DEIM2022 H34-3

# BLOCK-OPTICS：密度ベースクラスタリング手法OPTICSの高速化

湯川　皓太†　　天笠　俊之††

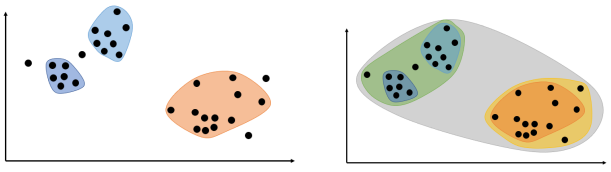
† 筑波大学システム情報工学研究科　〒 305–8573 茨城県つくば市天王台 1-1-1
†† 筑波大学計算科学研究センター　〒 305–8573 茨城県つくば市天王台 1-1-1
E-mail: †yukawa@kde.cs.tsukuba.ac.jp, ††amagasa@cs.tsukuba.ac.jp

**あらまし**　クラスタリングはデータ点同士の類似度に基づき，データをグループ分けする教師なし機械学習である．中でも，密度ベースのクラスタリングはデータ点が密な領域が疎な領域によって分離されている領域をクラスタとして検出する．Ordering Points To Identify the Clustering Structure (OPTICS) は密度ベースクラスタリングの一種で，任意の形状のクラスタを検出できるという利点がある．一方で，全てのデータ点に対して密度の計算を行う処理を必要とするため，高次元，大規模なデータセットに対しては計算量が膨大になってしまう．この問題に対して，計算の必要のないデータ点に対しての計算を削減することによって全ての点に対して計算を行うことなく厳密な解を得られるような手法，BLOCK-OPTICS を提案する．実験ではいくつかの実世界データセットを用いて厳密解を得た上で，実行時間を削減できていることを示した．

**キーワード**　クラスタリング，DBSCAN，OPTICS

## 1　はじめに

クラスター分析とは互いに類似したデータの集合を一つのクラスターとして，データセット内のクラスターを抽出するデータ分析の手法である．クラスター分析は教師データの与えられない非教師型のデータ分析の手法である．そのためクラスター分析を用いることによって，データセット内に潜在的に存在するクラスターを抽出することが可能になる．クラスター分析の例としてアンケート等のクラスター分析がある．アンケート等で回答した内容を用いてクラスター分析を行うことによって，アンケート回答者のグループや，回答者の指向に対する潜在的なグルーピングを自動で行うことが可能になる．このようにクラスター分析には事前に人間が識別した教師データを必要とする情報分類とは違い，データセットに存在する未知のクラスターを発見することが可能になる．



**(a)** 非階層型クラスタリング　　**(b)** 階層型クラスタリング

**図 1:** 非階層型クラスタリングと階層型クラスタリングの違い

クラスター分析には大きく分けて階層型クラスター分析と非階層型クラスター分析に分類することができる．階層型クラスター分析はデータが属するクラスターが複数の階層に分かれて形成されているようなクラスターを抽出する分析である．一方で，非階層型クラスター分析は，クラスターに階層が存在せず，あるデータは単一のクラスターに属するようになる．階層型クラスター分析と非階層型クラスター分析の違いを図 1 に示す．階層型クラスター分析は非階層型クラスター分析と比較して，データに潜在的な階層構造が存在する場合によりデータセットに関する深い知見を得ることが可能になる．

クラスター分析の中でも特に密度ベースのクラスタリングでは，各データ点に対して一定距離範囲内に存在するデータ点の個数を密度として計算し，密な領域と疎な領域を分類してクラスタリングをする．これによってクラスタ数を指定せずに任意の形状のクラスタを発見することが可能になる．一方で，密度ベースのクラスタリング手法は一般的に全てのデータ点に対して距離計算を用いた密度の計算を必要とする．そのため，大規模，高次元なデータセットに対しては計算量が膨大になるので実行が難しいという問題点がある．

本研究では密度ベースのクラスタリング手法である **OPTICS** [1] に対して密度計算の不要な点に対する距離の計算を省略することによって計算量を削減した手法，**BLOCK-OPTICS** を提案する．実験では，提案手法に対して OPTICS と同じ厳密解を得た上で実行時間を削減できていることを示した．

## 2　関連研究

ここでは密度ベースクラスタリングに関する関連研究について述べる．

**DBSCAN** [2] は密度ベースクラスタリングの代表的な手法である．DBSCAN は **k-means 法** [3] などと異なり，任意の形状のクラスターを発見でき，クラスター数を指定する必要がない．またクラスターから離れた点を Noise Point としてクラスターから除外するので，外れ値に対してロバストな手法でもある．**GDBSCAN** [4] は DBSCAN を一般化した手法である．GDBSCAN は近傍，密度の概念を拡張し，空間的属性と

非空間的属性の両方に従ってクラスタリングすることができる．**OPTICS** [1] では DBSCAN の Core Point，到達可能性の概念を拡張した Core distance，Reachability distance を用いている．これによって密度の異なるクラスターも抽出することが可能になっている．**HDBSCAN** [5] では DBSCAN，OPTICS を応用して階層型のクラスターの抽出を可能にしている．

密度ベースクラスタリングのクラスター抽出手法の発展とは異なり，クラスタリングの高速化を目的とした手法も提案されている．最適化されたインデックス構造を用いた **fast DBSCAN** [6] では DBSCAN の高速化アルゴリズムを提案している．**GriDBSCAN** [7] はグリッドパーティショニングとマージを用いて DBSCAN の性能を向上させる高い並列性の利点を生かして高い性能を示した．**$\rho$-approximated DBSCAN** [8] は高次元データに対してクラスタリング結果を近似解として精度を少し犠牲にする代わりに実行時間を大幅に改善した．**AnyDBC** [9] は，データをプリミティブクラスターと呼ばれる密度結合した小さな部分集合に圧縮し，これらの結合成分に基づいてオブジェクトにラベルを付けることにより，ラベル伝播時間を短縮している．**NQDBSCAN** [10] は新しい局所近傍探索手法を提案し，それを DBSCAN へ適用することで，不要な距離計算の多くを効果的に減少させている．**BLOCK-DBSCAN** [11] は全てのオブジェクトが Core Point であるような Inner Core Block を特定し，ブロック単位で到達可能性を計算することによって密度計算の実行回数を減少させている．

## 3 前提知識

密度ベースクラスタリング手法である **OPTICS** [1] を説明する．OPTICS で使用する用語を説明してから，実際のアルゴリズムを示す．

### 3.1 Core distance

**Core distance** は DBSCAN で定義された Core Point を拡張したものである．Core distance は式 1 のように計算される．

$$core\_distance_{\epsilon,MinPts}(\boldsymbol{p}) = \begin{cases} UNDEFINED & (|N_\epsilon(\boldsymbol{p})| < MinPts) \\ distance(\boldsymbol{p}, \boldsymbol{q}_{MinPts}) \end{cases} \quad (1)$$

式 1 で，$\boldsymbol{q}_{MinPts}$ は点 $\boldsymbol{p}$ の $\epsilon$-近傍点における $MinPts$ 番目に近傍な点を示している．DBSCAN における Core Point は，定義を満たすような点に対して Core Point であるか，Core Point ではないかという二値の離散的な分類を行なっている．一方で，Core distance はこれを拡張し，定義を満たすような点に対して $MinPts$ 番目の近傍点までの距離である連続値を割り当てている．このようにすることによってより密な領域にある点は点には小さい値の Core distance 割り当て，比較的疎な領域にある点には大きな値の Core distance を割り当てることが可能になる．このようにすることによってクラスタを形成する点に対してより密なクラスタと比較的疎なクラスタを区別して両方を抽出できる．また，定義を満たさない点に関しては定義されない値として，$UNDEFINED$ を割り当てる．

### 3.2 Reachability distance

**Reachability distance** は DBSCAN の到達可能の概念を拡張した概念である．Reachability distance は式 2 のように計算される．

$$reachability\_distance_{\epsilon,MinPts}(\boldsymbol{p}, \boldsymbol{o}) = \begin{cases} UNDEFINED & (|N_\epsilon(\boldsymbol{o})| < MinPts) \\ \max(core\_distance(\boldsymbol{o}), distance(\boldsymbol{o}, \boldsymbol{p})) \end{cases} \quad (2)$$

Reachability distance も Core distance と同様に DBSCAN の到達可能性を二値で割り当てる代わりに連続値の距離を割り当てる．Reachability distance を割り当てることによってある点に対して到達可能な点の距離の小さい点から順に探索することができる．これによって Reachability distance に関してクラスタリング対象の点集合の昇順の順序を得ることができる．このような Reachability distance の順序付きの配列を OPTICS によって得ることによってアルゴリズム 4 に示すように，DBSCAN で定義されたクラスタを Reachability distance と Core distance を参照して得ることが可能になる．

### 3.3 OPTICS アルゴリズム

以上を踏まえて OPTICS アルゴリズムをアルゴリズム 1，2，3 に示す．OPTICS アルゴリズムでは DBSCAN と同様に与えられた点に対して，まだ処理を実行していない点であれば，4 行目で ExpandClusterOrder 関数を呼び出す．*OrderdList* はアルゴリズムの適用結果得られる到達可能な点が順に Reachability distance の昇順で並んでいる順序付きの配列である．

**Algorithm 1** OPTICS

**Input:** $\boldsymbol{X}$ //クラスタリング対象のベクトル集合
$\epsilon$ //距離閾値
$MinPts$ //密度閾値
*OrderdList* //順序付き配列

```
1: for i ← 1 to size(X) do
2:     x ← X[i]
3:     if not x.processed then
4:         ExpandClusterOrder(X, x, ϵ, MinPts, OrderdList)
```

アルゴリズム 2 では，与えられた点 $\boldsymbol{x}$ に対して，2 行目で $\epsilon$-近傍点を探索し，到達可能な点を順に *OrderdList* に加えていく．3 行目の CoreDistance() は与えられた点 $\boldsymbol{x}$ に対する Core distance を計算する．6 行目の PriorityQueue() は与えられた点の Reachability distance の昇順で点を保持する優先度付きキューを返す．PriorityQueue は最も Reachability distance の小さい値を持つ点を取り出す pop()，キューにまだ存在しない点を格納する push($reachability\_distance, \boldsymbol{x}$)，キューに存在する点の Reachability distance を新しく書き換えて格納する moveup($reachabilitydistance, \boldsymbol{x}$) の三つの操作を持つ．

優先度付きキュー *seeds* に $\epsilon$-近傍点を追加してゆき，順に

pop() して処理してゆく．点が Core distance の定義を満たしていれば，その点を用いて 7,16 行目で *seeds* の更新を行う．

アルゴリズム 3 では与えられた優先度付きキュー *seeds* の更新を行う．4 行目で与えられた点 $\boldsymbol{x}$ と，その $\epsilon$-近傍点 $\boldsymbol{n}$ に対して，Reachability distance を計算する．$\boldsymbol{n}$ がまだ *seeds* 内に存在しない時，7 行目で計算された Reachability distance を用いて *seeds* に push して追加する．$\boldsymbol{n}$ がすでに *seeds* 内に存在し，かつ新しく計算された Reachability distance が古い Reachability distance よりも小さい時（より近くに到達可能な点 $\boldsymbol{x}$ が発見された時），新しく計算された Reachability distance を用いて moveup し，*seeds* 内の $\boldsymbol{n}$ の位置を更新する．

**Algorithm 2** ExpandClusterOrder

**Input:** $\boldsymbol{X}$ //クラスタリング対象のベクトル集合
$\boldsymbol{x}$ //処理対象のベクトル
$\epsilon$ //距離閾値
$MinPts$ //密度閾値
*OrderdList* //順序付き配列
1: $\boldsymbol{x}$.processed() //$\boldsymbol{x}$ を処理済みにする．
2: $neighbors \leftarrow$ rangeQuery$(\boldsymbol{x}, \epsilon)$
3: $\boldsymbol{x}.core\_distance \leftarrow$ CoreDistance$(\boldsymbol{x}, neighbors, MinPts)$
4: $OrderdList$.append$(\boldsymbol{x})$
5: **if** $\boldsymbol{x}.core\_distance \neq UNDEFINED$ **then**
6: $seeds =$ PtiorityQueue()
7: **update**$(\boldsymbol{x}, neighbors, seeds)$
8: **while** not $seeds$.empty() **do**
9: $\boldsymbol{y} \leftarrow seeds$.pop()
10: **if** not $\boldsymbol{y}$.processed **then**
11: $\boldsymbol{y}$.process() //$\boldsymbol{y}$ を処理済みにする．
12: $neighbors \leftarrow$ rangeQuery$(\boldsymbol{y}, \epsilon)$
13: $\boldsymbol{y}.core\_distance \leftarrow$ CoreDistance$(\boldsymbol{y}, neighbors, MinPts)$
14: $OrderdList$.append$(\boldsymbol{y})$
15: **if** $\boldsymbol{y}.core\_distance \neq UNDEFINED$ **then**
16: **update**$(\boldsymbol{y}, neighbors, seeds)$

**Algorithm 3** update

**Input:** $\boldsymbol{x}$ //中心点
$neighbors$ //$\epsilon$-近傍点
$seeds$ //優先度付きキュー
1: **for** $i \leftarrow 1$ to size$(neighbors)$ **do**
2: $\boldsymbol{n} \leftarrow neighbors[i]$
3: **if** not $\boldsymbol{n}$.processed **then**
4: $new\_reachable\_distance \leftarrow \max(\boldsymbol{x}.core\_distance, \mathbf{dist}(\boldsymbol{x}, \boldsymbol{n}))$
5: **if** $\boldsymbol{n}.reachability\_distance == UNDEFINED$ **then**
6: $\boldsymbol{n}.reachability\_distance \leftarrow new\_reachable\_distance$
7: $seeds$.push$(new_reachable_distance, \boldsymbol{n})$
8: **else**
9: **if** $new\_reachable\_distance < \boldsymbol{n}.reachability\_distance$ **then**
10: $\boldsymbol{n}.reachability\_distance \leftarrow new\_reachable\_distance$
11: $seeds$.moveup$(new_reachable_distance, \boldsymbol{n})$

アルゴリズム 4 は，OPTICS の結果得られた順序付きの配列 *OrderdList* を用いて，DBSCAN の定義に基づいたクラスタを抽出する．クラスタリングのための距離閾値 $\epsilon'$ は，OPTICS アルゴリズムで用いた距離閾値 $\epsilon$ と異なる値であるが，$\epsilon' \leqq \epsilon$ の条件を満たす必要がある．また $UNDEFINED$ となる Core distance，Reachability distance はいかなる値よりも大きい値として扱う．

アルゴリズムでは *OrderdList* の先頭から順に Reachability distance を比較する．Reachability distance が距離閾値 $\epsilon'$ よりも大きい時，その点は先行する点からは到達不可能な点である．この点の Core distance が $\epsilon'$ 以下である時，この点はクラスタを形成する点 (DBSCAN の Core Point に相当する点) なので，6 行目で次の新しいクラスタを割り当てる．一方，Core distance が $\epsilon'$ より大きい場合，この点はクラスタを形成する点ではない（DBSCAN の Noise Point に相当する）ので，8 行目でこの点は $NOISE$ とする．Reachability distance が $\epsilon'$ 以下である時，この点は先行する点から到達可能な点であるので，先行する点と同じクラスタに割り当てる．

このように OPTICS を用いることによって，Reachability distance を基準としたクラスタごとに到達可能な点を近傍順に並べた順序付き配列を得ることができる．また，得られた順序付きの配列によって DBSCAN の定義を満たすクラスタを抽出することが可能になる．

**Algorithm 4** ExtractDBSCAN Clustering

**Input:** *OrderdList* //順序付き配列
$\epsilon'$ //クラスタリングに関する距離閾値 ($\epsilon' \leqq \epsilon$)
1: $clusterID \leftarrow NOISE$
2: **for** $i \leftarrow 1$ to size$(OrderdList)$ **do**
3: $\boldsymbol{x} \leftarrow OrderdList[i]$
4: **if** $\boldsymbol{x}.reachability\_distance > \epsilon'$ **then**
5: **if** $\boldsymbol{x}.core\_distance \leqq \epsilon'$ **then**
6: $\boldsymbol{x}.clusterID \leftarrow$ **next**$(clusterID)$
7: **else**
8: $\boldsymbol{x}.clusterID \leftarrow NOISE$
9: **else**
10: $\boldsymbol{x}.clusterID \leftarrow clusterID$

## 4 提案手法

提案手法では先行研究である，**BLOCK-DBSCAN** を用いた **OPTICS** の発展手法 **BLOCK-OPTICS** を提案する．まず Touched Inner Core Point, Inner Core Point, Inner Core Block, Outer Core Point, Noise を定義する．それらを用いた Inner Core Point の計算を除外した BLOCK-OPTICS アルゴリズムを説明する．

### 4.1 問題設定

本章のテーマは密度ベースクラスタリング手法である OPTICS の高速化である．密度ベースクラスタリングの代表的な手法である DBSCAN では各データ点に対して一定距離範囲内に存在するデータ点の個数を密度として計算し，密な領域

と疎な領域を分類してクラスタリングをする．DBSCAN では k-means 法等と異なり，クラスタの形状が球状であるという前提がなく，任意の形状のクラスタを抽出することが可能である．また，クラスタから大きく離れている外れ値をノイズして抽出するので，ノイズに対して頑健性の高いクラスタリングを行うことができる．OPTICS は同じく DBSCAN を発展させた密度ベースのクラスタリング手法である．DBSCAN ではデータセットに対して一つの距離閾値と密度閾値をハイパーパラメータとして設定する必要がある．一方で，この閾値はデータの分布にを考慮して設定することが必要であり，クラスタリングの結果に大きく影響する．また，密度のパラメータを設定することによって，データセット内の異なる密度のクラスタを抽出することが難しい，OPTICS ではこのような問題に対して，DBSCAN の Core Point の定義を拡張し，ある点から到達可能な点を距離順でソートし，計算された Core distance, Reachability distance を用いてクラスタを決定する．このようにすることによって，得られた順序付きのデータ点に対してパラメータを変更することができる．また，データセット内の異なる密度のクラスタを抽出することも可能になる．しかし，OPTICS は全てのデータ点に対して距離計算を用いた密度の計算を必要とする．そのため，大規模，高次元なデータセットに対しては計算量が膨大になるため実行が難しいという問題点がある．

このような問題点に対して，本研究では密度の計算をする必要のないデータ点に対しては計算を除外し，全てのデータ点に対して計算を行うことなくクラスタリング結果を得る BLOCK-OPTICS を提案する．

### 4.2 提案手法のアイデア

提案手法は BLOCK-DBSCAN で示された以下の定理に基づいている．

**定理.** $|N_{\frac{\epsilon}{2}}(p_i)| \geqq MinPts$ を満たす時，$\forall q \in N_{\frac{\epsilon}{2}}(p_i)$ は Core Point である．

**証明.** $\forall p_j, p_k \in N_{\frac{\epsilon}{2}}(p_i)$ の時，$d_{i,j} \leqq \frac{\epsilon}{2}$，$d_{i,k} \leqq \frac{\epsilon}{2}$ を満たす．距離に関する三角定理より，$d_{j,k} \leqq d_{i,j} + d_{i,k} \leqq \epsilon$ が得られる．これは $p_i$ の距離 $\frac{\epsilon}{2}$ 以内に存在する，どのような二点間の距離も $\epsilon$ 以下であることを意味する．従って，$\forall q \in N_{\frac{\epsilon}{2}}(p_i)$ を満たす $q$ に関して，$N_{\frac{\epsilon}{2}}(p_i) \subset N_{\frac{\epsilon}{2}}(q)$ が成り立つ．以上より $|N_{\frac{\epsilon}{2}}(p_i)| \geqq MinPts$ を満たす時，$|N_{\frac{\epsilon}{2}}(p\ \mathrm{q})| \geqq MinPts$ もまた満たすので $\forall q \in N_{\frac{\epsilon}{2}}(p_i)$ は Core Point である． □

定理より，$|N_{\frac{\epsilon}{2}}(p_i)| \geqq MinPts$ を満たすような点 $p_i$ を **Touched Inner Core Point**，$N_{\frac{\epsilon}{2}}(p_i)$ を **Inner Core Block**，Inner Core Block に含まれる点 $\forall q \in N_{\frac{\epsilon}{2}}(p_i)$ を **Inner Core Point** と呼ぶ．一方で，$|N_{\frac{\epsilon}{2}}(p_i)| \geqq MinPts$ を満たさない Core Point は **Outer Core Point** と定義する．また，DBSCAN と同様に Border Point と Noise も定義される．

また，定理を図 2 に示す．図に示すように，定理を満たすような Inner Core Blosk は Inner Core Point である点 $\boldsymbol{q}$ の $\epsilon$-近傍点の範囲に完全に包含される．定理より Inner Core Block に含まれる Inner Core Point は Core distance，Reachability distance を計算することなく Core Point であるということを判定することができる．

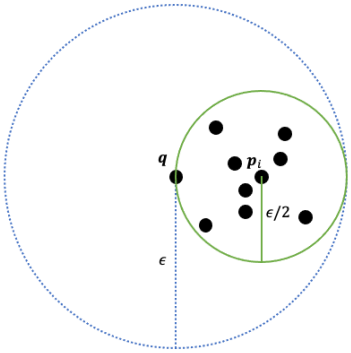



**図 2:** Inner Core Block と Inner Core Point

### 4.3 BLOCK-OPTICS アルゴリズム

BLOCK-OPTICS の ExpandClusterOrder アルゴリズムをアルゴリズム 5 に示す．

アルゴリズムの Core distance を順に計算する処理は OPTICS の ExpandClusterOrder アルゴリズムと同様である．取り出した点を順に rangeQuery によって $\epsilon$-近傍点を計算し，クラスターを形成する点であれば，クラスターを拡大するような処理を $\epsilon$-近傍点に対しても行う．

7,21 行目の条件分岐では対象の点が Touched Inner Core Point であることを示している．その場合，8,22 行目で $\epsilon$-近傍点を *inner_core_block* と，*not_inner_core_block* に分割する．*inner_core_block* は定理より，Core distance を計算しなくとも既に DBSCAN における Core Point のようなクラスターを形成する点であることが確定しているので，9,23 行目で Inner Core Point を全て処理済みとする．process(*inner_core_block*) は与えられた *inner_core_block* を全て処理済みとする関数である．また $\epsilon$-近傍点ではあるが *inner_core_block* ではない *not_inner_core_block* については，10,24 行目でアルゴリズム 3 の update 関数を用いて優先度付きキュー *seeds* の更新を行う．

一方で，12,26 行目では対象の点が Touched Inner Core Point ではなく，Outer Core だった場合である．このような場合は OPTICS と同様に $\epsilon$-近傍点全てを用いて *seeds* の更新処理を行う．以上のように対象の点が Touched Inner Core Point である場合には，その点の Inner Core Block に属する Inner Core Point の Core distance，Reachability distance の計算を削減できていることがわかる．

**Algorithm 5** BLOCK-OPTICS:ExpandClusterOrder

**Input:** $\boldsymbol{X}$ //クラスタリング対象のベクトル集合
$\boldsymbol{x}$ //処理対象のベクトル
$\epsilon$ //距離閾値
$MinPts$ //密度閾値
$OrderdList$ //順序付き配列

1: $\boldsymbol{x}$.processed() //$\boldsymbol{x}$ を処理済みにする．
2: $neighbors \leftarrow$ rangeQuery$(\boldsymbol{x}, \epsilon)$
3: $\boldsymbol{x}.core\_distance \leftarrow$ CoreDistance$(\boldsymbol{x}, neighbors, MinPts)$
4: $OrderdList$.append$(\boldsymbol{x})$
5: **if** $\boldsymbol{x}.core\_distance != UNDEFINED$ **then**
6: $seeds =$ PtiorityQueue()
7: **if** $\boldsymbol{p}.core\_distance \leqq \frac{\epsilon}{2}$ //$\boldsymbol{p}$ が Touched Inner Core Point の場合 **then**
8: $inner\_core\_block, not\_inner\_core\_block$ ←devide_block$(neighbors)$
9: process$(inner_core_block)$
10: **update**$(\boldsymbol{x}, not\_inner\_core\_block, seeds)$
11: **else**
12: **update**$(\boldsymbol{x}, neighbors, seeds)$
13: **while** not $seeds$.empty() **do**
14: $\boldsymbol{y} \leftarrow seeds$.pop()
15: **if** not $\boldsymbol{y}$.processed **then**
16: $\boldsymbol{y}$.process() //$\boldsymbol{y}$ を処理済みにする．
17: $neighbors \leftarrow$ rangeQuery$(\boldsymbol{y}, \epsilon)$
18: $\boldsymbol{y}.core\_distance \leftarrow$ CoreDistance$(\boldsymbol{y}, neighbors, MinPts)$
19: $OrderdList$.append$(\boldsymbol{y})$
20: **if** $\boldsymbol{y}.core\_distance != UNDEFINED$ **then**
21: **if** $\boldsymbol{p}.core\_distance \leqq \frac{\epsilon}{2}$ //$\boldsymbol{q}$ が Touched Inner Core Point の場合 **then**
22: $inner\_core\_block, not\_inner\_core\_block$ ←devide_block$(neighbors)$
23: process$(inner_core_block)$
24: **update**$(\boldsymbol{y}, not\_inner\_core\_block, seeds)$
25: **else**
26: **update**$(\boldsymbol{y}, neighbors, seeds)$

### 4.4 Inner Core Block のマージ

BLOCK-OPTICS アルゴリズムによって OPTICS で得られる順序付きの配列を得られる．しかし，抽出した点の中で Inner Core Block は Reachability distance を計算していないので，各ブロックが到達可能かどうかは不明である．そのため，BLOCK-DBSCAN と同様に Inner Core Blosk 同士のマージ処理が必要になる．

抽出された Inner Core Block の集合を $icbs$ とし，$\exists(icb_1, icb_2) \in icbs$ に対して，$core\_distance(icb_1) \leqq \epsilon, core\_distance(icb_2) \leqq \epsilon$ を満たすときに，二つのブロック間の最小距離によって以下の三つの場合分けが可能である．ここで $tic_1, tic_2$ は対応するブロックの Touched Inner Core Point を示す．

（1） $dist(tic_1, tic_2) \leqq \epsilon$
（2） $\epsilon < dist(tic_1, tic_2) < 2\epsilon$
（3） $2\epsilon \leqq dist(tic_1, tic_2)$

1つ目の場合，二つのブロックは必ず到達可能なので，マージする．2つ目の場合，二つのブロックは到達可能であるかは判別できない．そのため各ブロックの Inner Core Point が到達可能であるか，計算する必要がある．3つ目の場合，二つのブロックは到達可能ではない．そのため，マージ処理は行わない．

以上の条件分岐を用いた Inner Core Block のマージ処理をアルゴリズム 6 に示す．3 行目では二つの Inner Core Block に対応する Touched Inner Core Point 間の距離を計算している．4-6 行目は 1 つ目の場合で，二つの Inner Core Block は到達可能なので，マージする．7-9 行目は 2 つ目の場合で，density_reachable 関数を用いて到達可能であれば，マージする．3つ目の場合では到達可能ではないのでマージ処理は行わない．5,8 行目のマージ処理では $icbs$ の中で，先行する Inner Core Block の $icbs[i]$ へと $icbs[j]$ をマージする．

**Algorithm 6** handle inner core

**Input:** $\boldsymbol{icbs}$ //Inner Core Blocks
$\epsilon$ //距離閾値

1: **for** $i = 1$ to size$(\boldsymbol{icbs})$ **do**
2: **for** $j = i + 1$ to size$(\boldsymbol{icbs})$ **do**
3: $dist \leftarrow dist(icbs[i], icbs[j])$
4: **if** $dist \leqq \epsilon$ **then**
5: merge $icbs[j]$ to $icbs[i]$
6: remove $icbs[j]$ from $icbs$
7: **else if** $\epsilon < dist < 2\epsilon$ & **density_reachable**$(icbs[i], icbs[j])$ **then**
8: merge $icbs[j]$ to $icbs[i]$
9: remove $icbs[j]$ from $icbs$

density_reachable 関数をアルゴリズム 7 に示す．ここでは与えられた二つの Inner Core Block である $\boldsymbol{icb}_1, \boldsymbol{icb}_2$ が到達可能であるか判定する．アルゴリズムは BLOCK-DBSCAN で証明されている，ブロック間の近似距離を計算する手法と同じ手法を用いる．

**Algorithm 7** density_reachable

**Input:** $\boldsymbol{icb}_1$ //Inner Core Block 1
$\boldsymbol{icb}_2$ //Inner Core Block 2
$\epsilon$ //距離閾値
$rnt$ //繰り返し回数
**Output:** $result$ //$\boldsymbol{icb}_1$ と $\boldsymbol{icb}_2$ が到達可能かの真偽値

1: $result \leftarrow false$
2: **for** $i = 1$ to $rnt$ **do**
3: $seed_1 \leftarrow tic$ in $\boldsymbol{icb}_1$
4: **while** not converged **do**
5: $seed_2 \leftarrow \arg\min_{p \in \boldsymbol{icb}_2}(p, seed_1)$
6: $seed_1 \leftarrow \arg\min_{p \in \boldsymbol{icb}_1}(p, seed_2)$
7: $app\_dist \leftarrow dist(seed_1, seed_2)$
8: **if** $app\_dist < \epsilon$ **then**
9: $result \leftarrow true$
10: **return** $result$
11: **return** $result$

## 5 実　　験

### 5.1 データセット

合成データセットとして図 3 に示すようなデータを用いた (図 3 では次元数 2，データ数 1,000)．以下の実験ではこの合成データセットの次元数，データ数を変化させて実験する．

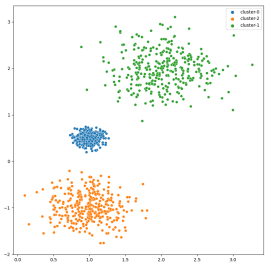


**図 3:** 合成データセットの散布図

また実世界データセットとしては MoCap データセット [12]，APS データセット [13] を用いた．実世界データセットでは前処理を行なっている．まず，データセット内の欠損値については 0 補間を行なっている．MoCap データセットでは'User'，'Class' 属性，APS データセットでは'class' 属性を削除している．さらにそれぞれのデータセットの各属性値は最小値 0，最大値 100,000 となるように正規化を行なっている．MoCap データセットのデータ数は 78,095，次元数は 36 である．APS データセットのデータ数は 76,000，次元数は 170 である．

### 5.2 パラメータ

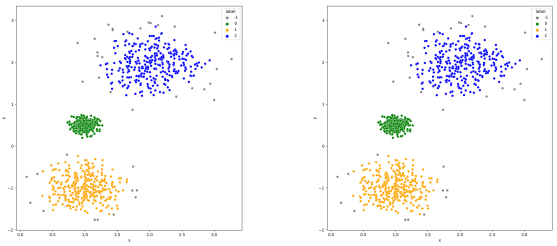
今回の実験では先行研究の OPTICS と提案手法である BLOCK-OPTICS を比較する．OPTICS と BLOCK-OPTICS はどちらもハイパーパラメータである，距離閾値 $\epsilon$ と密度閾値である $MinPts$ を設定する必要がある．合成データセットでは，$(\epsilon, MinPts) = (0.2, 10)$ として設定した．

実世界データセットでも同様に $\epsilon$ と $MinPts$ を設定した．MOCAP データセットでは $(\epsilon, MinPts) = \{(70000, 5), (70000, 10)\}$，APS データセットでは $(\epsilon, MinPts) = \{(75000, 5), (75000, 10)\}$ として設定した．

### 5.3 合成データセットのクラスタリング実験

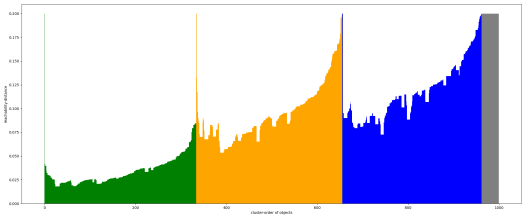
合成データセットに対する OPTICS と BLOCK-OPTICS のクラスタリング結果を図 4 に示す．二つの散布図からわかるように二つのアルゴリズムでは同一のクラスタリング結果が得られていることがわかる．

また二つの手法で得られる順序付き配列の Reachability distance を図 5 に示す．OPTICS では全ての点に対して Reachability distance を計算しているので，配列の長さはデータ数と等しく 1,000 となっている．一方で，BLOCK-OPTICS では探索中に特定した Inner Core Block に含まれる Inner Core Point に対しては Core distance，Reachability distance を求

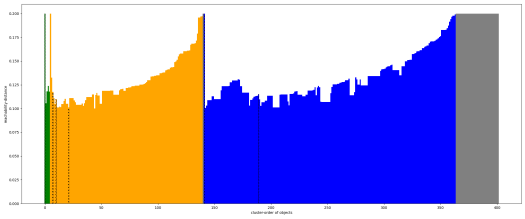


**(a)** OPTICS　　**(b)** BLOCK-OPTICS

**図 4:** 各手法の合成データセットに対するクラスタリング結果



**(a)** OPTICS



**(b)** BLOCK-OPTICS

**図 5:** 各手法の Reachability distance

めるための計算を行わない．そのため配列の長さは Inner Core Point の数を除いた長さになっている．図中の網掛けされている部分は，Touched Inner Core Point の Reachability distance を示している．実際に得られた Reachability distance を先頭から順に走査し，クラスタリングを実行する際には，対象の点が Touched Inner Core Point であれば，その点の Inner Core Block（既にマージ済み）に属する Inner Core Point は同じクラスタとしてクラスタリングするようにしている．

BLOCK-OPTICS によって得られたクラスタを Touched Inner Core Point (TIC)，Inner Core Point (IC)，Outer Core Point (OC)，Border Point (BORDER)，Noise Point (NOISE) の点の分類ごとの散布図を図 6 に示す．図に示されているようによりクラスタの中心に位置する密な領域には TIC が分布している．クラスタの外側のより疎な領域になるにつれて IC，OC，BORDER，NOISE が順に分布している．このようにクラスタ中心のようなより密な領域に存在する多数の IC は距離の計算を省略できていることがわかる．

### 5.4 次元数とデータ数の違いによる比較実験

次元数の違いによる OPTICS と BLOCK-OPTICS の実行時間の比較結果を図 7 に示す．次元数を増加させた場合に実行

**表 1:** 実世界データセットに対する実行時間

| データセット | 手法 | $\epsilon$ | $MinPts$ | ExpandClusterOrder 実行時間 [s] | Merge 実行時間 [s] |
|---|---|---|---|---|---|
| MoCap | OPTICS | 70000 | 5 | 13401.65 | - |
| | BLOCK-OPTICS | 70000 | 5 | 3833.03 | 70.32 |
| | OPTICS | 70000 | 10 | 14972.03 | - |
| | BLOCK-OPTICS | 70000 | 10 | 3055.32 | 24.7528 |
| APS | OPTICS | 75000 | 5 | 201547.55 | - |
| | BLOCK-OPTICS | 75000 | 5 | 312.46 | 3317.38 |
| | OPTICS | 75000 | 10 | 207633.93 | - |
| | BLOCK-OPTICS | 75000 | 10 | 363.45 | 2931.78 |

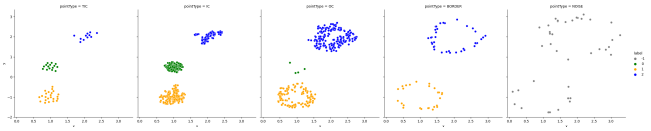


**図 6:** BLOCK-OPTICS の点の種類別の散布図

時間の大きな変化は見られなかった．しかし，いずれの次元数においても提案手法の BLOCK-OPTICS は OPTICS と比較して 6 倍程度の高速化を達成できている．

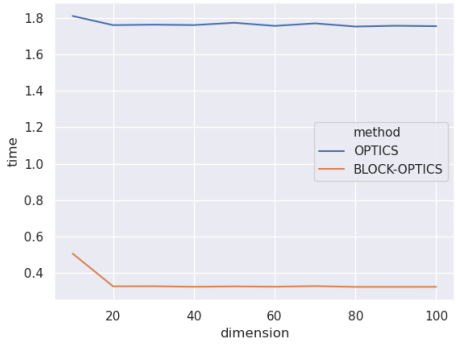



**図 7:** 次元数を変化させた時の実行時間

データ数の違いによる OPTICS と BLOCK-OPTICS の実行時間の比較結果を図 8 に示す．OPTICS はデータ数の増加によって OPTICSOPTICS はデータ数の増加に伴って大きく実行時間も増加している．一方で，BLOCK-DBSCAN ではあまり変化しなかった．これは Inner Core Point が大幅に増加したため，実際に計算される点の数はあまり増加していないためだと考えられる．

### 5.5 実世界データセットを用いた実験

二つの実世界データを用いた実行時間の比較結果を表 1 に示す．提案手法 BLOCK-OPTICS は OPTICS よりも大幅に実行時間を削減できている．特に APS データセットでは大きな差が見られた．図 9 に各データセットに対する BLOCK-DBSCAN の特定された TIC, IC, OC, BORDER, NOISE の割合を示している．MOCAP データセットは $(\epsilon, MinPts) = (70000, 5)$, APS データセットは $(\epsilon, MinPts) = (75000, 5)$ である．MOCAP データセットでは計算が省略される IC は約 6 割程度であ

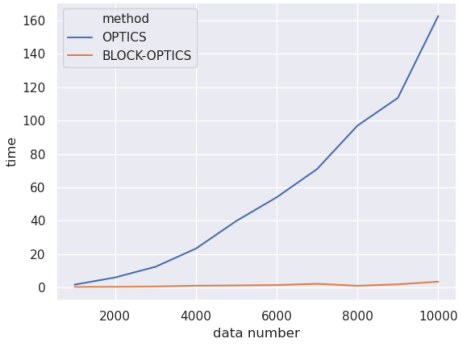
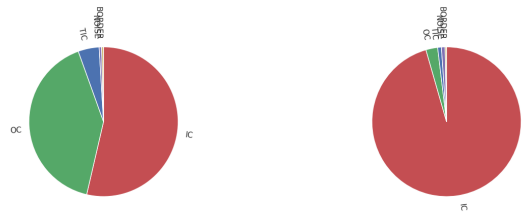



**図 8:** データ数を変化させた時の実行時間

る．一方で，APS データセットでは 9 割以上が IC として抽出されている．これは APS データセットが非常に密なクラスタを形成しているという特徴を持っているためと考えられる．そのため，提案手法の BLOCK-OPTICS は密なクラスタを持つようなデータセットに対しては効率的にクラスタリングを実行できると考えられる．

一方で，マージ処理の実行時間の割合は MOCAP データセットの方が小さい．これは APS データセットの場合と比較して，MOCAP データセットでは Touched Inner Core Block の検出数が少ないためである．Touched Inner Core が少ないため，そのマージ処理を行う回数も少なくなる．結果として，実行時間全体に対するマージ処理の実行時間の割合も小さくなる．



**(a)** MoCap データセット　　**(b)** APS データセット

**図 9:** BLOCK-OPTICS で抽出された点の種別割合

## 6 まとめと今後の課題

本研究では密度ベースクラスタリング OPTICS の高速化を

達成した．提案手法の BLOCK-OPTICS では定理に従って，距離計算をせずとも Core Point であることが判明する Inner Core Block を特定することによって，全ての点の距離計算を行うことなくクラスタリングを実行できる．また，実験によって合成データセット，実世界データセットを用いて提案手法は OPTICS と比較して高速にクラスタリングが実行できることを示した．

今後の研究課題としては HDBSCAN などを用いて階層型クラスタリングができるような手法に対するクラスタリングの高速化を行いたい．

## 謝　　辞

本研究は，SKY(株)（CPI03114）による共同研究経費の助成を受けたものです．また，この成果は国立研究開発法人新エネルギー・産業技術総合開発機構 (NEDO) の委託業務 (JPNP20006) の結果得られたものです．

## 文　　献

[1] Mihael Ankerst, Markus M Breunig, Hans-Peter Kriegel, and Jörg Sander. Optics: Ordering points to identify the clustering structure. *ACM Sigmod record*, Vol. 28, No. 2, pp. 49–60, 1999.

[2] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, Vol. 96, pp. 226–231, 1996.

[3] John A Hartigan and Manchek A Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the royal statistical society. series c (applied statistics)*, Vol. 28, No. 1, pp. 100–108, 1979.

[4] Jörg Sander, Martin Ester, Hans-Peter Kriegel, and Xiaowei Xu. Density-based clustering in spatial databases: The algorithm gdbscan and its applications. *Data mining and knowledge discovery*, Vol. 2, No. 2, pp. 169–194, 1998.

[5] Leland McInnes, John Healy, and Steve Astels. hdbscan: Hierarchical density based clustering. *Journal of Open Source Software*, Vol. 2, No. 11, p. 205, 2017.

[6] Ade Gunawan and M de Berg. A faster algorithm for dbscan. Master ' s thesis, 2013.

[7] Shaaban Mahran and Khaled Mahar. Using grid for accelerating density-based clustering. In *2008 8th IEEE International Conference on Computer and Information Technology*, pp. 35–40. IEEE, 2008.

[8] Junhao Gan and Yufei Tao. Dbscan revisited: Mis-claim, un-fixability, and approximation. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data*, pp. 519–530, 2015.

[9] Son T Mai, Ira Assent, and Martin Storgaard. Anydbc: An efficient anytime density-based clustering algorithm for very large complex datasets. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1025–1034, 2016.

[10] Yewang Chen, Shengyu Tang, Nizar Bouguila, Cheng Wang, Jixiang Du, and HaiLin Li. A fast clustering algorithm based on pruning unnecessary distance computations in dbscan for high-dimensional data. *Pattern Recognition*, Vol. 83, pp. 375–387, 2018.

[11] Yewang Chen, Lida Zhou, Nizar Bouguila, Cheng Wang, Yi Chen, and Jixiang Du. Block-dbscan: Fast clustering for large scale data. *Pattern Recognition*, Vol. 109, p. 107624, 2021.

[12] Mocap hand postures data set. https://archive.ics.uci.edu/ml/datasets/MoCap+Hand+Postures.

[13] Aps failure at scania trucks data set. https://archive.ics.uci.edu/ml/datasets/APS+Failure+at+Scania+Trucks.