メトリック空間における Density Peaks Clustering に対する木構造を用いた効率的なアルゴリズム

中谷　諒平† 　天方　大地††,††† 　原　隆浩††

† 大阪大学工学部電子情報工学科　〒 565–0871 大阪府吹田市山田丘 1-5
†† 大阪大学大学院情報科学研究科　〒 565–0871 大阪府吹田市山田丘 1-5
††† JST さきがけ
E-mail: †{nakatani.ryohei,amagata.daichi,hara}@ist.osaka-u.ac.jp

あらまし　データマイニングの分野において，密度ベースクラスタリングが重要な課題となっている．密度ベースクラスタリングの手法の 1 つに Density Peaks Clustering（DPC）がある．また得られるデータの多様化および大規模化に伴ってメトリック空間における高速なクラスタリング手法が求められている．本研究ではメトリック空間における DPC の高速化に取り組む．DPC は類似度を用いて検索を行うが，メトリック空間で類似検索を高速に行うための木構造として multi vantage point tree（MVPT）が知られている．しかし，そのままの構造では DPC を高速に計算できない．この問題を解決するため MVPT を DPC に適用するためのアルゴリズムを提案する．また，マルチスレッド環境での効率的な並列アルゴリズムを提案する．人工データおよび実データによって，それぞれ異なる距離指標を用いて評価実験を行い，提案アルゴリズムの有効性を示した．

キーワード　Density Peaks Clustering, メトリック空間

1. はじめに

近年データマイニングの分野において密度ベースクラスタリングが注目されている．密度ベースクラスタリングの手法の一つに Density Peaks Clustering（DPC）がある [12]．このクラスタリングが対象とするデータはクラスタの中心付近の密度が高く，中心から離れるほど密度が低い分布を持つデータである．実データの多くがこの特徴をもつため，DPC は実データの特徴を捉える有用な手法である [12]．また，DPC は，自動運転 [11]，モーション検知 [20]，顧客セグメンテーション [16]，文書要約 [14]，コミュニティ検出 [1]，SNS による社会集団の発見 [15]，および単語分析 [9] などの幅広い分野での応用が考えられている．このように取得可能なデータの多様化および大規模化によってメトリック空間において高速に動作するアルゴリズムが求められている．そのため本研究ではメトリック空間における DPC の高速化に取り組む．DPC は「オブジェクト周りの密度」ρ と「自身よりも密度が高いオブジェクトの中で最も近いオブジェクトまでの距離」δ を計算することでそれぞれのオブジェクトのクラスタ上での位置を特徴づけるクラスタリング手法である．ρ と δ が大きいオブジェクトは周辺で最も密度が高いオブジェクトであるためクラスタセンタとし，他のクラスタメンバは δ の計算時に参照したオブジェクトと同じクラスタに所属させることでクラスタを形成する．

課題．DPC によってクラスタを高速に計算することを考える上で課題となる点をいくつか検討する．1 つ目の課題は δ を求めるための有効な方法が考えられていないことである．単純に自身から最も距離が近いオブジェクトを求める問題は考えられているが，そこに自身よりも密度が高いという条件を加えた計算を行う問題は考えられていない．単純な方法として，全てのオブジェクトの中から近いオブジェクトを探しその上でそのオブジェクトが自身よりも密度が高いか調べる方法が考えられる．しかし，この方法は効率的ではない．2 つ目の課題は高速化を実現するために並列処理を用いることが有効であるが，メトリック空間における DPC の効率的な並列化の手段が考えられていないことである．ユークリッド距離における DPC の並列化は考えられているが [17] [18] [21]，この手法ではグリッド分割が用いられているため，メトリック空間で用いることはできない．

提案アルゴリズムの概要．これらの問題を解決するため，シングルスレッドおよびマルチスレッド環境のそれぞれに対して，DPC の効率的なアルゴリズムを提案する．シングルスレッド環境での ρ の計算にはメトリック空間上で ρ の計算が高速な MVPT を，δ の計算には MVPT の構造に δ の計算のための構造を追加するアルゴリズムを提案する．またマルチスレッド環境での ρ の計算には計算時間をサンプリングによって推定し，負荷分散を行うアルゴリズムを，δ の計算にはシングルスレッド環境の δ の計算で行った MVPT の変更をさらに拡張し，並列化に適した構造を提案する．

貢献．以下に本稿の貢献を示す．

- メトリック空間における DPC に対する高速アルゴリズムを提案する．
- マルチスレッド環境での効率的なアルゴリズムを提案する．
- 人工データおよび実データを用いた実験により，提案ア

ルゴリズムの有効性を確認する．

本稿の構成． 2 章で本稿の予備知識を確認し，3 章で関連研究について述べる．4 章で提案アルゴリズムについて説明し，5 章で実験の結果を示す．最後に 6 章で本稿のまとめを述べる．

2. 予 備 知 識

本章では DPC，関連研究，および提案アルゴリズムに必要な定義を紹介する．

2.1 Density Peaks Clustering（DPC）

DPC においてクラスタセンタは「周辺で最も密度が高いオブジェクト」として特徴付けられる．そこでクラスタを計算する上で重要となるのは「あるオブジェクトの密度」と「自身よりも密度が高いオブジェクトの中で最も近いオブジェクトまでの距離」である．それらをそれぞれ次のように定義する．

定義 1 (ρ). オブジェクト o_i の周りの密度 ρ_i は，o_i から距離 d_c 以内に存在するオブジェクトの数であり，式 (1) により定義される．

$$\rho_i = \sum_j \chi(d(o_i, o_j) - d_c) \tag{1}$$

ただし

$$\chi(x) = \begin{cases} 1(x \leqq 0) \\ 0(x > 0) \end{cases}$$

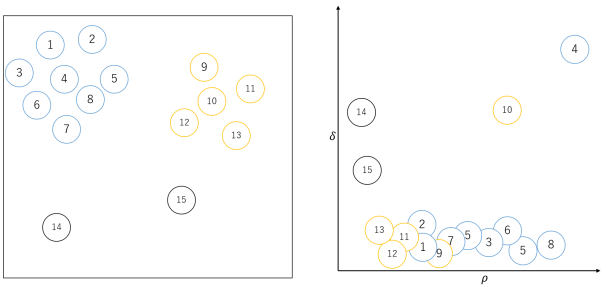
定義 2 (δ). δ_i はオブジェクト o_i よりも密度が高いオブジェクトの中で最も近いオブジェクトまでの距離であり，式 (2) により定義される．

$$\delta_i = \min_{j:\rho_i<\rho_j} (d(o_i, o_j)) \tag{2}$$

ただし，ρ が最も大きいオブジェクトの δ はそのオブジェクトから最も遠いオブジェクトまでの距離とする．

横軸に ρ，縦軸に δ を取り各オブジェクトをプロットしたものは decision graph と呼ばれる．ρ と δ が大きいオブジェクトは自身の周りの密度が高く，かつ周辺で最も密度が高いオブジェクトであるためクラスタセンタであることが分かる．また ρ が大きく δ が小さいオブジェクトは自身の周りの密度が大きく，近くに自身より密度が高いオブジェクトが存在するためクラスタメンバであることが分かる．このときクラスタメンバが所属するクラスタは δ の計算時に参照したオブジェクトと同様である．また ρ が小さく δ が大きいオブジェクトは周囲にオブジェクトがあまり存在しないためノイズであることがわかる．閾値を用いてクラスタセンタ，クラスタメンバ，およびノイズとなるオブジェクトを決めることで最終的な出力であるクラスタリング結果を得る．

例 1. 図 1 に 2 次元空間におけるオブジェクト集合の decision graph の例を示す．図 1(a) が 2 次元空間におけるオブジェクトを表しており，図 1(b) がその decision graph である．また，2 つのクラスタをそれぞれ黄色および青色で表しており，黒いオブジェクトはノイズである．クラスタセンタであるオブジェクト 4 およびオブジェクト 10 が，decision graph 上で ρ および δ が大きい右上の部分に現れている．また，ノイズであるオブジェクト 14 およびオブジェクト 15 は ρ が小さく δ が大きい左上の部分に現れている．



(a) オブジェクト集合　(b) decision graph

図 1: 2 次元空間での DPC

時間計算量． DPC の単純な実装の時間計算量を確認しておく．オブジェクトの数を N とする．まず ρ を求める場合は全てのオブジェクト対の距離が，d_c 以下か否かを調べるため $O(N^2)$ である．δ についても同様に $O(N^2)$ である．この二つから最終的なクラスタリング結果を得る部分については十分高速に動作することが知られているため，これ以降は ρ と δ を求める処理の高速化について議論する．

2.2 メトリック空間

メトリック空間は距離空間とも呼ばれる，距離関数が与えられている空間である．与えられる距離として以下の式を満たす任意の関数を想定しており，多様なデータ集合に対応できる．

定義 3 (距離関数). 集合 X 上のオブジェクト $o_1, o_2,$ および o_3 に対して関数 d が以下を満たすとき，d は X の距離関数であるという．

$$d(o_1, o_2) \geqq 0 (\text{非負性}) \tag{3}$$

$$d(o_1, o_2) = d(o_2, o_1) (\text{対称性}) \tag{4}$$

$$d(o_1, o_2) + d(o_2, o_3) \geqq d(o_1, o_3) (\text{三角不等式}) \tag{5}$$

2.3 multi vantage point tree（MVPT）

DPC を高速に計算するために類似探索を高速に行うことが重要である．MVPT はメトリック空間上での範囲検索や kNN 検索を高速に行うためのデータ構造である [2].

2.3.1 MVPT の構造

図 2 に MVPT の構造を示す．この木は節点にピボットと呼ばれるオブジェクトを一つ決め，そのオブジェクトからの距離によって子を分割することで得られる木である．例えば図 2 では，2 と 5 はピボット o_2 からの距離を表している．つまり，$o_2, o_4,$ および o_8 は o_2 からの距離が 2 以下，$o_1, o_3,$ および o_7 は o_2 からの距離が 2 以上 5 未満，$o_5, o_6,$ および o_9 は o_2 からの距離が 5 以上であることを表している．分割を行った後もノードのもつオブジェクトの数が大きい場合は同じようにピボットを選び分割を行う．そのときに分割の基準となる距離を適切に選ぶことで全体として平衡木となる．

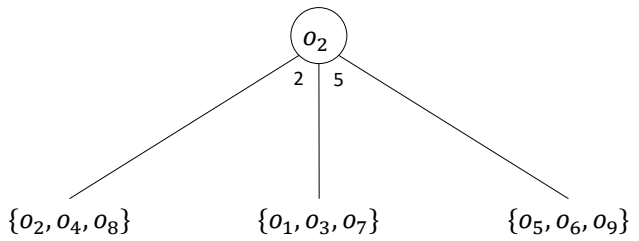


図 2: MVPT の構造

2.3.2 MVPT の構築

MVPT を構築する手順を示す．まず根ノードに全てのオブジェクトをもたせることを考える．ここで根ノードのもつオブジェクト数が多すぎる場合はそのオブジェクト集合を分割して子ノードを作る．ピボットからの距離の昇順にもっているオブジェクトをソートし，m 個に等分割する．m 分割したオブジェクトをそれぞれ子ノードに割り当てる．このようにしてできた子ノードのオブジェクト数もまた多すぎる場合には根ノードに対して行った処理を同様に繰り返す．この手順を全ノードのもつオブジェクトの数が一定以下になるまで繰り返す．

2.3.3 MVPT を用いた検索

次に，完成した木構造を用いて求めるオブジェクトを検索することを考える．ここではまずオブジェクト o_q と距離 d_c に対して，o_q から距離 d_c 以内のオブジェクトを求める範囲検索問題を考える．このとき，根ノードから葉の方向に探索することを考える．式 (5) の三角不等式より o_q とピボット o_p，o_p の子孫のあるオブジェクト o_s に対して，$d(o_p, o_s) \leqq d(o_p, o_q) + d(o_q, o_s)$ が成立する．またこの o_s が解となるとき，$d(o_q, o_s) \leqq d_c$ が成立する．これらから解となるオブジェクトが満たすべき $d(o_p, o_s)$ の範囲が分かるため，それにしたがって不必要な距離計算を枝刈りできる．

アルゴリズム 1 は MVPT による範囲検索の手順を示している．ここでは構築済みの MVPT t 上でオブジェクト o_q から距離 d_c 以内のオブジェクトを探す．また距離関数として d を用いる．このアルゴリズムでは *searchNode* で辿るべきノードを管理する．根ノードから順に辿るため，まず *searchNode* に根ノードを *push* する（1 行）．次の while ループがこのアルゴリズムのメインとなる部分である．*searchNode* からノードを一つ取り出しそのノードが葉ノードだった場合は葉ノードに格納されたオブジェクト o_p.*objects* を走査し，解となるオブジェクトを *ans* に *push* する（3–7 行）．逆に節点ノードだった場合はその子ノード o_p.*children* のうち，三角不等式に基づいて解をもつ可能性のあるノードを *searchNode* に *push* する（8–13 行）．この操作を *searchNode* が空になるまで繰り返す．これにより，o_q から距離 d_c 以内のオブジェクトを全て見つけることができる．

次に，オブジェクト o_q に最も近いオブジェクトを求める最近傍探索問題を考える．この問題においても先ほどと同じように三角不等式による枝刈りを考える．しかしこの問題の場合は検索すべき範囲が分からないため探索しながら範囲の変更を行う．具体的には，まず探索範囲として十分に大きな値をとり，範囲検索と同じ要領で探索を行う．その過程で今の探索範囲以内にオブジェクトが見つかればそのオブジェクトを解候補として保持しておき，そのオブジェクトと o_q との距離を新しい探索範囲とする．

Algorithm 1: MVPT を用いた範囲検索

Input: t, o_q, d_c, d
Output: t 上でオブジェクト o_q から距離 d_c 以下のオブジェクトを持つ配列 *ans*

```
1  searchNode.push(t.root)
2  while searchNode が空でない do
3      o_p ← searchNode.pop()
4      if o_p が葉ノード then
5          foreach o_s ∈ o_p.objects do
6              if d(o_s, o_q) ≦ d_c then
7                  ans.push(o_s)
8      else
9          foreach childNode ∈ o_p.children do
10             d_min ← childNode のピボットからの距離の最小値
11             d_max ← childNode のピボットからの距離の最大値
12             if d_min ≦ d(o_p, o_q) ≦ d_max または
                  |d_min − d(o_p, o_q)| ≦ d_c または
                  |d(o_p, o_q) − d_max| ≦ d_c then
13                 searchNode.push(childNode)
14 return ans
```

アルゴリズム 2 は MVPT t を用いて，オブジェクト o_q に最も近いオブジェクトを求める手順を示している．2 行目で定義した *searchRadius* を探索範囲として，範囲内のオブジェクトを検索する．その上で範囲内のオブジェクトが見つかれば *searchRadius* をそのオブジェクトまでの距離に変更する (8 行)．*searchRadius* の更新はこの部分（8 行）でしか行われない．また，*searchRadius* が小さいほど辿る必要のあるノードの数が減り，探索が高速になる．そのため，可能な限り早い段階で *searchRadius* の更新を行うことで，効率的に探索を実行できる．そこで，while ループ内で可能な限り早く葉ノードにアクセスするため，*searchNode* として用いる構造は最後に *push* したノードが優先的に *pop* されるスタックの構造を用いる．以上によってオブジェクト o_q に最も近いオブジェクトを求めることができる．

3. 関 連 研 究

文献 [12] では初めて DPC を提案している．DPC は任意の形のクラスタを発見し，自動的にクラスタ数を決定する新しいクラスタリングアルゴリズムである．類似したクラスタリングアルゴリズムである DBSCAN [4] と比較して，DPC は必要なパラメータが少ない．またそのパラメータの決定が容易である．文献 [13] ではグラフ構造を用いた kNN ベースの DPC の

Algorithm 2: MVPT を用いた最近傍検索

Input: t, o_q
Output: o_q に最も近いオブジェクト ans

```
searchNode.push(t.root)
searchRadius ← 十分大きな値
while searchNode が空でない do
    o_p ← searchNode.pop()
    if o_p が葉ノード then
        foreach o_s ∈ o_p.objects do
            if d(o_s, o_q) < searchRadius then
                searchRadius ← d(o_s, o_q)
                ans ← o_s
    else
        foreach childNode ∈ o_p.children do
            d_min ← childNode のピボットからの距離の最小値
            d_max ← childNode のピボットからの距離の最大値
            if d_min ≦ d(o_p, o_q) ≦ d_max または
              |d_min − d(o_p, o_q)| ≦ searchRadius または
              |d(o_p, o_q) − d_max| ≦ searchRadius then
                searchNode.push(childNode)
return ans
```

高速化アルゴリズムを提案している．この手法では ρ の定義に範囲ではなく，kNN を用いた kNN-density を用いている．kNN-density の大きいオブジェクトと小さいオブジェクトのそれぞれに適した 2 つの δ の計算アルゴリズムを用いることで高速化な動作を実現している．文献 [17] [18] [21] ではグリッドベースの DPC の高速化アルゴリズムを提案している．この手法ではグリッド分割を行うことで距離計算を行う必要のあるオブジェクトを近くのオブジェクトに限定し，高速な計算を実現している．しかしこの方法はユークリッド距離のみを対象としており，他の距離指標に適用できない．文献 [6] ではストリーミングデータに対する DPC の応用およびその高速化アルゴリズムを提案している．この手法では δ の計算に参照しているオブジェクトを管理するための木構造 DP-Tree を提案している．データの変更のあった箇所のみに対してこの木を更新することで高速化している．この方法はストリーミングデータを対象としているため本研究とは問題が異なる．文献 [10] では Map Reduce による並列化を用いた高速化手法が提案されている．これを用いることで Spark 上で高速に DPC を計算することができる．文献 [19] では Pre-Screening を用いたアルゴリズム，PDPC を提案している．この手法では Pre-Screening の過程で近似を行っている．文献 [8] では GPU を用いた並列化プログラミングによる高速化が提案されている．この研究では GPU による高速化が考えられているのに対して，本研究では CPU による高速化を考えている．

4. 提案アルゴリズム

本章では 1 章で述べた問題を解決する提案アルゴリズムの詳細を紹介する．まず，シングルスレッド用のアルゴリズムを紹介し，その後，マルチスレッド用のアルゴリズムについて述べる．

4.1 シングルスレッド環境での DPC

ρ の計算． シングルスレッド環境での ρ の計算は 2.3.3 項で述べた MVPT における範囲検索アルゴリズムを用いればよい．

δ の計算． 2 章で定義したように δ とは「自身よりも密度が高いオブジェクトの中で最も近いオブジェクトまでの距離」である．δ を求めるためには「自身よりも密度が高いオブジェクトのみで構成されたオブジェクト集合に対する最近傍探索問題」を解けばよい．しかし MVPT は構築済みの木への点の追加ができないため，そのままでは δ の計算に MVPT を使うことができない，

そこで ρ の計算に用いた MVPT を δ の計算にも用いることを考える．具体的にはオブジェクトを ρ の大きい順に MVPT で計算を行い，MVPT 上の葉ノードのオブジェクトにフラグを設け，計算したオブジェクトから順にそのオブジェクトのフラグを立てる．これにより自身より密度が高いオブジェクトのみにアクセスすることが可能となる．またそれと同時に節点ノードにもフラグを設け，計算したオブジェクトの祖先の節点ノードの全てのフラグを立てる処理を行うことにより，その節点が子孫に辿るべきオブジェクトをもつかどうかが分かるため無駄なノードへのアクセスを避けることができる．

4.2 マルチスレッドでの DPC

ρ の計算． MVPT 上での範囲検索は各オブジェクトが独立して実行できるため，単純な並列化が考えられる．しかしこの方法はそれぞれのオブジェクトに対する ρ の計算時間のばらつきが大きいとき，効率的に並列化できない．そこであらかじめそれぞれのオブジェクトに対する計算時間を推定し，それに基づいて並列化を行うことを考える．ここで計算時間の推定を行う上で以下の仮定をおく．

仮定． MVPT 上で同じ葉ノードに所属するオブジェクトは，そのオブジェクトに対する範囲検索問題の計算時間が似通ったものになる．例えば図 2 で同じ葉ノードに所属する o_1 と o_3 に対する範囲検索の計算時間が同様になりやすい．

この仮定を用いると，それぞれの葉ノードから 1 つずつオブジェクトをサンプリングし，そのオブジェクトに対して実行した範囲検索の時間を同じ葉ノードに含まれるオブジェクトに対する範囲検索の時間と推定できる．推定されたコストと貪欲法を用いて，各スレッドに割り当てるオブジェクトを決定する．貪欲法では，各スレッドに対して，それに割り当てるオブジェクトの推定計算のコストの和を保持する．あるオブジェクトが与えられた際，コストの和が最小のスレッドに割り当てる．

δ の計算． 前節のシングルスレッド環境での δ の計算アルゴリズムは密度が高いオブジェクトから順に計算を行う必要があるため，並列化できない．そこで MVPT にもたせる情報を変更することで並列化を行う．シングルスレッド環境では自身よりも密度が高いかを $true$ と $false$ のフラグで扱った．そのフラグをもつ目的は「そのオブジェクトもしくはその節点ノードを

表 1: データセット

データセット	データ数	距離関数
SS	1,500,000	ユークリッド距離
dictionary	30,916	編集距離
3D road network	434,874	マンハッタン距離

辿るべきか（自身よりも密度が高いオブジェクトを含むか）を判別すること」である．節点ノードに自身の子孫のうちで最も ρ が大きいオブジェクトの ρ の値をもたせておけば，密度が大きいオブジェクトから順に処理を行う必要なく辿るべきノードを判別できる．具体的には，δ を求めようとしているオブジェクトの密度が節点ノードで管理している ρ よりも大きければ，その子孫を辿る必要がない．この方法は各オブジェクトに対して独立して実行できるため，並列化できる．これを実現するために ρ の計算の終了後，各ノードの更新を行い，全てのノードに自身とその子孫の中で最も密度が大きいオブジェクトの ρ を保持させる．

5. 評価実験

本章では提案アルゴリズムの性能評価のために行った実験の結果を紹介する．本実験は Windows 10 Pro, 3.20GHz Intel Core i7, コア数 6, 論理スレッド数 12, および 64GB RAM を搭載した計算機で行い，全てのアルゴリズムは C++で実装を行った．

5.1 セッティング

5.1.1 実験対象と比較アルゴリズム

本実験ではクラスタリング全体の実行時間を確認し，提案手法の有効性を確認した．また，マルチスレッド環境での ρ の計算，シングルスレッド環境での δ の計算，およびマルチスレッド環境での δ の計算の 3 つについて実験し評価を行った．比較アルゴリズムとしてマルチスレッド環境での δ の計算に，MVPT で推定コストを用いずに並列化を行った simple MVPT を用いた．シングルスレッド環境での δ の計算には，全てのオブジェクトと比較を行う scan とメトリック空間で働く木構造である M-Tree [3] を用いた．

5.1.2 データセット

実験には 3 つのデータセットを用いた．データセットの詳細を表 1 に示す．

SS は文献 [5] で提案されているクラスタリング用データセット生成アルゴリズムを用いて，作成した人工データである．3 次元のデータで，データ数を 100,000 から 1,500,000 まで変更して生成した．その他のデータセットの生成に用いるパラメータは文献 [5] に従っている．

dictionary(注1)は Moby Project から取得した名詞，接頭語，および複合語などの文字列のデータセットである．実験ではこのうちの人名および地名のデータを用いた．このデータセットは距離関数として編集距離を使用する．この関数は距離を計算する 2 つの文字列 S_1 および S_2 に対して計算量が $O(|S_1||S_2|)$ であるため，長い文字列に対しては計算時間が大きくなる．そのため単語ごとの距離計算時間のばらつきが大きくなり，マルチスレッド環境での ρ の分割の際に用いた範囲検索の時間の推定が上手く働かない．これを解決するため，このデータセットに関しては推定の計算時間を（サンプルの計算時間）/（サンプルの文字数）×（追加するデータの文字数）としている．

3D road network [7] は，3 次元の道路上の座標を表したデータである．また，距離関数としてマンハッタン距離を用いる．

(注1)：https://www.gutenberg.org/ebooks/3201

表 2: パラメータの設定

パラメータの設定	値
d_c （SS）	50, 100, 150, **200**
d_c （dictionary）	2,**3**,4,5
d_c （3D road network）	0.5,**1.0**,1.5,2.0
データ数（SS）	25, 50, 75, **100**, 125, 150($\times 10^4$)
使用スレッド数	2, 4, **6**, 8, 10, 12

5.1.3 パラメータ

表 2 に本実験で用いたパラメータを示す．太字で表されている値はデフォルトの値である．d_c はデータセットごとに適切な値を選んで用いている．

5.2 評価結果

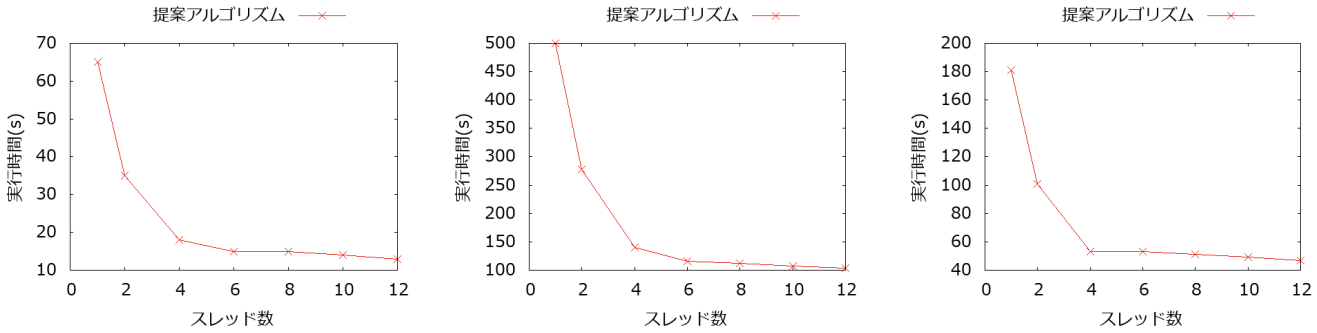
図 3 に提案手法によるクラスタリングの実行時間を示す．使用したスレッド数が 1 の場合がシングルスレッドの場合を表している．大規模なデータセットに対しても高速に動作していることがわかる．またスレッド数の増加に伴って，実行時間が短くなっている．

5.2.1 マルチスレッド環境での ρ の計算

図 4 にマルチスレッド環境での ρ の計算時間の結果を示す．いずれのデータセットに対しても提案アルゴリズムの方が高速に動作しており，推定コストによるスレッドへの割り当てが上手く働いていることが分かる．また SS では中心の密度が高く，周囲のクラスタとの距離が遠いクラスタが生成されやすい．これによって MVPT で同じ葉ノードに所属するオブジェクトが似たオブジェクトになる可能性がより高くなり，比較アルゴリズムとより大きな差がついていると考えられる．

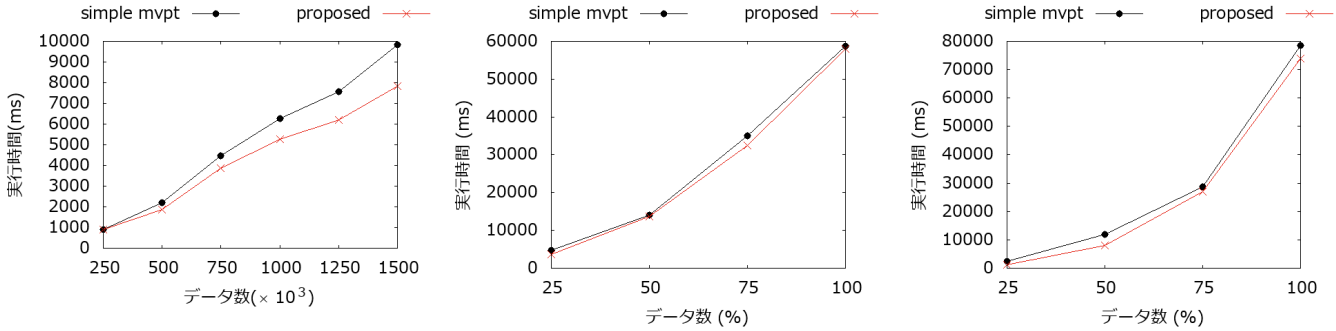
スレッド数の影響．図 5 に使用スレッド数を変化させたときの結果を示す．スレッド数を増やすとどちらのアルゴリズムも実行時間が短くなっているが，いずれの場合も提案アルゴリズムの方が高速に動作している．またスレッド数の増加に対する実行時間の減り幅は提案アルゴリズムの方が大きい．これはスレッド数が増加しても効率的な負荷分散ができているからである．

d_c の影響．図 6 に d_c を変化させたときの結果を示す．どちらのアルゴリズムも d_c が大きくなるにつれて計算時間が大きくなっている．これは d_c の増加に伴って MVPT 上で辿るべきノードの数が増加するからである．また提案アルゴリズムの方が d_c の増加による実行時間の増え方が小さい．1 つのオブジェクトに対する ρ の計算にかかる時間が大きくなったことで分割の単位が大きくなっているため，simple MVPT では上手く負荷分散できなくなっている．



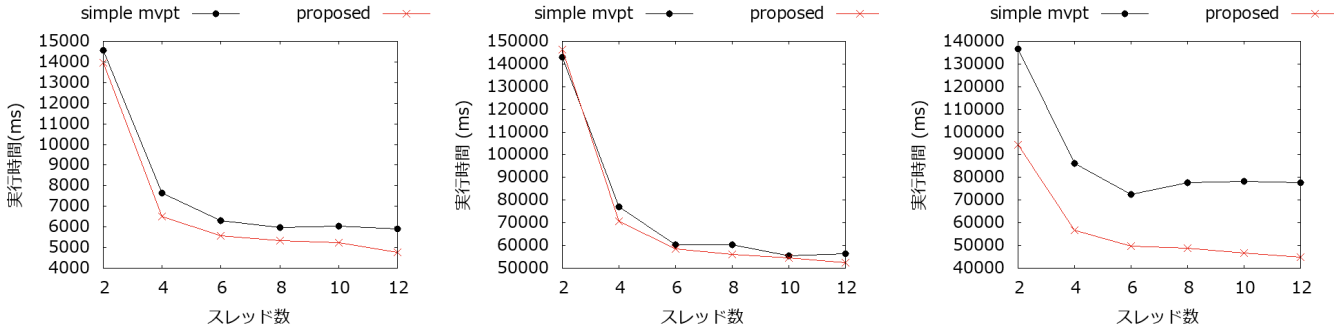
(a) SS (b) dictionary (c) 3D road network

図 3: クラスタリングの実行時間



(a) SS (b) dictionary (c) 3D road network

図 4: マルチスレッド環境での ρ の計算結果



(a) SS (b) dictionary (c) 3D road network

図 5: マルチスレッド環境での ρ の計算におけるスレッド数の影響

5.2.2 シングルスレッド環境での δ の計算

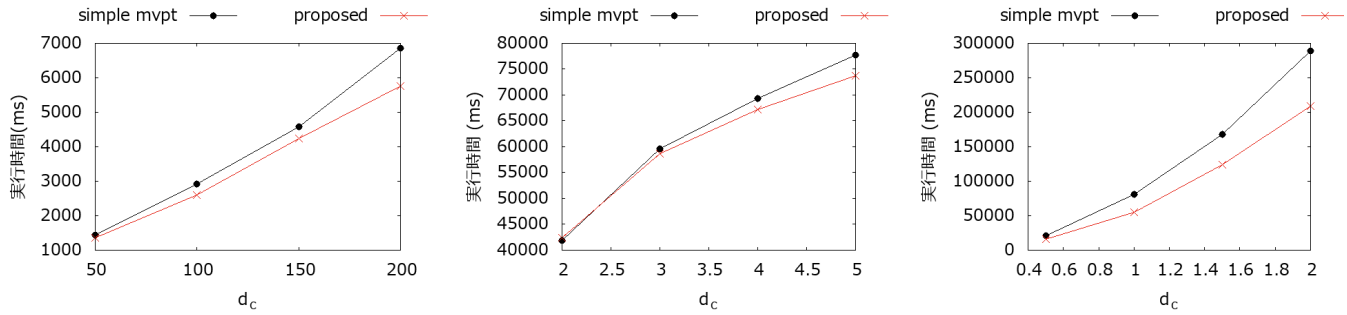
図 7 にシングルスレッド環境での δ の計算時間の結果を示す．データ数が増えるにしたがって実行時間が大きくなっていることが分かる．データセット SS と 3D road network に関してはどのデータ数に対しても MVPT を用いた手法が最も高速に動作した．M-Tree は MVPT とは違い逐次的な更新が可能であるためより簡単に δ を求める問題に適応可能だが，局所的にオブジェクトが集まっているようなデータセットに対しては MVPT の方が効率的に枝刈りができるため，MVPT が最も高速に動作している．一方 dictionary に関してはこの傾向が小さいため M-Tree の方が高速に動作している．

5.2.3 マルチスレッド環境での δ の計算

図 8 にマルチスレッド環境での δ の計算時間の結果を示す．使用するスレッド数が増えるにしたがって実行時間が小さくなっていることが分かる．マルチスレッド環境ではこのアルゴリズムを用いることで実行時間の短縮が行われることが分かる．

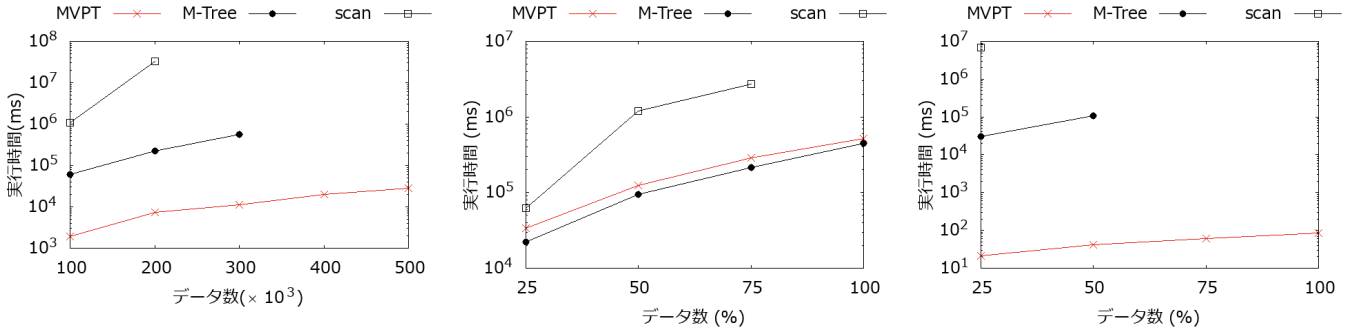
6. ま と め

近年データマイニングの分野において密度ベースクラスタリングが注目されている．また得られるデータの多様化および大規模化に伴って，メトリック空間上での高速なクラスタリングアルゴリズムが重要視されるようになった．本研究ではその



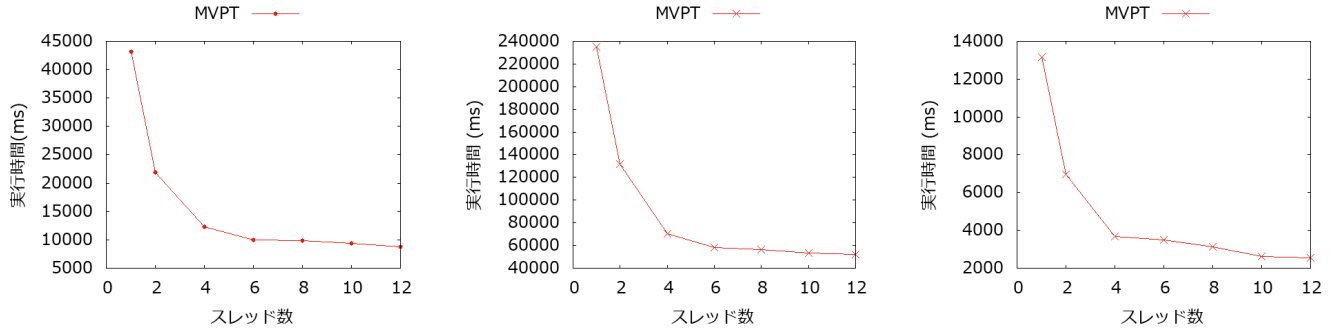
(a) SS (b) dictionary (c) 3D road network

図 6: マルチスレッド環境での ρ の計算における d_c の影響



(a) SS (b) dictionary (c) 3D road network

図 7: シングルスレッド環境での δ の計算結果



(a) SS (b) dictionary (c) 3D road network

図 8: マルチスレッド環境での δ の計算結果

傾向から密度ベースクラスタリング手法の 1 つである desity peaks clustering（DPC）のメトリック空間における高速に取り組んだ．DPC によって求めたい 2 つの重要な値 ρ および δ のそれぞれに対してアルゴリズムを提案した．ρ の計算に対してはマルチスレッド環境での計算時間のバランスを改善するためのサンプリングを用いたコストの推定アルゴリズムを提案した．また δ の計算に関してはメトリック空間上で高速に動作する木構造である MVPT を DPC の問題に適応し，効率的な枝刈りを行うアルゴリズムを提案した．評価実験の結果から提案アルゴリズムの有効性を確認した．

謝辞．本研究の一部は，文部科学省科学研究費補助金・基盤研究 (A)(18H04095) および JST さきがけ（JPMJPR1931）の支援を受けたものである．ここに記して謝意を示す．

文　献

[1] Bai, X., Yang, P., Shi, X.: An overlapping community detection algorithm based on density peaks. Neurocomputing 226, 7–15 (2017)

[2] Chen, L., Gao, Y., Zheng, B., Jensen, C.S., Yang, H., Yang, K.: Pivot-based metric indexing. PVLDB 10(10), 1058–1069 (2017)

[3] Ciaccia, P., Patella, M., Rabitti, F., Zezula, P.: Indexing metric spaces with m-tree. In: SEBD. vol. 97, pp. 67–86 (1997)

[4] Ester, M., Kriegel, H.P., Sander, J., Xu, X., et al.: A density-based algorithm for discovering clusters in large spa-

tial databases with noise. In: KDD
[5] Gan, J., Tao, Y.: Dbscan revisited: mis-claim, un-fixability, and approximation. In: SIGMOD. pp. 519–530. ACM (2015)
[6] Gong, S., Zhang, Y., Yu, G.: Clustering stream data by exploring the evolution of density mountain. PVLDB 11(4), 393–405 (2017)
[7] Kaul, M., Yang, B., Jensen, C.S.: Building accurate 3d spatial networks to enable next generation intelligent transportation systems. In: MDM. vol. 1, pp. 137–146 (2013)
[8] Li, M., Huang, J., Wang, J.: Paralleled fast search and find of density peaks clustering algorithm on gpus with cuda. In: SNPD. pp. 313–318 (2016)
[9] Liu, H., Guan, H., Jian, J., Liu, X., Pei, Y.: Clustering based on words distances. Cluster Computing 21(1), 945–953 (2018)
[10] Liu, R., Li, X., Du, L., Zhi, S., Wei, M.: Parallel implementation of density peaks clustering algorithm based on spark. Procedia Computer Science 107, 442–447 (2017)
[11] Lu, K., Xia, S., Xia, C.: Clustering based road detection method. In: CCC. pp. 3874–3879 (2015)
[12] Rodriguez, A., Laio, A.: Clustering by fast search and find of density peaks. Science 344(6191), 1492–1496 (2014)
[13] Sieranoja, S., Fränti, P.: Fast and general density peaks clustering. Pattern Recognition Letters 128, 551–558 (2019)
[14] Wang, B., Zhang, J., Liu, Y., Zou, Y.: Density peaks clustering based integrate framework for multi-document summarization. CAAI 2(1), 26–30 (2017)
[15] Wang, M., Zuo, W., Wang, Y.: An improved density peaks-based clustering method for social circle discovery in social networks. Neurocomputing 179, 219–227 (2016)
[16] Wang, Y., Chen, Q., Kang, C., Xia, Q.: Clustering of electricity consumption behavior dynamics toward big data applications. IEEE transactions on smart grid 7(5), 2437–2447 (2016)
[17] Wu, B., Wilamowski, B.M.: A fast density and grid based clustering method for data with arbitrary shapes and noise. IEEE TII 13(4), 1620–1628 (2016)
[18] Xu, X., Ding, S., Du, M., Xue, Y.: Dpcg: an efficient density peaks clustering algorithm based on grid. IJMLC 9(5), 743–754 (2018)
[19] Xu, X., Ding, S., Sun, T.: A fast density peaks clustering algorithm based on pre-screening. In: BigComp. pp. 513–516 (2018)
[20] Zhang, Y., Li, G., Xie, X., Wang, Z.: A new algorithm for fast and accurate moving object detection based on motion segmentation by clustering. In: MVA. pp. 444–447 (2017)
[21] Zheng, J., Sun, H., Zhang, M., Zhang, G.: Research on optimization of clustering by fast search and find of density peaks. In: WISA. pp. 129–133 (2016)