大規模データストリームに対する高速なS-FINCHクラスタリング

牛尼　索造†　　藤原　靖宏††　　塩川　浩昭†††

† 筑波大学 情報学群 情報科学類　〒 305–8577 茨城県つくば市天王台 1-1-1
†† NTT コミュニケーション科学基礎研究所　〒 243-0198 神奈川県厚木市森の里若宮 3-1
††† 筑波大学 計算科学研究センター　〒 305–8577 茨城県つくば市天王台 1-1-1
E-mail: †sushiama@kde.cs.tsukuba.ac.jp, ††yasuhiro.fujiwara@ntt.com, †††shiokawa@cs.tsukuba.ac.jp

あらまし　クラスタリングはデータ集合に内在する性質の類似した部分集合を検出する手法であり，データ分析において重要な要素技術となっている．その中でも近年注目を集めているのは，データストリームに対して凝集型階層的クラスタリングを行う S-FINCH である．S-FINCH は時々刻々と到来するデータストリームに対して各データオブジェクトの最近傍や共有最近傍を求める必要があるため，各データオブジェクト間の距離計算が必要となる．そのため，S-FINCH はクラスタリングにおいて膨大な計算時間を必要とする．そこで本稿では S-FINCH の高速化手法を提案する．提案手法は空間索引を利用することで，S-FINCH において生じる距離計算回数を削減する．本稿では実データを用いた評価実験により，提案手法は S-FINCH に対してクラスタリング精度を損なわずに効率的なクラスタリング処理が実行可能であることを示した．

キーワード　ストリームデータ処理，空間・時空間・時系列データ処理，データ構造・索引

1　はじめに

多次元データストリームに対するクラスタリングとは膨大なデータから性質の類似するデータごとのグループ分割を提供する手法である．多くの科学の分野において，膨大なデータから性質の類似するデータのグループを見つけだしラベル付けをすることは非常に重要になっている．特に近年では，取り扱われるデータのサンプル数や次元数がますます増加し，このようなデータに対する高速な解析処理技術への需要が高まっている．膨大なデータから性質の類似するデータのグループ分割を提供する手法のひとつとして，クラスタリングが挙げられる．

近年 Sarfraz らによって FINCH [1] というクラスタリング手法が提案された．FINCH は多次元のデータポイントの集合を入力として受け取り，その各データポイントの間の距離を計算し，最も近くにあるデータポイントとの間，または同じデータポイントを最も近いとするデータポイント間にエッジを張ることで得られるグラフの各連結成分を 1 つのグループとする．この各グループ内の各データポイントの平均を代表点とし，次の入力として同じ処理を繰り返す．これにより，異なる粒度のグループ分けを持つクラスタ階層を出力する．FINCH は従来の k-means 法 [2] などのクラスタリング手法と比較して，1.) 高い精度で真のクラスタを自動的に決定できる，2.) ハイパーパラメータを必要としない，3.) 異なるドメインのデータでも一般的に利用できる，4.) 膨大なデータセットに拡張できるなどのメリットがあり，高次元のデータセットに対してもハイパーパラメータを必要とせずに高速なクラスタリングが実現可能であることから，FINCH は画像や化学などの幅広い分野での応用 [3–5] に挑戦されている．

しかし，FINCH は静的なデータを対象としたアルゴリズムであるため，データストリームを処理しようとした場合，データが追加されるとデータ全体に対して最近傍探索を行いクラスタ構造を再構築する必要がある．そのため，FINCH はデータストリームのクラスタリングに膨大な処理時間を必要とする．この問題に対処するために Cunningham によって S-FINCH [6] が提案された．S-FINCH は，データが追加された時にクラスタ構造を全て再構築するのではなく，追加されたデータによって変更が生じた部分のみを対象として再計算することによって処理時間を短縮する．

しかしながら，S-FINCH は FINCH に比べて高速なストリーム処理が可能ではあるものの，大規模なストリームデータを対象としたクラスリングは困難である．S-FINCH では，追加されたデータと全ての頂点間に対して距離を計算する．そのため，n 件目のデータが追加された時に，距離計算のみで $\mathcal{O}(n)$ の時間計算量が生じる．また，S-FINCH は階層的クラスタリング手法であるため，データの追加によって他の階層のクラスタ構造も変化する．この階層は高々 $\log_2 n$ であることから，この処理は $\mathcal{O}(n \log n)$ の時間計算量を要する．N 件のストリームデータを処理するのに $\mathcal{O}\left(N^2 \log N\right)$ を要する．したがって，大規模なストリームデータに対しては膨大な処理時間を要することになる．

1.1　本研究の貢献

本稿では S-FINCH の高速化手法を提案する．提案手法では従来手法 S-FINCH の計算コストを削減するために，FINCH が提供するクラスタ構造が最近傍連結成分という小規模な集合によって構成されていることに着目する．また，FINCH で生成されたクラスタは他の最近傍クラスタを持ち最近傍関係にある連結成分が上の階層でのクラスタとなるため，各クラスタの関係を木構造として表現出来る．最近傍関係にある小規模な集

合を，特別に構築することなく得られるため，この集合による空間を空間索引として活用することで，空間索引の構築コストを削減できると考えられる．

そこで本稿では各クラスタに対し各データを包含する空間を求め，最近傍や逆最近傍を求めたいデータポイントとその空間との最短距離を各クラスタの各データの点との距離の下限とし，その下限を利用して最近傍・逆最近傍にならないデータポイントを特定することで距離計算回数を削減する．その結果として提案手法は以下の特性を示す．

計算回数

S-FINCH に比べ少ない計算回数で最近傍および被最近傍探索を行うことができる．

正確性

提案手法で用いるアプローチは，厳密な最近傍を求めるため FINCH が提供するクラスタ構造から精度を損わない．

本稿の構成は，次の通りである．2 節で本稿の前提となる知識について概説する．3 節にて提案手法の詳細について説明し，4 節において提案手法の評価と分析を行う．5 節にて，本稿をまとめ，今後の課題について論ずる．

2 事 前 準 備

この節では FINCH [1] とストリーム処理を可能にした S-FINCH [6] について説明する．表 1 に主な記号とその定義を示す．

FINCH [7] は，d 次元のデータポイントが N 件ある多次元データセット $\boldsymbol{S} \in \mathbb{R}^{N\times d}$ を入力として受け取る．これに対して，S-FINCH は入力として d 次元のデータポイントが N 件連続している多次元データストリームを $\boldsymbol{L} = [L_1, L_2, \ldots, L_N]$（ただし，$L_i \in \mathbb{R}^d$）を受け取る．距離尺度として，FINCH および S-FINCH はユークリッド距離を用いる．すなわち，2 つのデータポイントのユークリッド距離が小さいとき，それらは類似したデータポイントであることを意味する．多次元データポイント v, w があるとき，これらの距離を $dist(v, w)$ を表記する．加えて，多次元空間 X があるとき，v から X までの距離の最小値を $min\_dist(v, X)$ を表記する．データポイント x の最近傍とは，x に最も近いデータポイント y のことを指し $NN(x) = y$ と表記するこれに対して，データポイント x の被最近傍とは，x が最近傍であるデータポイント集合 $NNset(x)$ のことを指し $NNset(x) = \{y \mid NN(y) = x\}$ 定義する．

FINCH [7] および FINCH はクラスタリング結果として階層的なクラスタ構造を出力する．そしてこのクラスタ構造の階層の数，つまり階層の高さを H とする．本論文ではこのクラスタ構造の一番下の階層を高さ 1 と，一番上の階層を高さ H とする．それぞれの手法に入力として与えられたデータの各データポイントはまず高さ 1 の階層でクラスタを構成することになる．ここで入力として与えられた各データポイントは 1 つのデータポイントからなるクラスタとして扱う．これにより，高さ 1 の階層ではクラスタ構造の全てのデータポイントは全てクラスタの代表点となる．最上層ではない階層においてデータポイント x がクラスタ X_3 に属するとき，クラスタ X_3 のことを x の親と呼び，$parent(x) = X_3$ と表記する．また，x はクラスタ X_3 の子となり，$x \in children(X_3)$ と表記する．データポイント x の子となるデータポイント $children(x)$ や，さらに $children(x)$ の子となるデータポイント $children(children(x))$ 全てを子孫データポイントと呼ぶ．つまり，データポイント x の子孫となるデータポイントの集合を $descendants(x)$ とするとき，$children(x) \in descendants(descendants)$ であり，さらに $descendants(children(x)) \in descendants(x)$ となる．データポイント x の親となるデータポイント $children(x)$ や，さらに $children(x)$ の親となるデータポイント $children(children(x))$ 全てを祖先データポイントと呼ぶ．つまり，データポイント x の祖先となるデータポイントの集合を $ancestor(x)$ とするとき，$children(x) \in ancestor(ancestor)$ であり，さらに $ancestor(children(x)) \in ancestor(x)$ となる．

表 1: 記号と定義

記号	定義
N	データポイントの数
n	あるステップに与えられたデータポイントの数
d	データポイント x の次元数
κ_i^1	データポイント i の 最近傍データポイント
$L(x)$	データポイント x のクラスタラベル
$parent(x)$	高さ i のデータポイント x のクラスタの 高さ $i+1$ での代表データポイント
$children(x)$	$\{y \mid parent(y) = x\}$
$dist(v, w)$	データポイント v, w の距離
$min\_dist(v, W)$	データポイント v と空間 w の距離の最小値
$NN(x)$	x の最近傍データポイント
$NNset(x)$	x が最近傍であるデータポイント
$V_{h,sh,x}$	高さ h のデータポイント x の子孫のうち， 高さ sh にあるデータポイントを包含する空間
$W_{h,x}$	高さ h のデータポイント x を中心とし， x の最近傍までの距離を半径とする超球
$S_{h,sh,x}$	高さ h のデータポイント x の子孫のうち， 高さ sh にあるデータポイント y の $W_{h,y}$ を包含する空間

2.1 FINCH

FINCH は， Sarfraz らによって提案された出力するクラスタ数やハイパーパラメータの指定を必要とせず，データポイントの近傍関係を利用して凝集型階層的クラスタリングを行う手法である．FINCH は与えられたデータセットからグラフを生成し，各連結成分を 1 つのクラスタとすることを繰り返しクラスタ階層を生成する．生成されたクラスタの数が 1 ではないとき，クラスタの構成要素となる各データポイントの平均，つまり重心を代表点とする．この各クラスタの代表点の集合を次の処理でのデータセットとする．与えられたデータセットをクラスタリングすることを 1 つのステップとし，ステップを繰り返すことで階層的なクラスタ構造を生成する．

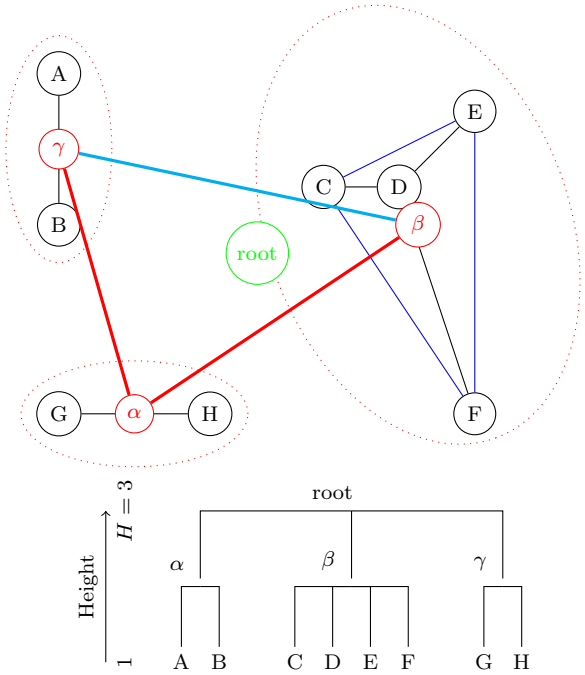

図 1: FINCH でクラスタを生成するときのグラフと対応するデンドログラム

次に連結成分を求めるために生成するグラフについて説明する．データセットの各データポイント間の距離を計算し，最近傍を求め，式 (1) で求まる隣接行列からグラフを生成する．

$$A(i,j) = \begin{cases} 1 \ if \ j = NN(i) \quad or \ j = NN(j) \\ \qquad\qquad\qquad\quad or \ NN(i) = NN(j) \\ 0 \ Otherwise \end{cases} \tag{1}$$

入力例とグラフの例を図 1 に示す．A から H は FINCH に入力として与えられる各データポイント，α から β は次のステップにおいて入力として与えられる各クラスタの構成要素の平均データポイントである．C, E, F を互いに結ぶ青い線，及び α, β を結ぶ水色の線は最近傍関係にはないものの同じデータポイントを最近傍に持つデータポイントの間に張られるエッジを示している．これは式 (1) において $NN(i) = NN(j)$ となる場合に該当する．

FINCH の時間計算量を解析するために，各ステップでの時間計算量を求める．あるステップの入力として n 件のデータポイントがある場合，最近傍探索に $\mathcal{O}(n^2 d)$，連結成分の探索に $\mathcal{O}(n)$ の時間計算量が必要となり，1 ステップに $\mathcal{O}(n^2 d)$ の時間計算量を要する．各データポイントはそれぞれの最近傍と同じの連結成分に含まれ 1 つのクラスタとなるため，ステップごとに入力となるデータセットのサイズは半分以下になる．そのため，FINCH に与えられた N 件の入力データセットに対して階層の数は高々 $\lceil \log_2 N \rceil$ となり，全体としての時間計算量は $\mathcal{O}(N^2 d \log N)$ となる．

2.2 S-FINCH

FINCH における逐次的なデータ更新を行うストリーム処理を達成することを目的に，近年 Cunningham によって S-FINCH が提案された．FINCH はデータの追加に対応できるクラスタリング手法ではないため FINCH を愚直にストリーム処理に対応させると，データが追加される度に過去のデータも含め全体のデータセットに対して FINCH を実行する必要があり $\mathcal{O}(N^3 d \log N)$ の時間計算量を要する．そこで S-FINCH では，データが追加される前のクラスタ構造を用いることで時間計算量を大きく改善する．

S-FINCH は 3 つのステージで動作する．新しいデータポイント x が追加されたとする．x の最近傍データポイントを q, x が追加されたことにより x が最近傍になったデータポイント，つまり x にとっての逆最近傍データポイント集合を S とする．ステージ 1 で x に対する，q, S を求める．q, S に合わせてグラフも更新する．ステージ 2 で x のクラスタラベル L_x を更新する．ステージ 3 で連結成分になんらかの更新があったクラスタの代表点を更新し，次の階層のクラスタの更新を行う．この 3 つのステージを下の階層から順に実行することで新しく追加されたデータポイントも含めた FINCH と同じクラスタ階層が得られる．

ここで，ステージ 1 では x と直前までに追加された各データポイントとの距離を計算する必要があるため，x が追加される前までの各データポイントの個数を $n-1$ とすると $\mathcal{O}(nd)$ の時間計算量を要する．ステージ 2 では q, S の最近傍の更新により連結成分が更新されたクラスタを高々$n-1$ 個の探索で求めることができるため，$\mathcal{O}(n)$ の時間計算量を要する．ステージ 3 ではステージ 2 で探索したクラスタのラベルを保持しておくことで解決される．また前述の通り階層構造の高さは高々 $\log_2 n$ なので，1 つのデータ追加に $\mathcal{O}(nd \log n)$ の時間計算量を要する．この手順を図 2 に示した．これにより S-FINCH は $\mathcal{O}(N^2 d \log N)$ の時間計算量を要することになり，FINCH で愚直にストリーム処理させることに比べ 時間計算量を改善できる．しかし，S-FINCH は各データの追加やクラスタが変更後の更新のときに既存の全各データポイントとの距離を計算する必要がある．これにより追加されたデータ量が増えるにつれクラスタリングに膨大な時間を要することが問題となっている．

3 提 案 手 法

本節では提案手法について概説する．提案手法は，S-FINCH のクラスタリング精度を損なわずにクラスタリング結果をより高速に計算することを目的とする．まず提案手法の基本的なアイデアについて述べ，その詳細について 3. 2 節以降で説明する．

3.1 基本アイデア

S-FINCH では新しいデータが追加された時に最近傍と逆最近傍を探索するために，これまでに追加された全てのデータポイントとの距離を計算しており，時間計算量の多くを占める．ゆえに，クラスタリングにかかる時間を短縮するためには距離

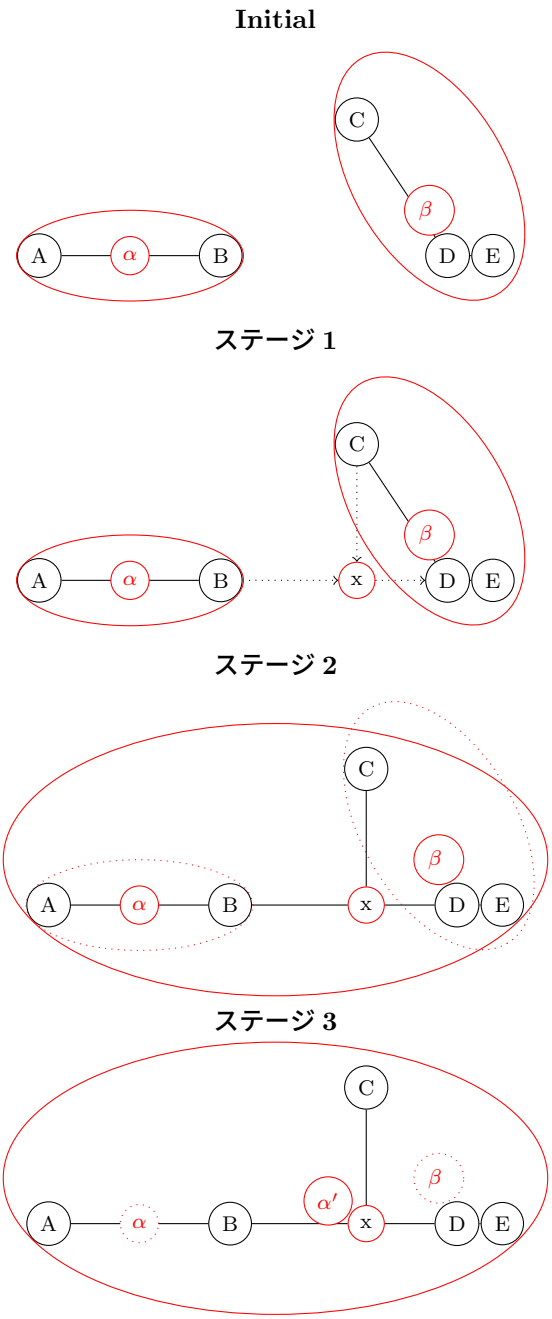

図 2: S-FINCH の詳細

計算回数を減らすことが重要である．そこで本研究ではこの最近傍と被最近傍をより少ない距離計算回数で求めることを提案手法の目的とする．

提案手法の基本アイデアは前述した距離計算において空間索引を用い，距離計算が不要なデータポイントを簡単に特定することで，全体の距離計算コストを削減することである．まず，空間索引を構築する (3.2 節)．続いて空間索引を用いた最近傍探索と逆最近傍探索で距離計算コストを下げる (3.3 節)．また，構築した空間索引はストリーム処理の各データ追加に伴い最新の状態に更新され続ける．

具体的にはデータポイント x の最近傍を探索するにあたってデータポイント a,b を包含する空間 V をおく．$dist(a,x), dist(b,x) \geqq min(dist(x,V))$ であることから，V との最小距離を計算することで V が包含する各データポイントの距離の下限を求められる．これにより最近傍を計算するにあたりデータポイント x との距離が最小距離以下であるデータポイントを既に求めている場合，データポイント x と V が包含するデータポイントの距離計算を省略できることがある．

データポイント x の逆最近傍を探索するにあたってデータポイント a を中心，a の最近傍への距離を半径とする超球 W をおく．$x \in W$ であることと a の最近傍が x になることは必要十分である．また，データポイント a,b の超球 W_a, W_b を包含する空間 S があるとする．$x \notin S$ ならば，$a,b \notin S$ である．これを利用することで，データポイント x が空間 S に含まれないとき空間 S 内の各データポイントは逆最近傍にはならないことがわかり，各データポイントとの距離計算を削減できる．

提案手法は，S-FINCH と比較して，クラスタリング精度を損なわずに大規模なデータに対して高速にクラスタリングが可能であるという優位性を持つ．これは，データが大規模になるにつれ探索において削減した距離計算コストが空間索引を構築するコストを大きく上回るようになる点によるものである．

3.2 空間索引構築

3.1 節 で述べた空間索引は，空間との最小距離を求めることと空間への包含の有無を判定する必要がある．これを実現するための空間として中心 p 半径 r の超球 S を採用する．超球を用いることでデータポイント x と空間との最小距離は $min\_dist(x,S) = max(dist(x,p) - r, 0)$ で求められ，S が x を内包するかの判定は $dist(x,p) \leqq r$ であるかどうかを確認することで実行できる．中心との距離計算は $\mathcal{O}(d)$ の時間計算量で求められるため，空間との最小距離，包含の有無いずれも $\mathcal{O}(d)$ の時間計算量しか必要としない．超矩形など異なる空間を表現するのに比べ時間計算量が小さく，効率的に空間索引の構築・利用ができる．1 つの空間索引を構築するための時間計算量と距離計算の時間計算量が同じ程度であるため，空間索引を 1 度用いるごとに 2 つ以上の頂点との距離計算が不要である判定を行うことが出来た場合に空間索引によって距離計算コストを削減出来る．また，最近傍探索と逆最近傍探索は同じ空間索引を使えないため 3.2.1 節 と 3.2.2 節 で説明する．

3.2.1 最近傍探索

3.1 節 で述べた通り，提案手法は最近傍探索のための空間索引として データポイントを包含する超球を求める．高さ h のデータポイント x の子孫のうち，高さ sh にあるデータポイントの全てを包含する超球を $V_{h,sh,x}$ とする．$h = sh$ の時， $V_{h,sh,x}$ は データポイント x の点となる．また，$\forall y \in children(x), V_{h-1,sh,y} \subset V_{h,sh,x}$ と定義できる．

また，x の全ての子 $children(x)$ の超球を包含する超球 $V_{h,sh,x}$ は中心を代表点とし，半径を $\max\{\max(dist(V_{h,sh,x}\text{の代表点}, V_{h-1,sh,y})) \mid y \in children(x)\}$ とすることで求められる．

3.2.2 逆最近傍探索

逆最近傍探索のための空間索引も 3.2.1 節 と同様に構築できる．最近傍探索と異なる点は，データポイントではなく，最

近傍距離を半径とする超球を包含する空間索引を構築することである．

高さ sh にあるデータポイント y を中心とし，y の最近傍距離を半径とする超球を $W_{sh,y}$ とする．高さ h のデータポイント x の子孫のうち，高さ sh にあるデータポイントを包含する超球を $S_{h,sh,x}$ とする．$h = sh$ の時，$S_{h,sh,x}$ は $W_{sh,x}$ となる．また，$\forall y \in children(x), S_{h-1,sh,y} \subset S_{h,sh,x}$ と定義できる．

また，x の全ての子 $children(x)$ の超球を包含する超球 $S_{h,sh,x}$ は中心を代表点とし，半径を $\max\{\max(dist(S_{h,sh,x}\text{の代表点}, S_{h-1,sh,y})) \mid y \in children(x)\}$ とすることで求められる．

3.3 探　　索

3.3.1 最近傍探索

最近傍探索は，最近傍を探索したい起点データポイント $data$，起点データポイントの存在する高さ $data\_h$ を入力として要求し，$data\_h$ における，$data$ の最近傍 $NN(data)$ を出力する．最近傍探索は，次の処理を再帰的に繰り返すことで実現される．探索する高さを $height$，起点データポイントを $data$，起点データポイントの存在する高さを $data\_h$，探索するクラスタ集合を $clusters$ とおく．暫定最近傍とは，その時点までに探索したデータポイントの中で最も $data$ に近いデータポイントのことを指し，暫定最近傍より $data$ に近いデータポイント new が見つかった場合に new が暫定最近傍へと更新される．

$height$ が $data\_h$ よりも高い場合，2 つのステージで動作する．ステージ 1 で，高さ $height$ の $clusters$ に含まれる各データポイントの超球との最小距離を計算する．ステージ 2 で，最小距離の小さいクラスタ $child$ から順に，クラスタ $child$ の子のデータポイントが暫定最近傍を更新する可能性があるか判定する．更新する可能性がある場合，次に探索する高さを $height - 1$，起点データポイントを $data$，起点データポイントの存在する高さを $data\_h$，次に探索するクラスタ集合を $children(child)$ として次の処理を実行し，返されるデータポイント x が暫定最近傍より近い場合，暫定最近傍を x で更新する．更新する可能性がない場合，残りのクラスタについては無視をする．この時点での暫定最近傍を最近傍とする．また，暫定最近傍が空であるとき，暫定最近傍との距離を ∞ とする．

$height$ が $data\_h$ と同じである場合，$height$ の $clusters$ に含まれるデータポイント x が暫定最近傍を更新するようであれば，暫定最近傍を x で更新する．全てのデータポイントについての処理後の暫定最近傍を返り値とする．

また，この処理における “クラスタ $child$ の子のデータポイントが暫定最近傍を更新する可能性” は次のように判定する．$child$ は，$height$ にある子孫に対して最近傍探索するための超球 $V_{height,data\_h,child}$ を持つため，この超球との最小距離 $min(dist(data, V_{height,data\_h,child}))$ が暫定最近傍との距離より小さい場合に更新する可能性があると判定する．以上の処理を，Algorithm 1 で示した．

Algorithm 1 に疑似コードを示す．最近傍探索の開始時には，

Algorithm 1: 最近傍探索

Input: 起点データポイント $data$，起点データポイントの存在する高さ $data\_h$，木の高さ H
Output: $data$ の最近傍

```
1  Func FIND_NN(d, d_h, h, cand, c_set):
2      distance_list ⟵ empty array
3      foreach c ∈ c_set do
4          distance_list ← distance_list.append
5          (pair(min_dist(d, V_{h,d_h,c}), c))
6      distance_list ← sort(distance_list)
7      foreach (distance, c) ∈ distance_list do
8          if distance ≧ dist(d, cand) then
9              break
10         else
11             if d_h < h then
12                 cand ←
                     FIND_NN(d, d_h, h − 1, cand, children(c))
13             else
14                 cand ← c
15     return cand
16 Func FIND_NN_entry(data, data_h):
17     return FIND_NN((data, data_h, H, null, children(root)))
```

最近傍を探索したいデータポイントを x，x の存在する高さを x_h として処理を開始する．まず，前述の入力で `FIND_NN_entry` 関数を呼び出す (line 16)．最上層のデータポイント集合が探索開始条件が設定され探索が開始される (line 1)．探索対象の各データポイントとの距離を格納するリストを初期化する (line 2)．探索対象の各データポイントとの距離を計算し (line 3)．データポイントとともに格納する (line 5)．リストを距離の昇順にソートする (line 6)．リストの先頭から確認し，暫定最近傍を更新する可能性があるかを判定する (line 8)．可能性がない場合はこのリストの処理を終了する (line 9)．探索中の階層が探索対象の階層でない場合は，データポイントの子の探索を行う (line 12)．探索中の階層が探索対象の階層である場合は，暫定最近傍の更新を行う (line 14)．リストの処理が終了した場合，暫定最近傍を返り値とする (line 15)．

3.3.2 逆最近傍探索

逆最近傍探索は，逆最近傍を探索したい起点データポイント $data$，起点データポイントの存在する高さ $data\_h$ を入力とし要求し，$data\_h$ における，$data$ の逆最近傍集合 $NNset(data)$ を出力する．逆最近傍探索は，次の処理を再帰的に繰り返すことで実現される．また，逆最近傍集合は各再帰において共通の集合である．探索する高さを $height$，起点データポイントを $data$，起点データポイントの存在する高さを $data\_h$，探索するクラスタ集合を $clusters$ とおく．

$height$ が $data\_h$ よりも高い場合，2 つのステージで動作する．$clusters$ の構成要素となる各クラスタ $child$ の子のデータポイントが逆最近傍を更新する可能性があるか判定する．可能性がある場合，次に探索する高さを $height - 1$，起点データポイント $data$，起点データポイントの存在する高さ $data\_h$，次

Algorithm 2: 逆最近傍探索

Input: 起点データポイント $data$，起点データポイントの存在する高さ $data\_h$， 木の高さ H
Output: $data$ の逆最近傍

```
1  Func FIND_NNset(d, d_h, h, cands, c_set):
2      foreach c ∈ c_set do
3          if d ∈ W_{h,d_h,c} then
4              if d_h < h then
5                  cands ← cands.append(
6                  FIND_NNset(d, d_h, h−1, cands, children(c)))
7              else
8                  cands ← c
9      return cands
10 Func FIND_NNset_entry(data, data_h):
11     return
       FIND_NNset((data, data_h, H, null, children(root)))
```

に探索するクラスタ集合 $children(child)$ として次の処理を行なう．可能性がない場合，次のクラスタへと処理をすすめる．

$height$ が $data\_h$ と同じである場合，$height$ の $clusters$ に含まれるデータポイント x 最近傍が $data$ になるようであれば，逆最近傍集合に x を追加する．

また，この処理における “各クラスタ $child$ の子のデータポイントが逆最近傍を更新する可能性” は次のように判定する．$child$ は，$height$ にある子孫に対して逆最近傍探索するための超球 $S_{height,data\_h,child}$ を持つため，この超球に $data$ が含まれる場合，逆最近傍集合を更新する可能性があると判定する．以上の処理を，Algorithm 2 で示した．

Algorithm 2 に疑似コードを示す．被最近傍探索の開始時には，被最近傍を探索したいデータポイントを x，x の存在する高さを x_h として処理を開始する．まず，前述の入力で `FIND_NNset_entry` の関数を呼び出す (line 10)．最上層のデータポイント集合が探索開始条件が設定され探索が開始される (line 1)．各データポイントが被最近傍になる可能性があるかを判定する (line 2)．可能性がない場合は次のデータポイントに進む．可能性があり，探索中の階層が探索対象の階層でない場合は，データポイントの子の探索を行い，返り値を被最近傍に追加する (line 6)．可能性があり，探索中の階層が探索対象の階層である場合は，被最近傍の追加を行う (line 7)．リストの処理が終了した場合，被最近傍を返り値とする (line 9)．

4 評価実験

提案手法の有効性を評価するために，我々の提案した高速化手法および S-FINCH に対し，処理の高速性およびクラスタリング結果の正確性の観点から比較評価を行う．

4.1 実験設定

4.1.1 実験環境

本実験には CPU Intel Xeon E5-2690 2.6 GHz, メモリ 128 GB の Linux サーバを利用する．実験に利用したプログラムは C++ で実装し，g++ (GCC) 9.2.0 で実行速度の最適化を目的に `O2` オプションをつけコンパイルした．

表 2: 実験に用いたデータセット

	N	d	計測範囲
Mice Protein	1,077	77	-
MNIST	10,000	784	-
3D Road Network	434,874	3	$[399900, 400000)$

4.1.2 データセット

本実験では，FINCH の評価実験でも利用されていたタンパク質の出現率と，8 種類の遺伝子型への分類である Mice Protein [8] ，手書き数字分類として有名な MNIST [9] で実験を行った．また新たに，各道路を構成する座標の集合である 3D Road Network [10] でも実験を行った．3D Road Network は道路 ID, 緯度, 経度, 高度 の情報を持つため，道路 ID を除く緯度, 経度, 高度 の 3 次元を入力として実験を行った．各データセットの詳細は表 2 に示す．

本実験では，各データセットをデータストリームであると想定し，1 件ずつ逐次的にデータを追加した際の，データポイント間距離計算を行った回数および実行時間の比較を行う．また，3D Road Network のように長大なデータセットにおいて実行が終了しなかったため，データセットの終盤における 100 件のデータを対象に評価を行った．対象としてデータの区間に関しては表 2 に示した．また，3D Road Network では全ての階層に対して空間索引を構築すると性能が悪化していたため階層 1,2 のみに対して空間索引を構築し階層 3 より上の階層に関しては従来手法と同じ全点への愚直な距離計算を行った．

4.2 高速性

本実験では，各データセットをデータストリームであると想定し逐次的にデータを追加し，データポイント間距離計算を行った回数の比較及び総実行時間の比較を行う．

距離計算実行回数の実験結果を図 3 と 4 に示す．横軸がそのデータが追加された時点でのデータストリームの長さであり，縦軸がそれまでに要した累計データポイント間距離計算回数とその内訳である．Mice Protein, MNIST 共に探索および空間索引構築および最近傍探索のための距離計算回数は少なくなっている．

3D Road Network の実験結果を 図 5 と 6 に示した．また，3D Road Network は，膨大な実行時間を必要としたため，10k 件で実行を打ち切っている．図 5 から提案手法は従来手法に比べて 28%ほど高速に実行可能であることが分かる．図 6 から提案手法は空間索引を構築した階層においては距離計算を 90% 以上削減，実行時間に関しては 70%削減を達成した．

4.3 正確性

提案手法と従来手法 S-FINCH の出力するクラスタリング結果の正確性について評価を行う．

提案手法は S-FINCH における最近傍と逆最近傍を探索する

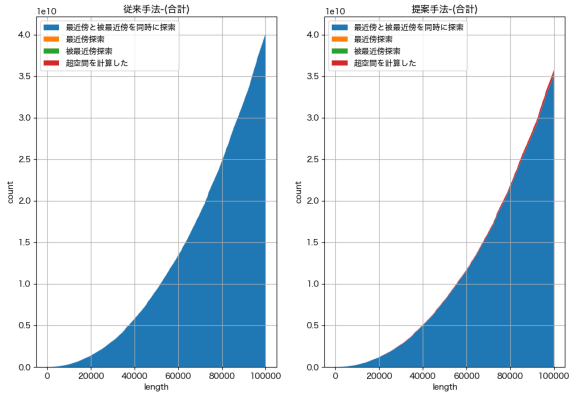

図 3: MNIST のデータポイント間距離計算の実行回数の比較

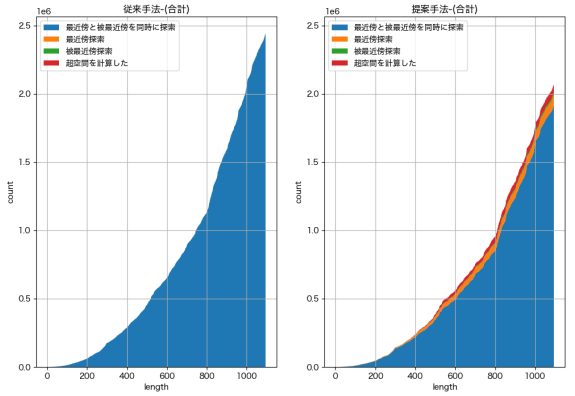

図 4: Mice Protein のデータポイント間距離計算の実行回数の比較

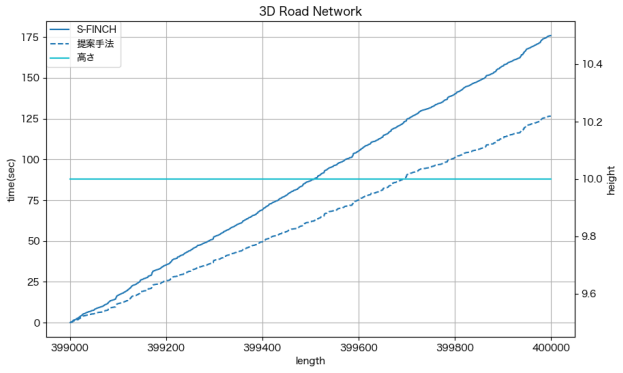

図 5: 3D Road Network の区間内での累計総実行時間と階層の高さ

部分について空間索引を導入する手法であり，この空間索引は愚直に計算を行ったときと同一の厳密解を出力するできる．そのため，提案手法は S-FINCH と完全に一致するクラスタリング結果を出力する手法となり，各データポイントが追加された後に出力される階層的クラスタリングの結果が一致する．この性質を実験的に検証するために，実行終了後における S-FINCH

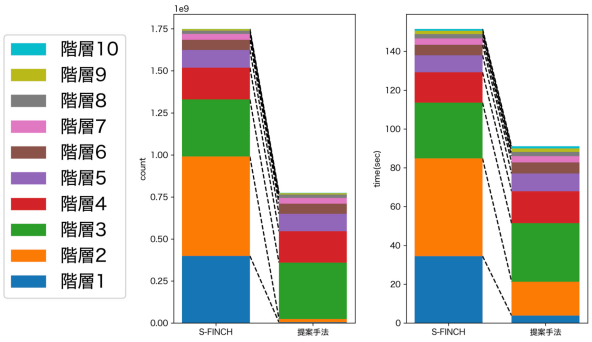

図 6: 3D Road Network の区間内での階層別の距離計算回数 (左) と探索時間 (右)

表 3: 各データセットを S-FINCH と提案手法でクラスタリングした結果の各階層おける NMI スコア

階層 (高さ)	Mice Protein	MNIST	3D Road Network
1	1.000	1.000	1.000
2	1.000	0.999	1.000
3	1.000	1.000	0.999
4	1.000	1.000	0.999
5	1.000	1.000	0.999
6	0.999	1.000	1.000
7	-	1.000	1.000
8	-	-	0.999
9	-	-	1.000

と提案手法のクラスタリング結果を比較する．評価指標として Normalized Mutual Information (NMI) を採用し，評価時には scikit-learn というライブラリに収録されている NMI 関数 [11] を利用して計算した．これによって計算される NMI スコアは 1 に近いほど性能が良く，1 は完全に一致していることを示す．その結果の NMI スコアを表 3 に示す．これは各手法のクラスタリング結果を NMI スコアを階層ごとに計算したものである．- は，そのデータセットのクラスタリング結果に存在しない階層である．その結果，全ての階層で 1.0 に近い値を出力した．しかし，1.0 を超えているものや，1.0 に限りなく近いものの 1.0 ではないもの結果も存在した．1.0 を超えることはないため，実行上の誤差ではないかと推測されるが，確証に至る根拠はない．そのため，クラスタラベルの出力を直接比較することにする．S-FINCH と提案手法は，最近傍・逆最近傍計算以外の処理は同一のものを採用している．そのため，提案手法の最近傍・逆最近傍探索の結果が正確である場合，クラスタリング結果のクラスタラベルに関しても同一のものが出力されるはずである．これを確かめるために各データセットにおいて出力されたクラスタラベルのファイルの差分を `diff` コマンドを用いて確かめた結果，一切の差分がなかった．これによりデータセットの全てにおいて提案手法と従来手法 S-FINCH は同一のクラスタリング結果を出力していることが分かった．

5 おわりに

本稿では，大規模データストリームに対して高速に S-FINCH する手法を提案し，その概要について示した．提案手法では，FINCH のもつ最近傍グラフによるクラスタを包含する空間によって空間索引を構築し，それを用いて最近傍計算における距離計算回数を削減した．提案手法は従来手法の貪欲な最近傍計算と同じ結果を返すことから，提案手法は従来手法の FINCH で生成されるクラスタリング結果と同様の処理結果をより少ない距離計算回数で出力する．これにより，本手法は大規模なデータストリームに対してより精度を損なうことなく空間索引による最近傍探索を可能にした．また，評価実験により低次元データセットに対しては実行時間を 30%削減出来ることを確認した．

今後は，高い階層における距離計算回数の削減，高次元における距離計算回数の削減を通して全体の実行時間の削減を目指すことや，より多くの低次元データセットでの評価を行いたい．

文　献

[1] Saquib Sarfraz, Vivek Sharma, and Rainer Stiefelhagen. Efficient parameter-free clustering using first neighbor relations. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8926–8935, June 2019.

[2] James MacQueen, et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, Vol. 1, pp. 281–297. Oakland, CA, USA, 1967.

[3] Yan-Lin He, Lei Chen, Yuan Xu, Qun-Xiong Zhu, and Shan Lu. A new distributed echo state network integrated with an auto-encoder for dynamic soft sensing. *IEEE Transactions on Instrumentation and Measurement*, Vol. 72, pp. 1–8, 2023.

[4] Yu-Ting Chang, Qiaosong Wang, Wei-Chih Hung, Robinson Piramuthu, Yi-Hsuan Tsai, and Ming-Hsuan Yang. Weakly-supervised semantic segmentation via sub-category exploration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[5] M. Saquib Sarfraz, Naila Murray, Vivek Sharma, Ali Diba, Luc Van Gool, and Rainer Stiefelhagen. Temporally-weighted hierarchical clustering for unsupervised action segmentation. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11220–11229, 2021.

[6] James Cunningham, Jim Davis, Kyle Tarplee, and Juan Vasquez. S-finch: An optimized streaming adaptation to finch clustering. In *2022 26th International Conference on Pattern Recognition (ICPR)*, pp. 1343–1349, Aug 2022.

[7] Saquib Sarfraz, Vivek Sharma, and Rainer Stiefelhagen. Efficient parameter-free clustering using first neighbor relations. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 8934–8943, 2019.

[8] Clara Higuera, Katheleen J Gardiner, and Krzysztof J Cios. Self-organizing feature maps identify proteins critical to learning in a mouse model of down syndrome. *PloS one*, Vol. 10, No. 6, p. e0129126, 2015.

[9] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, Vol. 86, No. 11, pp. 2278–2324, 1998.

[10] Manohar Kaul. 3D Road Network (North Jutland, Denmark). UCI Machine Learning Repository, 2013. DOI: `https://doi.org/10.24432/C5GP51`.

[11] sklearn.metrics.normalized_mutual_info_score. `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.normalized_mutual_info_score.html`, Date of access: 2024/01/31.