は，k 近傍グラフと M-Tree を併用した検索手法を提案する．

図 4 は提案手法の処理の流れを示す．まず，前処理として，超高次元の点データの M-tree と k 近傍グラフを作成し，保存しておく．次に，ユーザがクエリ点 q の l 近傍を問い合わせる時，q を k 近傍グラフに組み込む．そして，q が組み込まれた k 近傍グラフ上で検索を行う．k 近傍グラフ上で，l 近傍を検索をするためには，ダイクストラ法を用いる．ダイクストラ法は、本来最短経路を求めるためのアルゴリズムだが，アルゴリズムからして，出発点から近い順に点を走査していくので、結果的に l 近傍が求められるということになる．ここは，クエリ点を始点とし，クエリ点から最も遠い点が終点となる．

q を k 近傍グラフに組み込む操作には，二つの処理をする必要がある．

(1) q の k 近傍を求め，そして q から，それらの点へ，辺を張る．
(2) k 近傍グラフのあるノードが，q がその点の k 近傍に含まれたら，その点の k 番目の辺を削除し，そして q へ辺を張る．つまり，辺の繋げ換えを行う．

処理 (1) は 4.1 節で説明した手法で求め，比較的簡単な問題である．処理 (2) に関しては，全てのノードに対して，q が k 近傍に含まれるか否かをチェックすることは，データ数を考

```
1  function ModifiedDijkstra1(Graph, querypoint):
2      for each vertex v in Graph:                          // Initializations
3          dist[v] := infinity                              // Unknown distance function from querypoint to v
4          previous[v] := undefined                         // Previous node in optimal path from querypoint
5      dist[querypoint] := 0                                // Distance from  to querypoint
6      S := the set of all nodes in Graph
       // All nodes in the graph are unoptimized - thus are in S
7      while S is not empty  or lnn has not been found:     // The main loop
8          u := vertex in S with smallest dist[]
9          if dist[u] = infinity:
10             break                                        // all remaining vertices are inaccessible from querypoint
11         remove u from S, u is one of  lnn
12         for each neighbor v of u (except the kth neighbor ):    // where v has not yet been removed from S.
13             alt := dist[u] + dist_between(u, v)
14             if alt < dist[v]:
15                 dist[v] := alt
16                 previous[v] := u
17         for the kth neighbor v of u
18             alt := dist[u] + dist_between(u, v)
19             if dist_between(u,v) < dist_between(u,query)   //checking edge(u,v),whether it is a virtual edge or not
20                 dist[v] := alt
21                 previous[v] := u
22             else
23                 remove edge(u,v)
24                 set a new ege (u,querypoint)               // querypoint as the kth neighbor of u
```

図 5 アルゴリズム 1

えると，非現実的である．そこで，計算コストを考慮し，我々は処理 (2) をダイクストラ法でグラフをたどりながら適宜辺の削除を行うこととした．

ダイクストラ法でグラフをスキャンしながら処理 (2) を実行する手法として，以下の 2 つが考えられる．手法 1 は単純な手法で，手法 2 は計算効率を考慮する手法である．

手法 1

アルゴリズム 1 は手法 1 を示す．手法 1 では，ノードをスキャンする際，q が近傍かどうかをチェックし，そうである場合，辺の繋げ換えを行う．手法 1 は確実にできる手法だが，ノードをスキャンする度に q への距離を測らなければいけない．類似した l 個のものを検索する場合，スキャンするノードは高々l だが，扱っているデータが超高次元データのため，時間コストはある程度かかる．

以下，図 7 に対して手法 1 の実行例（k=2，l=4）を説明する．まず，クエリ点 Q をスキャンする．スタート点に関しては，チェックを行う必要がない．次に，Q から最短距離を持つ C をスキャンする．チェックを行い，Q は C の k 近傍であるという結果が得られる．そのため，C の k 番目の辺を削除し，Q へ新しく辺を張る．そして，Q から次に最短距離を持つ G をスキャンする．チェックを行ったが，結果として近傍はそのままである．この操作を繰り返し，最も近い l 個のノードが求まったら，検索が終了する．

手法 2

Vol.2009-MPS-76 No.47
Vol.2009-BIO-19 No.47
2009/12/18

```
1  function ModifiedDijkstra2(Graph, querypoint):
2      for each vertex v in Graph:                          // Initializations
3          dist[v] := infinity                              // Unknown distance function from querypoint to v
4          previous[v] := undefined                         // Previous node in optimal path from querypoint
5      dist[querypoint] := 0                                // Distance from  to querypoint
6      S := the set of all nodes in Graph
       // All nodes in the graph are unoptimized - thus are in S
7      while S is not empty  or Inn has not been found:          // The main loop
8          u := vertex in S with smallest dist[]
9          if dist[u] = infinity:
10             break                                        // all remaining vertices are inaccessible from querypoint
11         if dist_between(previous[u],u) < dist_between(previous[u],q) //checking whether the edge to u is a virtual edge or not
12             remove u from S, u is one of  Inn
13         else
14             dist[u] := min(all_dist[u])                   // remove edge from previous[u] to u
15             previous[u] := min(prev_for_all_dist[u])      //previous node on path of dist[u]
16             set a new ege (u,querypoint)                  // querypoint as the kth neighbor of previous[u]
17         for each neighbor v of u (except the kth neighbor ):    // where v has not yet been removed from S.
18             alt := dist[u] + dist_between(u, v)
19             if alt < dist[v]:
20               all_dist[v][i]=dist[v]                      // save all computed dist[v]
21               prev_for_all_dist[v][i]=previous[v]         // save all visited previous[v]
22               dist[v] := alt
23               previous[v] := u
24             else
25               all_dist[v][i]:=alt
26               prev_for_all_dist[v][i]:=u
```

図 6 アルゴリズム 2

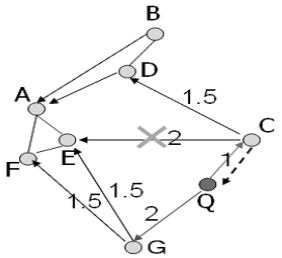

図 7 削除処理の例
Fig. 7 an example of checking/deleting edge process

q への距離計算を最小限に減らす方法として，手法 2 を提案する．手法 2 はアルゴリズム 2 で示す手順で行う．手法 2 では，k 番目の辺を最短パスとして通ろうとする時のみ，処理 (2) のチェックを行うので，計算コストが削減できる．しかし，スキャン済みノードの全てのパスとその距離を記録しなければいけない (アルゴリズム 21 行〜22 行，25 行〜26 行) ため，手法 1 よりも空間コストがかかる．

以下，図 7 に対して手法 2 の実行例（k=2，l=4）を説明する．まず，手法 1 と同じように，クエリ点 Q をスキャンする．次に，Q から最短距離を持つ C をスキャンする．ここで，

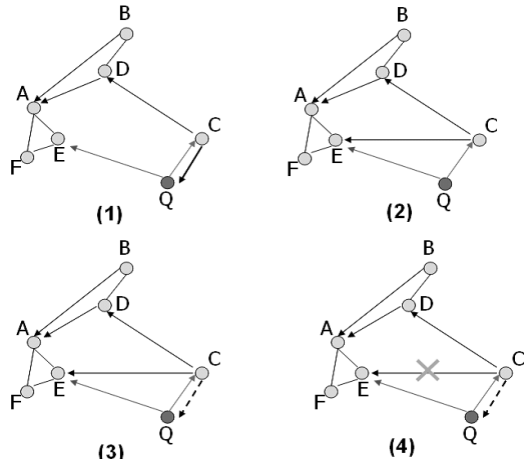

図 8 k=2 の kNN グラフ．(1)MethodA で生成した kNN グラフ，(2)MethodB で生成した kNN グラフ，(3)MethodC-I で生成した kNN グラフ，(4)MethodC-II で生成した kNN グラフ
Fig. 8 kNN graph(k=2). (1)Built with method A, (2)Built with method B, (3)Built with method C-I, (4)Built with method C-II

手法 1 と異なり，チェックを行わない．そして，Q から次に最短距離を持つ G をスキャンする．G は Q の k 近傍であるが，スタート点に限ってチェックを行わないためそのままスキャンする．次は，D をスキャンする．この時点では，近傍とその最短距離として，C（1），G（2），D（3.5）が得られた．次に最近である E をスキャンしようとするが，チェックが行われ，結果がポジティブであるため，辺の繋げ換えを行った．そして，次に最近である E（Q-G-E パスで）をスキャンし，全ての近傍が求まる．

以上の例では，ある k=2 のグラフに対して，l=4 を求めるために，チェック回数は，手法で 1 は 4 回，手法 2 では 1 回である．手法 2 の方が計算コストが小さいことが明確である．

5. 提案手法の評価

提案手法の有用性を測るために，人工データと遺伝子発現量データを使用し，評価実験を行う．実験では，次のそれぞれのグラフ上で検索を行った結果を比較・検討する．

- Method A) クエリ点を含んだグラフ ((図 8(1)). 問合せがあった時に k 近傍グラフをいちから作成し検索を実行する．このため，k 近傍グラフにはあらかじめクエリ点が含まれることになる．

Vol.2009-MPS-76 No.47
Vol.2009-BIO-19 No.47
2009/12/18

- Method B) クエリ点から差分的に辺を張ったグラフ ((図 8(2)). クエリ点を含まない k 近傍グラフをあらかじめ生成しておき，問合せがあった時に差分的にクエリ点を k 近傍に加える．その際クエリ点から k 近傍点への追加のみとし、辺の削除やつなぎ換えはしない．
- Method C-I) クエリ点から差分的に辺をつなぎ換えたグラフ (手法 1)(図 8(2)). クエリ点を含まない k 近傍グラフをあらかじめ生成しておき，問合せがあった時に差分的にクエリ点を k 近傍に加える．その際前節に述べた手法 1 で辺のつなぎ換えを行う．
- Method C-II) クエリ点から差分的に辺をつなぎ換えたグラフ (手法 1)(図 8(2)). クエリ点を含まない k 近傍グラフをあらかじめ生成しておき，問合せがあった時に差分的にクエリ点を k 近傍に加える．その際前節に述べた手法 2 で辺のつなぎ換えを行う．

期待される結果として，Method A の検索結果を正解の基準とする場合，4.2 節で述べた処理 (2) で得られた k 近傍グラフは，Method A のグラフとまったく同じなので，Method C (I と II) の検索結果は正確性がある．Method B の検索結果に関しては，必ずしも間違っているとは限らないと予測される．クエリ点の周辺が密である場合，検索結果の順位が多少変わるが，求めたい l 近傍は検索できる可能性がある．

計算コストについては，k 近傍グラフを一から構築するため，Method A が最も高い．そして，k 近傍グラフを予め用意しておき，またチェック回数が少ないので，最も小さいコストで検索を行うのは，Method C-II である．その次は，昇順で，Method C-I，そして Method B である．

5.1 人工データ

ロールケーキ状の多様体を構成する 6308 個の点を生成し，それらの点で k 近傍グラフを構築した．それぞれのデータは x 軸，y 軸，z 軸にあたる 3 つの属性を持っている．パラメータ k を 10 に設定し，それに対して l を 1 から 6308 までの値に変化させた．

各 l に対して，10 個の異なる点データで検索を行い，再現率，適合率，時間の平均を求めた．Method C-I と C-II に対して，更に，チェック回数も記録した．再現率と適合率はそれぞれ，Method A と比較したものである．Method-CI 及び Method C-II の再現率と適合率は常に 1 のため，グラフに表示しない．実験結果は図 9，10，11 に示す．

図 9 は Method B の適合率と再現率を示す．l の値が k より小さい場合と，全てのノードを検索する場合，再現率，適合率は共に 1 である．それ以外の場合は，適合率はほぼ 1 であるが，再現率は低い．これは毎回の検索に，Method B は l 個の検索ができたが，l 個の中に，回答ではないものもが含まれていることを表す．これは，最短パスとして記録されたパスが実際存在しないバーチャル辺を通り，最短距離が間違っていることが原因である．

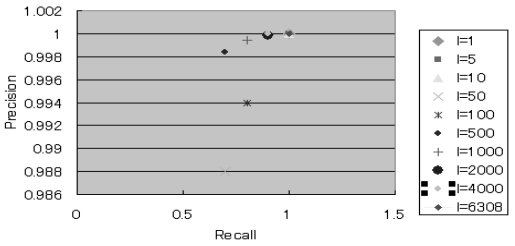

図 9 人工データの実験結果 (k=10)
Fig. 9 Experimental result for artificial data(k=10)

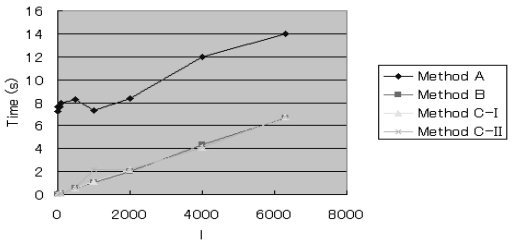

図 10 人工データの実験結果 (k=10)
Fig. 10 Experimental result for artificial data(k=10)

図 10 は各グラフ上での検索の実行時間を，図 11 は Method C-I と C-II のチェック回数を示す．Method B は問い合わせがあった際にグラフをいちから生成するため，他のグラフの検索時間と大きな差がある．l 個の近傍を検索するために，Method C-I の方がチェック回数が多いが，時間的に Method C-II とあまり変わらない結果となっている．今回の実験では，k 近傍グラフが全てメモリ上にあったため，チェック時に行う距離計算はあまり時間がかからない．そして，Method C-II では，毎回走査済みのノードの情報を記録しなければいけない（アルゴリズム 2 21〜22 行， 25〜26 行）ため，時間がかかる．

5.2 遺伝子発現量データ

データ数は 6318，その中から 10 個をクエリ点として問い合わせを行い，残りに対して k 近傍グラフを構築した．それぞれのデータは 474 属性を持ち，すなわち 474 次元を持つ点

Vol.2009-MPS-76 No.47
Vol.2009-BIO-19 No.47
2009/12/18

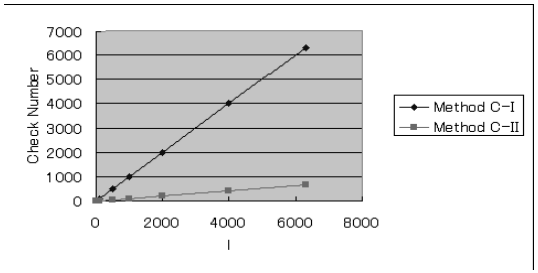

図 11 人工データの実験結果 (k=10)
Fig. 11 Experimental result for artificial data(k=10)

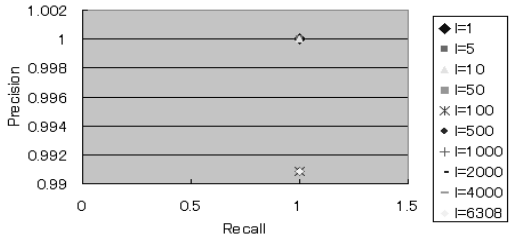

図 12 遺伝子発現量データの実験結果 (k=10)
Fig. 12 Experimental result for gene expression data(k=10)

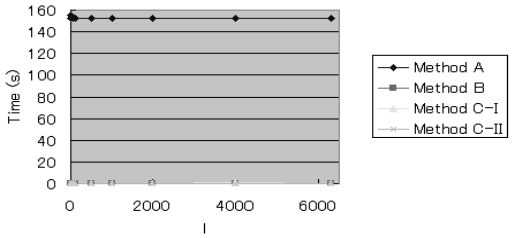

図 13 遺伝子発現量データの実験結果 (k=10)
Fig. 13 Experimental result for gene expression data(k=10)

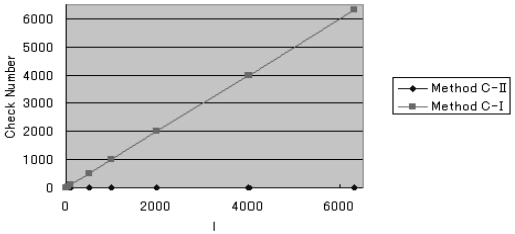

図 14 遺伝子発現量データの実験結果 (k=10)
Fig. 14 Experimental result for gene expression data(k=10)

となる．パラメータ設定は人工データと同じで，k=10，l は 0 から 6308 まである．実験結果を図 12，13，14 に示す．

実データを使用した実験では，Method B の再現率は全ての l に対して 1 だが，適合率は 2 つの値を持つ．l が k より小さい時，再現率は 1 であり，l が k より大きい時再現率は 0.99 である．これは，求めたい回答を全て検索できたが，回答ではないものも検索結果として返しているが原因である．Method C-I と C-II は再現率と適合率は常に 1 である．

検索時間においては，人工データと同じ傾向である．但し，この実験では，l をデータ数まで変化させたが，最終的に求められた回答数は 46 だった．そのため，l が l が k より大きい時，適合率，チェック回数，計算時間がずっと同じ値である．最終的に求められた回答数が求めたい数より小さいことは，k 近傍グラフが非連結であることが原因だと考える．

5.3 考察：Method B の正当性

設定パラメータを何種類かの値に変更し，Method B の正当性を検討した．Method B のグラフ上での検索結果は主に 2 種類ある．まず，距離が間違っているが，l 近傍は求まる．次は，求めたいものよりも，求まった結果が 1 個，2 個少ない．

l 近傍さえ求まったら良いとする状況で，結果が前者のようになる場合は，辺の繋げ換えを行う必要がない．つまり，余分な計算を行う必要がない．

後者に関しては，正当性がないとは言い切ることができない．応用分野によって，求めたいものより検索結果が少なくても大きな影響がないため，Method B を利用しても大丈夫な場合もあると考える．これに関しては，それぞれの応用分野での検討が必要だと考える．

Vol.2009-MPS-76 No.47
Vol.2009-BIO-19 No.47
2009/12/18

6. おわりに

本研究の目的は超高次元データの検索を高速化することである．超高次元データに対する検索を２つのケースに分けて考えた．超高次元データのみを考慮する場合，確立された技術である M-Tree を利用することができる．そして超高次元データが多様体をみなす場合，k 近傍グラフを利用する手法を提案した．提案手法を評価するために，検証実験を行った．結果として，超高次元データの類似検索に k 近傍グラフは利用できる．

しかし，実際の検索では，対象となるデータが膨大なため，k 近傍グラフが一度にメモリ上にロードできないと予測される．検索を行うために，k 近傍グラフを分割し，計算で必要な分割のみメモリーに呼び出す必要があると考える．効率の良い検索を行うために，k 近傍グラフをどのように分割するかが本研究の今後の課題である．

参 考 文 献

1) Ciaccia, P.,Patella, M and Zezula, P.: M-tree: An Efficient Access Method for Similarity Search in Metric Spaces, *Proceedings of the 23rd International Conference on Very Large Data Bases*, pp.426–435 (1997).
2) Harpaz, R. and Haralick, R.: Exploiting the Geometry of Gene Expression Patterns for Unsupervised Learning, *18th International Conference on Pattern Recognition (ICPR'06) Volume 2*, pp.670–674 (2006).
3) Haralick,R. and Harpaz,R.: Linear manifold clustering. *In 4th International Conference on Machine Learning and Data Mining in Pattern Recognition (MLDM 2005)*, pp.132-141 (2005).
4) Souvenir,R. and Pless,R.: Manifold clustering. *in Proceedings of the 10th International Conference on Computer Vision (ICCV 2005)*, pp.648-653 (2005).
5) Tenenbaum, J B.,Langford, J C and de Silva, V.: A Global Geometric Framework for Nonlinear Dimensionality Reduction, *SCIENCE*, Vol.290, No.9, pp.2319–2323 (2000).
6) Tenenbaum, J B.,Langford, J C and de Silva, V.: In Response to The Isomap Algorithm and Topological Stability, *SCIENCE(Technical Comments)*, Vol.295, No.9, pp.7a (2002).
7) Samet, H.: *Foundations of Multidimensional and Metric Data Structures*, Morgan Kaufmann Publishers, San Fransisco (2006).
8) 梅澤香矢乃, 瀬々, 潤: MARE: 遺伝子発現量検索エンジンの構築に関する一考察, **第 *1* 回データ工学と情報マネジメントに関するフォーラム**, (2009).
9) Sebastian, T.B.,and Kimia, B.B.: Metric-based shape retrieval in large databases, *Proceedings of the 16th International Conference on Pattern Recognition*, pp.291–296 (2002).