NAIST-IS-DD0761013

博士論文

効率的な近傍検索のためのピボット学習法

木村　学

2010年3月17日

奈良先端科学技術大学院大学
情報科学研究科 情報処理学専攻

本論文は奈良先端科学技術大学院大学情報科学研究科に
博士 (工学) 授与の要件として提出した博士論文である。

木村　学

審査委員：

松本　裕治	教授	(主指導教員)
池田　和司	教授	(副指導教員)
乾　健太郎	准教授	(副指導教員)
新保　仁	助教	(副指導教員)

効率的な近傍検索のためのピボット学習法*

木村　学

内容梗概

近傍検索は，与えられたクエリオブジェクトに類似したオブジェクトをデータベース中から見つけるタスクである．画像，音声，ビデオクリップ，3次元オブジェクトなどのマルチメディアデータの急速な増加とともに，その重要性は高まっている．

マルチメディアデータの類似度計算は高コストであり，多くの時間が必要なため，問い合わせ時にクエリと全データオブジェクト間の類似度計算を行うのは現実的ではない．

ピボットに基づくインデキシングは，ピボットオブジェクト集合とデータベース内の全データオブジェクト間類似度を事前に計算しておき，これを利用して問い合わせ時の類似度計算回数を減らす．既存のピボットインデキシング法は，組み合わせ最適化技術を使い，データベース内のデータオブジェクトからピボットを選ぶ．

本論文では，データベース内には無いオブジェクトから，ピボットオブジェクトの集合を学習する機械学習アプローチを提案し，その有効性を示す．このアプローチに基づいて二つのアルゴリズムを提案する．一方はユークリッド空間のオブジェクトに適用できるアルゴリズムである．もう一方はカーネル関数により生じる特徴空間のオブジェクトに適用できるアルゴリズムである．後者のアルゴリズムは，前に述べたマルチメディアデータを含む，多様な非ベクトルデータ（構造データ）に対しても効率的な近傍検索を可能にする．

ベクトルデータや構造データ（自然言語文の構文解析木）のデータベースを用いた実験を行い，データベース内のデータオブジェクトからピボットを選ぶ，既存のインデキシング手法に比べて，提案するアルゴリズムが問い合わせ時の類似度計算回数を大幅に削減することを示す．

*奈良先端科学技術大学院大学 情報科学研究科 情報処理学専攻 博士論文, NAIST-IS-DD0761013, 2010年3月17日.

キーワード

近傍検索, 高速検索, 索引法, 機械学習

Pivot Learning for Efficient Nearest Neighbor Search*

Manabu Kimura

Abstract

Nearest neighbor search is the task of finding objects in a database that are similar to a given query object. Its importance is growing in recent years, with the rapid increase of multimedia data such as images, voice data, video clips, and 3d objects.

Computing similarity between multimedia data is time-consuming, and this poses a challenge in nearest neighbor search. For these data, it is impractical to compute similarity between a query and all database objects at query time.

Pivot-based indexing reduces the number of query-time similarity computations, utilizing pre-computed similarity between a set of *pivot* objects and all objects in the database. The existing pivot-based indexing methods choose pivots from database objects using combinatorial optimization techniques.

In this thesis, we present a novel machine learning approach that learns a set of pivots from objects not in the database. We propose two algorithms following this approach: one applicable to objects in Euclidean spaces, and the other for feature spaces induced by kernel functions. The latter algorithm opens up efficient nearest neighbor search for diverse non-vectorial (structured) data including the multimedia data listed above.

In experiments with databases of vectorial data and structured data (natural language parse trees), the proposed algorithms significantly reduce the number of query-time similarity computations, compared with existing pivot-based methods that choose pivots among database objects.

Keywords:

nearest neighbor search, fast search, indexing method, machine learning

*Doctoral Dissertation, Department of Information Processing, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-DD0761013, March 17, 2010.

目次

図目次

表目次

第1章　はじめに

1.1　近傍検索

近傍検索とは，クエリとなるオブジェクトに類似した（クエリとの距離が小さい）データオブジェクトをデータベースの中から同定するタスクを指す．通信技術，電子機器の発達により，画像，音声，ビデオクリップ，3次元オブジェクト，文書などの高次元のマルチメディアデータが大量に蓄積されており，これら大量のデータの存在によって，近傍検索に対する需要はますます高まっている．さらに，近傍検索は，機械学習，データマイニング，地理情報システム，レコメンデーションシステム，バイオインフォマティクス等，幅広い分野で使われている．

こういった幅広い応用分野において，近傍検索が対象とするデータオブジェクトはベクトルデータだけとは限らない．例えば，文書は文字列であり，最近は構文木，係り受け木等，木構造として扱われることも多い．これら構造データの距離を測るには高いコストがかかる．例えば，2つの文字列の間で編集距離[1]を計算するには，その文字列長 m, n に対し，$O(mn)$ の計算コストが必要である．

このような距離計算をクエリと全てのデータオブジェクトの間で行うことが近傍検索を実現する最も単純な方法である，しかしながら，1回あたりの距離計算が高コストで，かつデータベースの規模が大きい場合，問い合わせ時にそのような処理を行うのは非現実的である．問い合わせ時の計算量を減らし検索を高速化するためには，データベースに対するインデキシングが必要となる．近年では，ある程度の不正確性を許す検索（近似近傍検索）のためのインデキシング手法が注目されている [1, 20, 19, 38]．しかしながら，アプリケーションによっては，検索結果に正確さを要求する．絶対に取りこぼしたくない重要なデータがある場合，このインデキシング手法は不向きである．

本研究では，構造データへの適用を念頭におき，検索結果を正確に返すためのインデキシング手法を検討する．既存研究は，いずれも検索時に距離下界による枝刈りを行う点で共通しており，インデキシングの構造が階層的であるか，無いかによって分類できる．

---

[1]付録 A 参照

インデキシングに階層構造を用いる手法の多くは，低次元空間では高速に検索できるが，10次元以上で線形検索と同等の検索時間になる性質がある．[37] は，grid-file [29] や K-D-B-tree [31]，quadtree [10]，R-tree [11]，R^+-tree [33]，R^*-tree [2]，X-tree [3]，SR-tree [17]，M-tree [8]，TV-tree [21]，hB-tree [22] に対し，理論的，実験的にその性質を示している[2]．

インデキシングに階層構造を用いない方法には，全データオブジェクトとの距離が記録されたピボットと呼ばれるオブジェクトを利用する方法 [27, 6, 5, 36, 32]（ピボット法）がある．この方法は，データベースオブジェクトがクエリからある距離 r 以内にあるかどうかを判定する際に，まず，あらかじめ計算しておいたピボットとオブジェクトとの距離を利用してクエリとデータオブジェクト間の距離下界を低コストで求める．この下界が r を上回る場合には，必然的に実距離も r より大きいことから，このオブジェクトとクエリ間の距離計算を省略（枝刈り）できる．オブジェクトをどの程度枝刈りできるかはピボットに依存する．そのため，多くの枝刈りを可能にするピボット集合をいかにして求めるかが高速化のカギとなる．

これまでに提案されたピボットの構成方法 [27, 6, 5, 36, 32] は，データオブジェクト集合からピボットを選択する点で共通しており，目的関数，最適化方法の観点から整理できる．その中でも，Bustos ら [6] は，全てのデータオブジェクトに関する距離下界の総和を目的関数とし，その目的関数をグリーディに最適化する手法を提案している．同手法は，それまでの手法よりも，検索時間が改善することを実証している．

しかしながら，データオブジェクト集合を含む距離空間には，データオブジェクト集合と同等かそれ以上に探索効率の高いピボット集合が存在しうる．これまでに，距離空間全体からピボットを構成する研究は行われていない．本研究は，ピボットの探索範囲をデータベース上のオブジェクト集合から距離空間全体に拡張する可能性を探る．

1.2 研究の目的

本研究は，ピボットの探索範囲をデータベース上のオブジェクト集合から距離空間全体に拡大する．探索範囲を広げることで，現実的な時間でピボットを構成することが可能かどうか検証する必要がある．研究の目的は，そのようなピボットを構成するためのアルゴリズムを構築することである．

[2] [37] では，クエリの分布及びデータベースの分布がユークリッド空間の超立方体上に分布していることを仮定したうえで理論的に性質を示している．

1.2.1 距離空間全体からのピボットの構成

既存のピボット選択手法は，組み合わせ最適化のアプローチによりデータベース内の有限個のオブジェクト集合からピボットを選択する．このため，一般に無限のピボット候補が存在する距離空間全体からピボットを構成する問題にそのまま既存手法を適用することは難しい．

本論文では，距離空間全体からピボットを効率的に探索するために機械学習アプローチに基づくアルゴリズム，ピボット学習法を提案する，ピボット学習法は，ユークリッド距離を扱い，ニュートン法，準ニュートン法を用いることで，距離空間内から効率的にピボットを探索する．

1.2.2 ピボット学習法の一般化

近傍検索は様々なアプリケーションに用いられる．アプリーケーションによっては，オブジェクトはベクトルデータではない．また，クエリとデータベース上のオブジェクトが異なる性質を持つことがある．例えば，検索エンジンにおいて，クエリ文字列の長さは検索対象である文書，web ページに比べて遥かに短いことが多い．本論文では，以上のような，現実の検索タスクが持つ性質を意識し，提案したピボット学習法を拡張する．

様々な非ベクトルデータ（構造データ）を扱えるように，適用可能な距離尺度をユークリッド距離からカーネル関数に基いて定義された特徴空間上の距離に拡張する．カーネル関数は，データオブジェクトの特徴空間への明示的な写像を行わずに特徴空間上の内積を計算するための関数である．様々な種類のカーネルが提案されており，ベクトルデータだけでなく文字列，木，グラフといった構造データに対して適用可能なカーネルも提案されている．拡張した手法をカーネルピボット学習法と名付ける．木構造データ，木カーネル関数を対象に実験を行い，カーネルピボット学習法の有効性を確認する．

さらに，クエリとデータベース上のオブジェクトの分布が異なることを仮定した拡張も行う．拡張した手法をクエリカーネルピボット学習法と名付ける．ノード数が少ない木をクエリ，ノード数が多い木をデータベースとし，木カーネルを用い実験を行い，クエリカーネルピボット学習法の有効性を確認する．

1.3 論文の構成

本論文の構成は以下となる．2章では，近傍検索を高速化するためにこれまでに提案されたインデキシング手法をまとめる．まず，距離空間一般と距離空間の性質から導かれる距離の下界について説明する．そして，既存のインデキシング手法を下界による枝刈り条件から分類し，問題点について述べる．3章では，ピボット学習法の枠組みを提案し，ベクトル表現されたデータを対象にユークリッド距離空間でアルゴリズムを具現化する．4章では，構造データを扱うため，3章で提案したピボット学習法を，カーネル関数に基づく特徴空間上の距離に適用できるよう一般化する．5章では，クエリとデータベース上のオブジェクトとの分布が異なることを仮定したピボット学習法を提案する．6章では結論と今後の課題について述べる．

第2章　近傍検索のためのインデキシング手法

この章では，近傍検索を高速化するインデキシング手法の既存研究を調査する．まず，高速化の対象となる近傍検索の種類を2.1節で述べる．次に，インデキシング手法が高速化のために用いる，下界による枝刈りの仕組みを2.2節で説明する．2.3節，2.4節では既存のインデキシング手法を階層構造方式とそうでない方式（ピボット法）に分類し，それぞれの代表的な手法について解説する．われわれが3章以降で提案する手法は，このうちピボット法（非階層的インデクシング法）に分類できる．

2.1　近傍検索の種類

近傍検索はクエリqに対し“ある条件”を満たすオブジェクトをデータ集合$\mathbb{U}$から同定する．この“ある条件”によって様々な検索が定義される．以下では，定義される近傍検索の種類を説明する．ただし，説明の際導入する$d(\cdot,\cdot)$は距離尺度とする．[1]

近傍検索の代表的なタスクとして，最近傍検索，K近傍検索，レンジクエリがある．最近傍検索 は，クエリqに対しデータ集合$\mathbb{U}$から以下の性質を満たすオブジェクトu_{NN}を同定する．

$$d(q,u_{NN}) \leq d(q,u), \forall u \in \mathbb{U} - \{u_{NN}\} \tag{2.1}$$

つまり，$\mathbb{U}$の中でqに最も近いオブジェクトu_{NN}を同定するタスクである．K近傍検索 は，最近傍検索を拡張したもので，以下の性質を満たすK個のオブジェクト集合R（$|R| = K$）を同定する．

$$u' \in R \Rightarrow d(q,u') \leq d(q,u), \forall u \in \mathbb{U} - R \tag{2.2}$$

[1]代表的な距離尺度の例については付録Aを参照．

これは，qとの距離が近いオブジェクト上位K個を同定するタスクである．最近傍検索やK近傍検索が決められた個数のオブジェクトを列挙するのに対し，レンジクエリは，qからの距離がある閾値$r(\geq 0)$以下のオブジェクトを列挙する．つまり，以下の性質を満たすオブジェクト集合Rを同定する．

$$R = \{u : d(q,u) \leq r, u \in \mathbb{U}\} \tag{2.3}$$

レンジクエリと最近傍検索，K近傍検索は密接な関係があり，K近傍検索は，問い合わせ時にrを動的に変化させるレンジクエリとみなすことができる．特に本論文で取り扱うピボット法及びピボット学習法における，K近傍検索とレンジクエリの関連については付録Bを参照されたい．

以上の検索にある程度の不正確性を許す検索（近似近傍検索）が定義されている．最近傍検索に対する近似近傍検索は次の定義となる．

- c-近似最近傍検索: $r_{NN} = \min_{u \in \mathbb{U}} d(q,u)$，$c(\geq 1)$に対して，$d(q,u) \leq cr_{NN}$となる$u \in \mathbb{U}$を返す

レンジクエリに対する近似近傍検索は次の定義となる．

- c-近似-r-レンジクエリ: $d(q,u) \leq r$となる$u \in \mathbb{U}$が一つでも存在するなら，$c(\geq 1)$に対し$d(q,u) \leq cr$となる$u \in \mathbb{U}$を返す

この検索タスクを扱う代表的な手法は，Locality Sensitive Hashing (LSH) [1]である．検索条件が前述のタスクに比べて厳しくないため，効率的な検索を実現している．

ただし本論文では，1.1節でも述べたように，検索結果に正確さを要求する検索を対象とする．したがって，本論文で扱う近傍検索は，最近傍検索，K近傍検索，レンジクエリの3種である．

2.2 下界による枝刈り

本研究では検索結果に正確さを要求する検索（最近傍検索，K近傍検索，レンジクエリ）を扱う．以上の検索を高速化するために，既存のインデキシング手法は，計算コストが重い実距離の計算を可能なかぎり行わずに，クエリからの距離が離れたオブジェクトを同定し，クエリに近接していないものとして枝刈りする．この枝刈りのために，多くのインデキシング手法は，距離空間の性質に基づいて計算した実距離の下界を利用する．本節は，この距離空間の性質についてまず述

べ，その上で下界による枝刈りについて説明する．下界の具体的な導出方法は，各インデキシング手法によって異なり，2.3節と2.4節で説明する．

距離空間 $\mathbb{X}$ 及び距離関数 $d : \mathbb{X} \times \mathbb{X} \to \mathbb{R}$ は，$\forall x, y, z \in \mathbb{X}$ に対し，非負性 $(d(x,y) \geq 0)$ と対称性 $(d(x,y) = d(y,x))$，同一性 $(x = y \Leftrightarrow d(x,y) = d(x,y))$，三角不等式 $(d(x,z) \leq d(x,y) + d(y,z))$ の性質が成り立つ空間及び関数のことある．距離空間を仮定することで，オブジェクト間の距離に対する下界を推定できる．下界の導出方法は，インデキシング手法によって異なるが，三角不等式の性質が特に重要となる．

下界を利用した枝刈りの様子を図2.1を用いて説明する．クエリ q からの距離がレンジ r 以下にあるオブジェクトを同定するレンジクエリを考える．クエリ q からデータオブジェクト u への距離を $d(q,u)$, この距離に対する下界を $l(q,u)$ と表記し，$d(q,u)$ はまだ計算しておらず $l(q,u)$ のみを計算している状態を想定する．もし，下界 $l(q,u)$ がレンジ r よりも大きい，すなわち，$l(q,u) > r$ が成立するならば，下界の定義 $d(q,u) \geq l(q,u)$ により，推移律により $d(q,u) \geq l(q,u) > r$ が成立するため，データオブジェクト u をクエリ q からの距離が r よりも大きいと判断できる．一方，下界 $l(q,u)$ がレンジ r 以内にあるとき，すなわち，$l(q,u) \leq r$ が成立するとき，$d(q,u)$ が r よりも小さいか大きいか判別つかないので，実際に $d(q,u)$ を計算しなければならない．下界 $l(q,u)$ が r を超えれば枝刈りできるので，インデキシング手法は下界 $l(q,u)$ を距離 $d(q,u)$ に近づける必要がある．

さらに，インデキシング手法は，下界の計算コストを実距離の計算コストより小さくしなければならない．下界計算に膨大な時間がかかるようなら，これを用いて実距離計算の枝刈りを試みる価値はない．

2.3 階層構造方式によるインデキシング

階層構造インデキシング [39, 35, 29, 31, 10, 11, 33, 2, 3, 17, 8, 21, 22] は，データオブジェクトの集合を階層化したインデクス構造を用いる．この階層構造は，データオブジェクト集合を各ノードに割り当てた木構造とみなすことができる．この際, 木構造の親ノードのデータオブジェクト集合が，子ノードのデータオブジェクト集合を必ず包含する関係を保つようにしている．

ノードには，データオブジェクト集合とともに，クエリからこれらのデータオブジェクトまでの距離の下界を検索時に簡単に計算するための情報が付与される．この情報により，ノードのデータオブジェクト集合を全て枝刈りすることが可能となる．

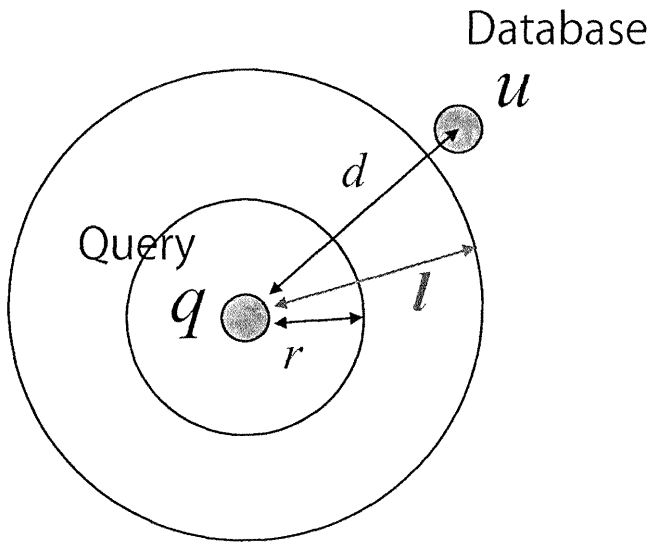


図 2.1: 下界による枝刈り

階層構造インデキシング方式には，ノードに付与する情報に応じてさまざまな手法が提案されているが，2.3.1 節と 2.3.2 節では，このうち代表的な 2 手法（vantage point tree および generalized hyperplane tree）について紹介する．

2.3.1 Vantage point tree 方式

Vantage point tree 方式 [39] は木構造のノードに対して，2 種類の情報を付与する．(i) “バンテージポイント” と呼ばれるデータオブジェクト集合内の一つのデータオブジェクトと，(ii) データオブジェクトの集合を 2 つに分割するバンテージポイントからの “分割距離” である．図 2.2 を例にとり，(i) と (ii) を説明する．図に描かれているオブジェクト集合をノード内の全てのオブジェクトとする，p がバンテージポイント，r_p がピボットオブジェクトからの “分割距離” である．分割距離 r_p によって，オブジェクト集合は，r_p 内のオブジェクト集合，r_p 外のオブジェクト集合に分割する．

前処理として行うインデキシングは，“バンテージポイント”を選択し，“分割距離” 及びオブジェクト集合の分割を決定する操作をトップダウンに行う．インデキシングのスタートとして，データベース内の全てのデータオブジェクトに対し同操作を行い，分割した集合内でも同操作を再帰的に繰り返す．

検索時に行う枝刈りの判定法を説明する．ノード毎に，分割距離内のデータオブジェクト集合，分割距離外のデータオブジェクト集合，それぞれの集合に対して枝刈りが可能かどうか調査する．つまり，クエリqからオブジェクト集合への距離の下界が，クエリqからのレンジrよりも大きいか判定する．分割距離内のデータオブジェクト集合に対しては図2.2の状態のときに枝刈りできる．この状態のとき枝刈りできるのは，

$$d(q,p) - r_p > r \tag{2.4}$$

が成立するためである．ここで，$d(q,p) - r_p$がクエリから分割距離内のデータオブジェクト集合への下界に対応する．下界を計算するためには$d(q,p)$のみを計算すれば良い．

分割距離外のデータオブジェクト集合に対しては図2.3の状態のときに枝刈りできる．この状態のとき枝刈りできるのは，

$$r_p - d(q,p) > r \tag{2.5}$$

が成立するためである．ここで，$r_p - d(q,p)$がクエリから分割距離内のデータオブジェクト集合への下界に対応する．$d(q,p)$を計算しさえすれば，分割距離内のデータオブジェクト集合，分割距離外のデータオブジェクト集合，両者ともに枝刈りの判定ができる．

枝刈りの判定は，まず内側のノードに対して行う．$d(q,p) - r_p > r$が成立すれば，内側のノードを枝刈りし，外側のノードを再帰的に調べる．成立しなければ，次に外側のノードに対する枝刈りの判定を行う．$r_p - d(q,p) > r$が成立すれば，外側のノードを枝刈りし，内側のノードを再帰的に調べる．$d(q,p) - r_p > r$, $r_p - d(q,p) > r$両者とも満足しなければ，外側のノード，内側のノードとも再帰的に調べる．

2.3.2 Generalized hyperplane tree

Generalized hyperplane tree方式 [35] が木構造のノードに付与する情報はデータオブジェクト集合内の2つのデータオブジェクトである．この2つのデータオブジェクトは“サポートポイント”と呼ばれる．図2.4と図2.5に描かれているオブジェクト全てをノード内の全てのオブジェクト，そして，p_l, p_rをサポートポイントとする．p_lとp_rとどちらのサポートポイントへの距離が近いかでデータ集合を2つに分割する．p_rよりもp_lに近いデータオブジェクトは，子ノードではp_l

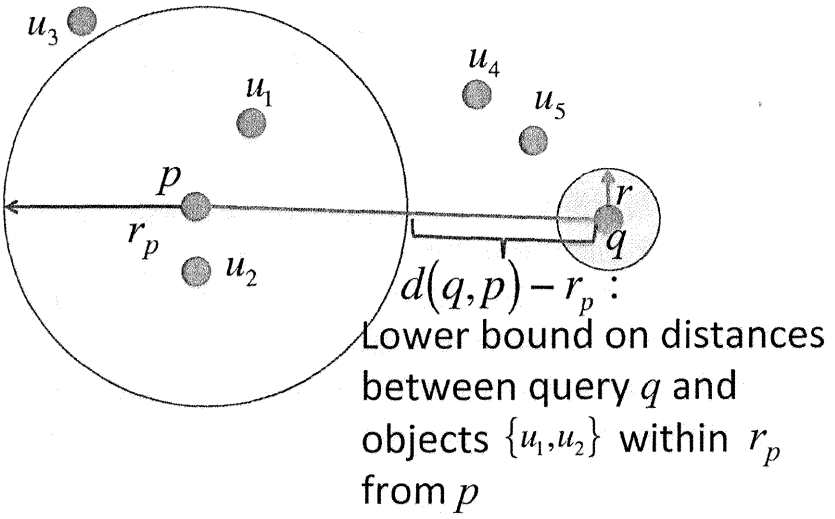


図 2.2: Vantage point tree 方式での内側ノードの枝刈り

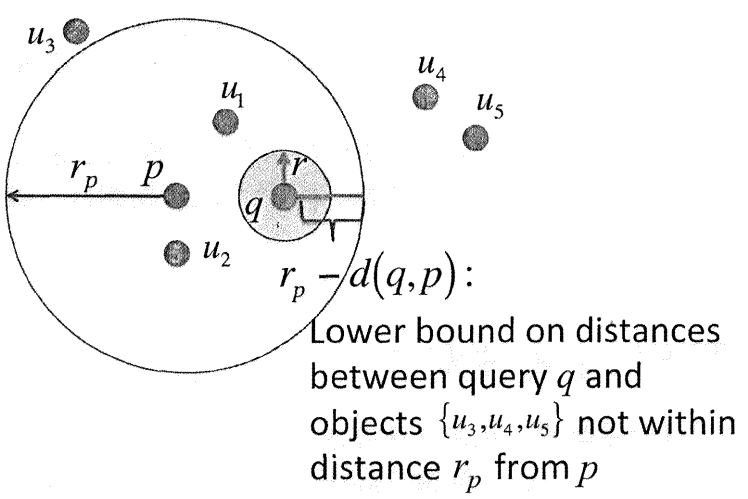


図 2.3: Vantage point tree 方式での外側ノードの枝刈り

側のノードに属する．p_l よりも p_r に近いデータオブジェクトは，子ノードでは p_r 側のノードに属する．

前処理として行うインデキシングは，“サポートポイント”を2つ選択し，ノード内のオブジェクト集合を分割する操作を，Vantage Point Tree 方式と同様にトップダウンに再帰的に行う．

q とレンジ r に対して，検索時に行う枝刈りの判定を説明する．ノード毎に，p_l に近いデータオブジェクト集合，p_r に近いデータオブジェクト集合，それぞれの集合に対して枝刈りが可能かどうか調査する．p_l に近いデータオブジェクト集合に対しては図2.4の状態のときに枝刈りできる．この状態のとき枝刈りできるのは，

$$d(q, p_l) - r > d(q, p_r) + r \tag{2.6}$$

が成立するためである．ここで，q からの領域 r の p_l 側への境界が，p_l と p_r との中間にある超平面を超えないとして，p_l 側のノードオブジェクトを枝刈りしている．上記の式を変形すると，

$$\frac{d(q, p_l) - d(q, p_r)}{2} > r \tag{2.7}$$

となる．$(d(q, p_l) - d(q, p_r))/2$ がクエリから p_l に近いデータオブジェクト集合に対する下界に対応する．

p_r に近いデータオブジェクト集合に対しては図2.5の状態のときに枝刈りできる．この状態のとき枝刈りできるのは，

$$d(q, p_r) - r > d(q, p_l) + r \tag{2.8}$$

が成立するためである．ここで，q からの領域 r の p_r 側への境界が，p_l と p_r との中間にある超平面を超えないとして，p_r 側のノードオブジェクトを枝刈りしている．上記の式を変形すると，

$$\frac{d(q, p_r) - d(q, p_l)}{2} > r \tag{2.9}$$

となる．$(d(q, p_r) - d(q, p_l))/2$ がクエリから p_r に近いデータオブジェクト集合に対する下界に対応する．

クエリから各ピボットへの距離 $d(q, p_l)$，$d(q, p_r)$ を計算しさえすれば，両オブジェクト集合に対する下界を計算でき，枝刈りの判定ができる．枝刈り条件を満足しなければ，その枝刈り条件に対応するオブジェクト集合を再帰的に調べる．

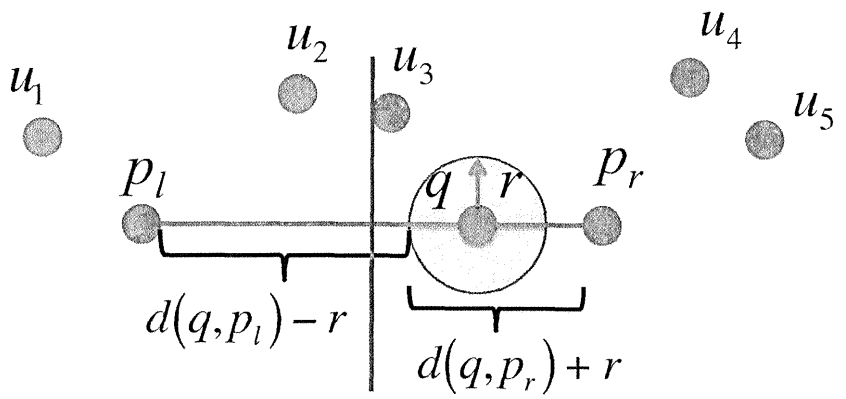


図 2.4: Generalized hyperplane Ttee 方式での p_l 側のノードの枝刈り

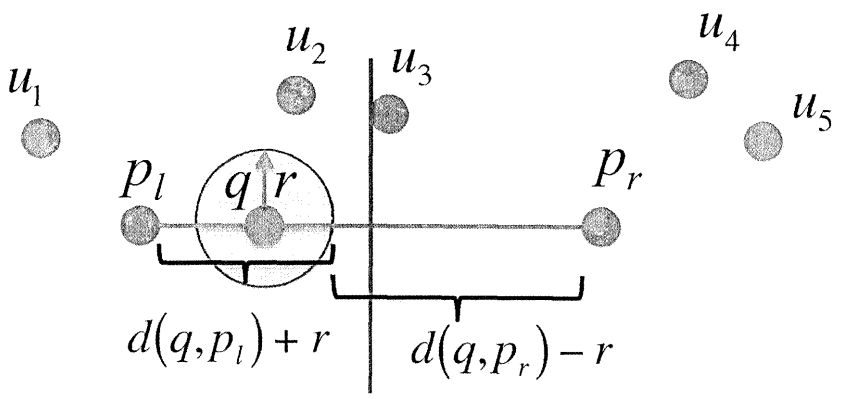


図 2.5: Generalized hyperplane tree 方式での p_r 側のノードの枝刈り

2.4 ピボット法によるインデキシング

階層構造方式は，低次元空間では高速に検索するが，高次元空間ではその高速化の性能に限界がある．例えば，[37] は，grid-file [29] や K-D-B-tree [31]，quadtree [10]，R-tree [11]，R^+-tree [33]，R^*-tree [2]，X-tree [3]，SR-tree [17]，M-tree [8]，TV-tree [21]，hB-tree [22] に対し，10 次元以上で線形検索と同等の検索時間になることを，理論上そして実験でも示している．

これに対し，インデキシングに階層構造を用いない方法として，全データオブジェクトとの距離が記録されたピボットと呼ばれるオブジェクトを利用する方法 [27, 6, 5, 36, 32]（ピボット法）がある．

検索時の時間的計算コストについて，ピボット法は階層構造方式よりも優れている．文献 [7] の Table 1. によれば，階層構造方式の中では vantage point tree [39] が $\log|\mathbb{U}|$ で最も低いのに対し，ピボット法の LAESA [27] が $H+O(1)$ を達成する [27]．さらに，文献 [40] では，それを裏付ける実験結果が報告されている．

本節では，ピボット法についての概念及び関連するインデキシング手法を説明する．

2.4.1 ピボットによる枝刈り

ピボット p は，検索の前処理の段階で $\mathbb{U}$ 上の全てのオブジェクト $u \in \mathbb{U}$ との距離 $d(p,u)$ を計算しておくオブジェクトである．本節では，検索時に H 個のピボット集合 $\mathcal{P}_H = \{p_1, \cdots, p_h, \cdots, p_H; p_h \in \mathbb{U}\}$ を用いて，$d(q,u)$ を計算することなく $d(q,u) > r$ を判定する枝刈について説明する．

今，図2.6に示すように，クエリ q と $u \in \mathbb{U}$ に対し，2つのピボット p_1, p_2 が配置されているものとする．距離の公理（三角不等式）より $d(q,u) \geq |d(q,p_1) - d(p_1,u)|$ が成り立つ．図より，明らかに，$d(q,u) > r$ であるが，$|d(q,p_1) - d(p_1,u)| < r$ 故，上記不等式から $d(q,u) > r$ と判定できない．一方，p_2 に対しては，$d(q,u) \geq |d(q,p_2) - d(p_2,u)| > r$ 故，$d(q,u) > r$ と判定できることが分かる．即ち，H 個のピボット $\mathcal{P}_H = \{p_1, \ldots, p_H\}$ の場合，クエリ q とオブジェクト u 間の距離 $d(q,u)$ の下限値は

$$l_{(q,u)}(\mathcal{P}_H) \stackrel{\text{def}}{=} \max_{1 \leq h \leq H} |d(q,p_h) - d(p_h,u)|$$

で得られるので，$d(q,u) \leq r$ となる $u \in \mathbb{U}$ を検索する場合，$\mathbb{U}' = \{u; u \in \mathbb{U}$ かつ $l_{(q,u)}(\mathcal{P}_H) > r\}$ とすると，$|\mathbb{U}'|$ 個のオブジェクトは，$d(q,u) > r$ として枝刈り可能で，距離計算 $d(q,u)$ が不要となる．

$H << |\mathbb{U}|$ とすると，クエリ q に対するレンジクエリの場合，全探索では $|\mathbb{U}|$ 回の距離計算が必要であるのに対し，上記枝刈り法では，q と $p_i \in P$ との距離計算（H 回）および（$|\mathbb{U}| - |\mathbb{U}'|$）個のオブジェクトとの距離計算で済む．つまり，枝刈り数：$|\mathbb{U}| - (H + (|\mathbb{U}| - |\mathbb{U}'|)) = |\mathbb{U}'| - H$ より，$|\mathbb{U}'| > H$ である限り，ピボット法では，全探索に比べ $|\mathbb{U}'| - H$ 回の距離計算を削減できる．明らかに，$|\mathbb{U}'|$ の数は H 個のピボットの配置に依存する．換言すれば，$|\mathbb{U}'|$ を最大化する $\mathcal{P}_H$ の決定法が課題となる．$\mathcal{P}_H$ の決定法は次節で述べる．

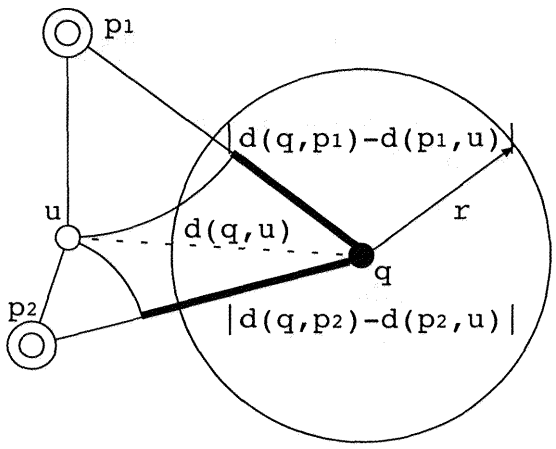


図 2.6: ピボットを利用した枝刈り

2.4.2 インデキシング手法の詳細

これまでに提案されたピボット集合 $\mathcal{P}_H = \{p_1, \cdots, p_H; p_h \in \mathbb{U}\}$ の構成方法について整理する．効率的な探索を実現するために設計された，ピボット集合 $\mathcal{P}_H$ に対する目的関数，そして，最適化方法，二つの観点から整理することができる．

最適化手法に関して，既存手法は，繰り返し計算に基づき，ループの都度 $\mathbb{U}$ の中からピボットを一つずつ確定するインクリメンタルな手段をとる．すなわち，h 回目のループ時に，すでに選んだ $h-1$ 個のピボットに基づき，それぞれの手法の方針により h 個目のピボットを選ぶ．目的関数に関しては，Bustos ら [6]，Micó ら [27]，各々が提案したものがある．

Bustos らは，枝刈りのための条件 $l_{(q,u)}(\mathcal{P}_H) > r$ を満足する $u \in \mathbb{U}$ の数をできる限り増加させるために，$\mathbb{U}$ 全体での $l_{(q,u)}(\mathcal{P}_H)$ の和を表現する目的関数を以下のように提案している．

$$F_{\mathcal{A}}(\mathcal{P}_H) \stackrel{\text{def}}{=} \frac{1}{|\mathcal{A}|} \sum_{i=1}^{W} l_{(a_i,a'_i)}(\mathcal{P}_H). \tag{2.10}$$

ここで，$\mathcal{A} = \{(a_1, a'_1), \cdots, (a_W, a'_W)\}$ は $\mathbb{U}$ からランダムに選ばれる W 個のオブジェクトのペアの集合である．この目的関数 (2.10) を，BNC (*Bustos Navarro Chávez*) 基準と呼ぶ．BNC 基準に前述の最適化手法を組み合わせ，BNC incremental 法 [6] が提案されている．ただし，各ループで選択するピボットは，$\mathbb{U}$ からサンプリングする R 個のデータオブジェクトの中から選ぶ．

Micó らは，ピボット集合がお互いに離れたものにするために，ピボット集合が外れ値である度合いを表現する以下の目的関数を提案している．

$$G(\mathcal{P}_H) = \sum_{h=1}^{H} d(u, p_h). \tag{2.11}$$

加えて，ループの都度，この目的関数（2.11）を最大にするオブジェクトを$\mathbb{U}$の中から選ぶアルゴリズムも提案している．この方法は，LAESA（Outlier法）[27]と呼ばれる．

目的関数を持たない手法としてBrinが採用したMaxMin法[5]がある．これは，Micó らの手法と同様にお互いに離れたピボット集合を構成することを目的とし，以下のようにループの都度新しいピボットp_hを求める．

$$p_h = u^* = \arg\max_{u \in \mathbb{U}} \min \{d(u, p_1), \cdots, d(u, p_{h-1})\}.$$

2.4.3 ピボット法を用いた検索アルゴリズム

ピボット法によるインデクス構造を用いて実際にどのように近傍検索を行うか，を具体的にアルゴリズムを示して解説する．レンジクエリとKNNクエリのための近傍検索アルゴリズムをそれぞれ，アルゴリズム1，アルゴリズム2に示す．

両者とも，オブジェクト集合$\mathbb{U}$の中から検索条件を満たすオブジェクトを$\mathbb{U}_R$として出力し，その検索を高速化するためにピボット集合$\mathcal{P}$及びピボット集合とデータベースとの距離の集合$\{d(u_i, p_h)\}$がインデキシング情報として必要となる．

検索ステップは，共通して，(a) クエリqから各ピボットp_hへの距離$d(q, p_h)$の計算，(b) クエリから各オブジェクトへの下界$\{l(q, u_i)\}$の計算，(c) 各オブジェクトに対し下界が枝刈り条件を満たすかの判定，(d) 枝刈り条件を満たさなかったオブジェクトへの距離計算，を行う．

アルゴリズム1では，**step 2-7.**が(a)，**step 8-10.**が(b)，**step 12-16.**が(c)，**step 17-22.**が(d)に該当する．レンジクエリでは，検索範囲rがあらかじめ判っているので，(a)でピボットへの距離$d(q, p_h)$を計算する際に，距離$d(q, p_h)$がr以内で検索条件を満たすのか調べる．

アルゴリズム2では，(a)，(b)，(c)，(d)に加え，アルゴリズム1におけるrを決定するプロセス(e)が加わる．rは，既に距離計算したオブジェクトの中で，K番目に値が小さいオブジェクトの距離をとる．アルゴリズム中でrは$d(q, u_{Rank_d(K)})$と表現する．$Rank_d(K)$は，引数を順位K，出力をオブジェクトのインデックスとする逆ランキング関数である．既に計算済みの距離$d(q, u_i)$を対象に，K番目に

距離が小さいオブジェクトのインデックスを返す．したがって，プロセス (e) は，未計算のオブジェクト u_i について $d(q, u_i)$ を計算し，計算ずみのオブジェクトも含め距離 $d(q, u_i)$ をソートする操作となる．

r が小さいほどより多くの枝刈りが期待できるので，距離の下界が小さいオブジェクトから，プロセス (e) を実行し r を更新する．そのため，下界のランキングが必要となる．アルゴリズム中で下界のランキングは $Rank_l(k)$ により表現する．$Rank_l(k)$ も引数を順位，出力をオブジェクトのインデックスとする逆ランキング関数である．このプロセスをプロセス (f) とする．

以上のプロセスは，アルゴリズム 2 では，**step 1-3.** が (a)，**step 4-6.** が (b)，**step 7.** が (f)，**step 8-11.** と **step 17-18.** が (e)，**step 13-16.** が (c) に該当する

Algorithm 1 Range query for lower bound pruning

Require: $\mathcal{P}_H$, $\{d(u_i, p_h)\}$

Ensure: $\mathbb{U}_R = \{u : d(q, u) \leq r, u \in \mathbb{U}\}$

$\mathbb{U}_R \leftarrow \{\}$
for each $p_h \in \mathcal{P}_H$ **do**
  Calculate $d(q, p_h)$
  if $d(q, p_h) \leq r$ **then**
    $\mathbb{U}_R \leftarrow \mathbb{U}_R \cup p_h$
  end if
end for
for each $u_i \in (\mathbb{U} - \mathcal{P}_H)$ **do**
  $l(q, u_i) \leftarrow \max_{k=1,\ldots,K} |d(q, p_h) - d(u_i, p_h)|$
end for
$\mathbb{U}_N \leftarrow \{\}$
for each $u_i \in (\mathbb{U} - \mathcal{P}_H)$ **do**
  if $l(q, u_i) \leq r$ **then**
    $\mathbb{U}_N \leftarrow \mathbb{U}_N \cup u_i$
  end if
end for
for each $u_i \in \mathbb{U}_N$ **do**
  Calculate $d(q, u_i)$
  if $d(q, u_i) \leq r$ **then**
    $\mathbb{U}_R \leftarrow \mathbb{U}_R \cup u_i$
  end if
end for
return $\mathbb{U}_R$

Algorithm 2 KNN query for lower bound pruning

Require: $\mathcal{P}_H$, $\{d(u_i, p_h)\}$

Ensure: $\mathbb{U}_R = \mathcal{R}; d(q, u_R) \leq d(q, u), u_R \in \mathcal{R} \subseteqq \mathbb{U}, |R| = K, u \in \mathbb{U} - R\}$

1: **for each** $p_h \in \mathcal{P}_H$ **do**
2: Calculate $d(q, p_h)$
3: **end for**
4: **for each** $u_i \in (\mathbb{U} - \mathcal{P}_H)$ **do**
5: $l(q, u_i) \leftarrow \max_{k=1,\ldots,K} |d(q, p_h) - d(u_i, p_h)|$
6: **end for**
7: Get index $Rank_l(\cdot)$ satisfying $\{l(q, u_{Rank_l(1)}) \leq l(q, u_{Rank_l(2)}) \leq \cdots \leq l(q, u_{Rank_l(|\mathbb{U}-\mathcal{P}_H|)})\}$ by sorting $\{d(q, u_i) : u_i \in (|\mathbb{U} - \mathcal{P}_H)\}$
8: **for** $i = 1$ to K **do**
9: Calculate $d(q, u_{Rank_l(i)})$
10: **end for**
11: Get index $Rank_d(\cdot)$ satisfying $\{d(q, u_{Rank_d(1)}) \leq d(q, u_{Rank_d(2)}) \leq \cdots \leq d(q, u_{Rank_d(K)})\}$ by sorting $\{d(q, u_{Rank_l(i)}) : i = 1, \cdots, K\}$
12: **for** $k = K + 1$ to $|\mathbb{U}|$ **do**
13: **if** $d(q, u_{Rank_d(K)}) \leq l(q, u_{Rank_l(k)})$ **then**
14: $\mathbb{U}_R \leftarrow \{u_{Rank_d(j)} : j = 1, \cdots, K\}$
15: **return** $\mathbb{U}_R$
16: **end if**
17: Calculate $d(q, u_{Rank_l(k)})$
18: Get index $Rank_d(\cdot)$ satisfying $\{d(q, u_{Rank_d(1)}) \leq d(q, u_{Rank_d(2)}) \leq \cdots \leq d(q, u_{Rank_d(k)})\}$ by sorting $\{d(q, u_{Rank_l(i)}) : i = 1, \cdots, K\}$
19: **end for**

2.4.4 既存のピボット法の問題点

前節で述べた従来手法は，データオブジェクト集合からピボットを選択する点で共通している．しかしながら，$\mathbb{X}$ は $\mathbb{U}$ を包含するため，$\mathbb{X}$ には $\mathbb{U}$ 内の要素と同等かそれ以上に効率的な探索を実現するピボットが存在する．これを説明するために，以下の定理を導入する．

定理 1 任意の関数 $E(\cdot)$，および部分集合 $\mathbb{U}_s \subset \mathbb{U}$ に対して，$E(\mathbb{X}_s) \geq E(\mathbb{U}_s)$ となる $\mathbb{X}_s \subset \mathbb{X}$ が存在する．

証明 $\mathbb{U}_s$ が $\mathbb{X}$ 全体で $E(\cdot)$ を最大とする場合，$\mathbb{U} \subset \mathbb{X}$ より，$E(\mathbb{U}_s) = E(\mathbb{X}_s)$ となる $\mathbb{X}_s(= \mathbb{U}_s)$ を得ることができる．$\mathbb{U}_s$ が $\mathbb{X}$ 全体で $E(\cdot)$ を最大としない場合，$\mathbb{X} - \mathbb{U}$ すなわち，$\mathbb{X} - \mathbb{U}$ を包含する $\mathbb{X}$ に $E(\cdot)$ を最大とする $\mathbb{X}_s$ が存在する．したがって，任意の $\mathbb{U}_s$ に対して，$E(\mathbb{X}_s) \geq E(\mathbb{U}_s)$ となる $\mathbb{X}_s$ が存在する．

定理1より，目的関数が適切であれば，$\mathbb{U}$ よりも $\mathbb{X}$ に効率的な探索を実現するピボットが存在するといえる．

また，ピボットをインクリメンタルに求める最適化方法は，並列処理，新たに追加されるデータオブジェクトへの対応が容易ではない．並列処理に関しては，各ピボットを独立に最適化することができないため，同時に複数のピボットへの最適化が不可能である．追加データへの対応に関しては，データが追加されたときにピボットを全データに対して最適なものにするには零から再計算する必要があり，それまでの計算が無駄になる．

次章では，これらの問題点を克服する，新たなピボット構成方法を提案する．

2.5 本章のまとめ

この章では，近傍検索を高速化するインデキシング手法について既存研究を調査した．まず，2.1節で近傍検索の種類を述べた．次に，2.2節で下界による枝刈りの仕組みを説明した．最後の2節，2.3節と2.4節でインデキシング手法について階層構造方式とそうでない方式（ピボット法）を説明した．階層構造方式インデキシングの応用範囲が低次元空間に限られているのに対し，ピボット方式は，高次元空間に対してより頑健と報告されている．しかしながら，既存のピボット法は，より効果的なピボットが存在しうる距離空間全体をピボットの探索範囲にしていないという欠点を持ち，改良の余地がある．

第3章　ピボット学習法

2.4.4 節で述べた通り，既存のピボットインデキシング法には，ピボットオブジェクトをデータベース $\mathbb{U}$ 内のオブジェクトから選択し，オブジェクト空間 $\mathbb{X}$ 全体をピボットの探索範囲として考慮していない，という欠点がある．本章で提案するピボット学習法は，K-Means クラスタリング法 [12] に類似したアルゴリズムで，オブジェクト空間 $\mathbb{X}$ に属す任意のピボットに関する目的関数を局所最適化する．

ピボット学習法は，“各ピボットにオブジェクトのペアを割り当てる操作” と，“各ピポットを割り当てられたオブジェクトのペアに関してより最適的な位置に移動する操作” とを繰り返し実行する．前者の操作では，オブジェクトのペアが全てのピボットの中から最適なピボットを選ぶために，ピボット間で処理に依存関係が生じるが，後者の操作では，ピポットの移動処理が割り当てられたオブジェクトのペアの集合に関してのみ行われるために，各ピボットごとに独立に処理できる．これによって，2.4.4 節の後半で述べた，既存手法のインクリメンタルな最適化方法の問題点[1]を克服する．すなわち，後者の操作でピボットを独立に計算できるので，ピボットを並列に最適化することが可能である．新しいデータの追加に対しても，一度学習したピボットを次の学習の初期点として使用し，零から最適化することなくより短時間でピボットを最適化できる．

以下，3.1 節では，変数空間を $\mathbb{X}$ としたピボットに関する目的関数を設計し，それを前述の後者の操作が可能なものに変形する．3.2 節では，ユークリッド空間での実現例を説明する．

3.1　目的関数

データセット $\mathbb{U}$ が用意されたときに，入力されるクエリ q，レンジ r に対しての枝狩り数を最大化するピボット集合を構成したい．これに留意して，枝狩り条

[1](i) ピボットを並列に選択できない，(ii) 新たに追加されるデータオブジェクトへの対応が容易ではない

件 $l_{(q,u)}(\mathcal{P}_H) > r$ を満たす u の数の見積もりを目的関数とする．クエリとレンジに関する二つの確率分布 $s(q)$ と $s(r)$ を仮定し，以下の形で目的関数を定義する．

$$C(\mathcal{P}_H) \stackrel{\text{def}}{=} \int_{q\in\mathbb{X}, r\in\mathcal{R}^+} |\{u; l_{(q,u)}(\mathcal{P}_H) > r, u \in \mathbb{U}\}| \times s(q)s(r)\, dq\, dr \tag{3.1}$$

しかしながら，$s(q)$ と $s(r)$ は未知であるので，直接 $C(\mathcal{P}_H)$ を最大化することはできない．(3.1) を計算可能なものに変更する．まず，分布 $s(r)$ に関係なく枝狩り数を最大化するために，$l_{(q,u)}(\mathcal{P}_H)$ を最大化するものに変更する．ここで，$l_{(q,u)}(\mathcal{P}_H) \leq d(q,u)$ であり，$l_{(q,u)}(\mathcal{P}_H)$ の下限としての精度向上と見なすこともできる．次に，$s(q)$ を経験分布に置き換え，leave-one-out の要領で各オブジェクト u をクエリ q とみなし，$\mathbb{U}$ 上の全てのペア $\mathcal{B} = \{(u_1,u_2),(u_1,u_3),\cdots,(u_{|\mathbb{U}|-1},u_{|\mathbb{U}|})\}$ について計算するものにする．以上の修正により以下の関数 $F_{\mathcal{B}}(\mathcal{P}_H)$ を定義できる．

$$\begin{aligned} F_{\mathcal{B}}(\mathcal{P}_H) &\stackrel{\text{def}}{=} \sum_{i=1}^{|\mathbb{U}|-1} \sum_{j=i+1}^{|\mathbb{U}|} l_{(u_i,u_j)}(\mathcal{P}_H) \\ &= \sum_{i=1}^{|\mathbb{U}|-1} \sum_{j=i+1}^{|\mathbb{U}|} \max_{1\leq h\leq H} |d(p_h,u_i) - d(p_h,u_j)|. \end{aligned} \tag{3.2}$$

ここで，

$$\begin{aligned} \mathcal{S}_h(\mathcal{P}_H) \stackrel{\text{def}}{=} \{(i,j); h = \arg\max_{1\leq h\leq H} &|d(p_h,u_i) - d(p_h,u_j)| \\ &, (u_i,u_j) \in \mathbb{U}\} \end{aligned} \tag{3.3}$$

とおくと，

$$F_{\mathcal{B}}(\mathcal{P}_H) = \sum_{h=1}^{H} \sum_{(i,j)\in\mathcal{S}_h(\mathcal{P}_h)} |d(p_h,u_i) - d(p_h,u_j)| \tag{3.4}$$

を得る．(3.2) は，各 (i,j) で最適な h での和を求めている．一方 (3.3) のように，各 (i,j) に対し最適な h を定義しておけば，(3.4) の計算で (3.2) を求めることができる．つまり等価な表現である．(3.4) の右辺で h に関する足し合わせ $\sum_{h=1}^{H}$ の内側の項を

$$F_{\mathcal{B}}^{(h)}(\mathcal{P}_H) \stackrel{\text{def}}{=} \sum_{(i,j)\in\mathcal{S}_h(\mathcal{P}_h)} |d(p_h,u_i) - d(p_h,u_j)| \tag{3.5}$$

とおく．(3.5) の右辺は，$\{d(p_h,u_i); i = 1,\cdots,|\mathbb{U}|\}$ の線形和の形で書くことができる．線形和の係数を $\{\lambda_h(i); i = 1,\cdots,|\mathbb{U}|\}$ とおく．$\lambda_h(i)$ は $d(p_h,u_i)$ とそれ以外の項

の大小関係で計算できる．$d(p_h, u_i)$ が他の項以上に大きいときの数を $\lambda_h^+(i)$，他の項以下に小さいときの数を $\lambda_h^-(i)$，すなわち，

$$\begin{aligned}\lambda_h^+(i) &= \left|\{(i,j) \in \mathcal{S}_h(\mathcal{P}_H); d(p_h, u_i) \geq d(p_h, u_j)\}\right|, \\ \lambda_h^-(i) &= \left|\{(i,j) \in \mathcal{S}_h(\mathcal{P}_H); d(p_h, u_i) \leq d(p_h, u_j)\}\right|,\end{aligned}$$

とおくと，$\lambda_h(i)$ は

$$\lambda_h(i) = \lambda_h^+(i) - \lambda_h^-(i),$$

と計算できる．したがって，(3.5) は

$$F_{\mathcal{B}}^{(h)}(\mathcal{P}_H) = \sum_{i=1}^{|\mathbb{U}|} \lambda_h(i) d(p_h, u_i) \tag{3.6}$$

と変形できる．

(3.6) による $F_{\mathcal{B}}(\mathcal{P}_H) = \sum_h^H F_{\mathcal{B}}^{(h)}(\mathcal{P}_H)$ の計算は，相殺される加減算を抑える効果もある．実際，単純な加算回数に着目すれば，(3.4) を直接求めるのに，$|\mathbb{U}|(|\mathbb{U}|-1)/2$ ペアのオブジェクトペア群に対して $|d(p_h, u_i) - d(p_h, u_j)|$ の和が必要である．一方，(3.6) による $F_{\mathcal{B}}(\mathcal{P}_H)$ の計算では，$|\mathbb{U}|H$ 回の必要な項のみの和で計算できる．ピボットに基づいたアプローチは，$|\mathbb{U}| >> H$ を仮定してるので大幅な削減となる．このような相殺項の除去は，大規模データを扱うのに重要と考える．

3.2 ユークリッド空間での実現例

n 次元ベクトル空間を想定し，オブジェクトおよびピボットをそれぞれ $\{\boldsymbol{x}_i : i = 1, \cdots, |\mathbb{U}|\}$，$\{\boldsymbol{p}_h : h = 1, \cdots, H\}$ とする．距離関数は，

$$d(\boldsymbol{x}, \boldsymbol{y}) \stackrel{\text{def}}{=} \sqrt{\textstyle\sum_{e=1}^{n} (x_e - y_e)^2}$$

で定義する通常のユークリッド距離を用いる．

計算手順

提案するアルゴリズムの手順を以下に整理する．

step 1. $\mathbb{U}$ からランダムに H 個オブジェクトを選ぶことにより，$\{\boldsymbol{p}_1, \cdots, \boldsymbol{p}_H\}$ を初期化する．

step 2. 以下のステップを T 回繰り返す.

step 2-1. それぞれの $\boldsymbol{p}_h$ について，分割式（3.3）で $\mathcal{S}_h(\mathcal{P}_H)$ を求め，$\lambda_h(i)$ を計算して目的関数（3.6）を定める.

step 2-2. 各ピボット $\boldsymbol{p}_h$ $(h = 1, \ldots, H)$ を，$\boldsymbol{p}_h \leftarrow \boldsymbol{p}_h + \Delta\boldsymbol{p}_h$[2] により更新する．$\Delta\boldsymbol{p}_h$ はステップ幅であり，連立方程式 $H\Delta\boldsymbol{p}_h + \nabla F_{\mathcal{B}}^{(h)}(\boldsymbol{p}_h)$[3]$= 0$ を解くことで求める[4]．すなわち，ニュートン法 [23] による更新である．ここで，$\nabla F_{\mathcal{B}}^{(h)}(\boldsymbol{p}_h)$ および H は，$F_{\mathcal{B}}^{(h)}(\boldsymbol{p}_h)$ に対する勾配ベクトルおよびヘス行列であり，以下となる.

$$\nabla F_{\mathcal{B}}^{(h)}(\boldsymbol{p}_h) = \sum_{i=1}^{|\mathbb{U}|} \frac{\lambda_h(i)}{d(\boldsymbol{p}_h, \boldsymbol{u}_i)}(\boldsymbol{p}_h - \boldsymbol{u}_i),$$

$$H = \sum_{i=1}^{|\mathbb{U}|} \frac{\lambda_h(i)}{d(\boldsymbol{p}_h, \boldsymbol{u}_i)} \left(\boldsymbol{I}_n - \frac{(\boldsymbol{p}_h - \boldsymbol{u}_i)(\boldsymbol{p}_h - \boldsymbol{u}_i)^T}{d(\boldsymbol{p}_h, \boldsymbol{u}_i)^2} \right).$$

$\boldsymbol{I}_n$ は，n 次元単位行列を表す.

このアルゴリズムはユークリッド空間に特化したものであるが，各ピボットの更新式を定義できさえすれば，他の距離空間にも適用可能である.

提案アルゴリズムによるピボット学習過程の概要は以下となる．与えられたデータセット $\mathbb{U}$ と，ユーザが指定するピボットの個数 H と step 2 の反復回数 T がアルゴリズムへの入力である．まず，step 1 で規定されるように，H 個のピボットの初期値は，データセットからランダムに選択されるので，この段階では，ピボット集合はデータセット $\mathbb{U}$ に包含される．次いで，step 2 による更新を実行することにより，データセット $\mathbb{U}$ には限定されず，より広いオブジェクト空間 $\mathbb{X}$ において，目的関数を改善するようにピボットの場所が変わる．直感的には，我々の行った実験の範囲では，step 2 の反復を実行することにより，H 個のピボット集合の全体でデータセット $\mathbb{U}$ を外包するように各ピボットの場所が変化していく．アルゴリズムは，学習結果のピボット集合と，各ピボットとデータセット $\mathbb{U}$ のオブジェクトとの距離を出力して終了する.

[2] Δ は変数の前に付いて，その変数の増分を表す.
[3] ∇ はスカラーを出力する関数の前に付くことで，引数の座標での勾配ベクトルを表す.
[4] 連立方程式を解くにあたり LL 分解法 [26] を利用した.

補足

提案アルゴリズムは，**step 2-1.**，**step 2-2.** を繰り返す．それぞれのステップが $F_{\mathcal{B}}(\mathcal{P}_H)$ を増加させる理由を述べる．

まず，**step 2-1.** での $\mathcal{S}_h(\mathcal{P}_H)$ による再分割が，$F_{\mathcal{B}}(\mathcal{P}_H)$ を増加させる理由を述べる．全てのピボット $\mathcal{P}_H = \{p_h : h = 1, \cdots, H\}$ を $\mathcal{P}'_H = \{p'_h : h = 1, \cdots, H\}$ に更新後，オブジェクトのペア (u_i, u_j) に関する $|d(p_h, u_i) - d(p_h, u_j)|$ を最大化させるピボットは，更新前に最大化していた p_h が同じ h での更新後の p'_h であるとは限らない．再分割することで，全てのオブジェクトのペアに関して $|d(u_i, p_h) - d(p_h, u_j)|$ を最大にするピボットを選びなおし，同等か，それ以上の値を得ることが可能となる．以上の議論は，ユークリッド空間以外でも成り立つ．

次に，**step 2-2.** でのニュートン法による更新の妥当性を議論する．ニュートン法を適用する場合，ヘス行列 H が負定値となる必要がある．ヘス行列 H が常に負定値であることは保証されないものの，H のトレースが非正値であることが以下のように示される．

$$
\begin{aligned}
trace\{H\} &= (n-1) \sum_{i=1}^{|\mathbb{U}|} \frac{\lambda_h(i)}{d(\boldsymbol{p}_h, \boldsymbol{u}_i)} \\
&= (n-1) \sum_{(i,j) \in S_h(\mathcal{P}_H)} \left(\frac{1}{d(\boldsymbol{p}_h, \boldsymbol{u}_i)} - \frac{1}{d(\boldsymbol{p}_h, \boldsymbol{u}_j)} \right) \\
&= -(n-1) \sum_{(i,j) \in S_h(\mathcal{P}_H)} \frac{d(\boldsymbol{p}_h, \boldsymbol{u}_i) - d(\boldsymbol{p}_h, \boldsymbol{u}_j)}{d(\boldsymbol{p}_h, \boldsymbol{u}_i) d(\boldsymbol{p}_h, \boldsymbol{u}_j)} \leq 0.
\end{aligned}
$$

ヘス行列 H が負定値なら，トレースは負になる．しかし，逆が常に成り立つとは限らない．しかしながら，実験では，$F^{(h)}_{\mathcal{B}}(\mathcal{P}_H)$ に対するヘス行列 H は，常に負定値となっていた．

3.3 ピボット学習法を用いた検索

ピボット学習法を用いて近傍検索を行うためのアルゴリズムを示す．アルゴリズム 3 はレンジクエリのためのアルゴリズム，アルゴリズム 4 は K 近傍検索を実行するためのアルゴリズムである．これらはそれぞれ，従来のピボットインデキシング法の検索アルゴリズム，アルゴリズム 1，アルゴリズム 2 に類似しているが，従来手法がピボット集合を解の候補として考慮するのに対し，提案手法は考慮しない．これは，提案手法では，ピボットがデータベース中のオブジェクトとは限らないことを反映している．

Algorithm 3 Range query for Pivot learning mehods

Require: $\mathcal{P}_H$, $\{d(u_i, p_h)\}$

Ensure: $\mathbb{U}_R = \{u : d(q,u) \leq r, u \in \mathbb{U}\}$

1: **for each** $p_h \in \mathcal{P}_H$ **do**
2: Calculate $d(q, p_h)$
3: **end for**
4: **for each** $u_i \in \mathbb{U}$ **do**
5: $l(q, u_i) \leftarrow \max_{h=1,\ldots,H} |d(q, p_h) - d(u_i, p_h)|$
6: **end for**
7: $\mathbb{U}_N \leftarrow \{\}$
8: **for each** $u_i \in \mathbb{U}$ **do**
9: **if** $l(q, u_i) \leq r$ **then**
10: $\mathbb{U}_N \leftarrow \mathbb{U}_N \cup u_i$
11: **end if**
12: **end for**
13: $\mathbb{U}_R \leftarrow \{\}$
14: **for each** $u_i \in \mathbb{U}_N$ **do**
15: Calculate $d(q, u_i)$
16: **if** $d(q, u_i) \leq r$ **then**
17: $\mathbb{U}_R \leftarrow \mathbb{U}_R \cup u_i$
18: **end if**
19: **end for**
20: **return** $\mathbb{U}_R$

Algorithm 4 KNN query for Pivot learning mehods

Require: $\mathcal{P}_H$, $\{d(u_i, p_h)\}$

Ensure: $\mathbb{U}_R = \{R; d(q,u) \leq d(q,u), u \in R \subseteqq \mathbb{U}, |R| = K, u \in \mathbb{U} - R\}$

1: **for each** $p_h \in \mathcal{P}_H$ **do**
2: Calculate $d(q, p_h)$
3: **end for**
4: **for each** $u_i \in \mathbb{U}$ **do**
5: $l(q, u_i) \leftarrow \max_{h=1,\cdots,H} |d(q, p_h) - d(u_i, p_h)|$
6: **end for**
7: Get index $Rank_l(\cdot)$ satisfying $\{l(q, u_{Rank_l(1)}) \leq l(q, u_{Rank_l(2)}) \leq \cdots \leq l(q, u_{Rank_l(|\mathbb{U}|)})\}$ by sorting $\{d(q, u_i) : i = 1, \cdots, |\mathbb{U}|\}$
8: **for** $i = 1$ to K **do**
9: Calculate $d(q, u_{Rank_l(i)})$
10: **end for**
11: Get index $Rank_d(\cdot)$ satisfying $\{d(q, u_{Rank_d(1)}) \leq d(q, u_{Rank_d(2)}) \leq \cdots \leq d(q, u_{Rank_d(K)})\}$ by sorting $\{d(q, u_{Rank_l(i)}) : i = 1, \cdots, K\}$
12: **for** $h = K + 1$ to $|\mathbb{U}|$ **do**
13: **if** $d(q, u_{Rank_d(K)}) \leq l(q, u_{Rank_l(k)})$ **then**
14: $\mathbb{U}_R \leftarrow \{u_{Rank_d(k')} : k' = 1, \cdots, K\}$
15: **return** $\mathbb{U}_R$
16: **end if**
17: Calculate $d(q, u_{Rank_l(k)})$
18: Get index $Rank_d(\cdot)$ satisfying $\{d(q, u_{Rank_d(1)}) \leq d(q, u_{Rank_d(2)}) \leq \cdots \leq d(q, u_{Rank_d(k)})\}$ by sorting $\{d(q, u_{Rank_l(i)}) : i = 1, \cdots, k\}$
19: **end for**

3.4 ピボット集合を構成するための距離計算回数の比較

ピボットを構成するために必要となる距離計算の実行回数について，提案手法と BNC incremental 法 を比較する．

提案手法は，$|\mathbb{U}|(|\mathbb{U}| - 1)/2$ 個のペアワイズオブジェクトの距離に対し，その下限値の近似精度を最も向上させるように H 個のピボット集合での評価を行う．また **step 2.** では，T 回の反復処理で，ピボットを更新する．したがって，距離計算の実行回数は，$TH|\mathbb{U}|(|\mathbb{U}| - 1)$ となる．BNC incremental 法 と同様に，W 個のペアをサンプリングするならば，$|\mathbb{U}|(|\mathbb{U}| - 1)$ を W に置き換えることになる．

表 3.1: データ名とそれらの概要

名称	内容
8 次元人工データ	ランダムベクトル 8 次元 50,000 データ
16 次元人工データ	ランダムベクトル 16 次元 50,000 データ
NASA データ	画像アーカイブ 20 次元 40,700 データ
毎日新聞データ	毎日新聞 5 年分 20 次元 64,585 データ

表 3.2: 16 次元人工データにおける学習用データとテスト用データでの総近傍検索コストの比較 ($\times 10^3$)

ピボット数	20	40	60	80	100
学習用データによる評価 (総近傍検索コスト)	97.50	4.865	3.044	2.268	1.828
テスト用データによる評価 (総近傍検索コスト)	97.62	4.690	3.044	2.271	1.830

BNC incremental 法は，RW の計算コストが必要な式 (2.10) を，H 回繰り返し評価する．したがって，距離計算の実行回数は RHW となる．ただし，W はサンプリングされたオブジェクトのペアの数であり，提案手法と同様に，オブジェクトの全てのペアを対象とするならば，W を $|\mathbb{U}|(|\mathbb{U}|-1)$ に置き換えることになる．

オブジェクト間の全てのペアについての計算を考えた場合，BNC incremental 法と提案手法，それぞれの距離計算の実行回数は，$RH|\mathbb{U}|(|\mathbb{U}|-1)$，$TH|\mathbb{U}|(|\mathbb{U}|-1)$ である．両者の違いは，R と T である．前者 R の値は，文献 [6] 中の評価実験では，50 に設定されている．一方，後者 T の値は，次節で述べるように 10 で十分な値となる．したがって，上述の R と T を決めたならば，我々の実験では，提案手法は，BNC incremental 法に比べて 5 分の 1 ほどの計算量となる．オブジェクトのペア集合をサンプリングした場合でも同様の議論が成り立つ．

3.5 実験

3.5.1 データ

実験では二種類の人工データと二種類の実データを用いた．表4.2に示すように，二種類の人工データは，それぞれ，8，16次元の単位立方体内に一様に分布した50,000オブジェクトにより構成される．実データの一種類目は，40,700枚からなるNASAの画像データ[5]である．詳細には，Bustosら[6]と同様に，20次元のベクトルに変換された画像オブジェクトを実験に用いた．実データの二種類目は，1993年から1995年まで掲載された64,585文書からなる毎日新聞の国際面記事である．LSA (Latent Semantic Analysis) [25]により，20次元のベクトルに変換された文書オブジェクトを用いた．

3.5.2 評価基準

検索時に要求される距離計算の実行回数は，全データオブジェクトの下限値$l_{(q,u)}(\mathcal{P}_H)$を決定するため共通に計算される$\{d(p_1,q),\cdots,d(p_H,q)\}$の数$H$と，枝刈り条件$(l_{(q,u)}(\mathcal{P}_H) > r)$を満足しなかったオブジェクト群$\mathbb{U}'' = \{u; u \in \mathbb{U}$かつ$l_{(q,u)}(\mathcal{P}_H) \leq r\}$に対して実行される距離計算$d(q,u)$の数との和$H + |\mathbb{U}''|$によって表される．この和を総近傍検索コストと呼び，ピボット構成手法の評価基準として用いる．

3.5.3 提案手法の評価

提案手法でピボット集合を構成し，学習に用いたデータ，それとは異なる別途独立なテスト用のデータ，それぞれをクエリの集合とみなし近傍検索を行い，それぞれの総近傍検索コストの平均値を比較評価した．表3.2は，16次元人工データを用いた実験において，ピボット数を20，40，60，80，100に設定して構成し，その学習用データ，それとは独立に生成した16次元単位立方体に一様に分布した50,000個のテスト用データを使い，それぞれの総近傍検索コストを比較評価した結果である．どのピボット数でも，学習用データとテスト用データでの総近傍検索コストは，ほぼ等しいことが分かる．したがって，学習用データをテストのために利用しても，妥当な評価結果が得られると考えられる．機械学習分野でのテスト用データに対する汎化性能（generalization performance）の概念で言えば，

[5]Sixth DIMACS Implementation Challenge: Available Software. http://www.dimacs.rutgers.edu/Challenges/Sixth/Software.html

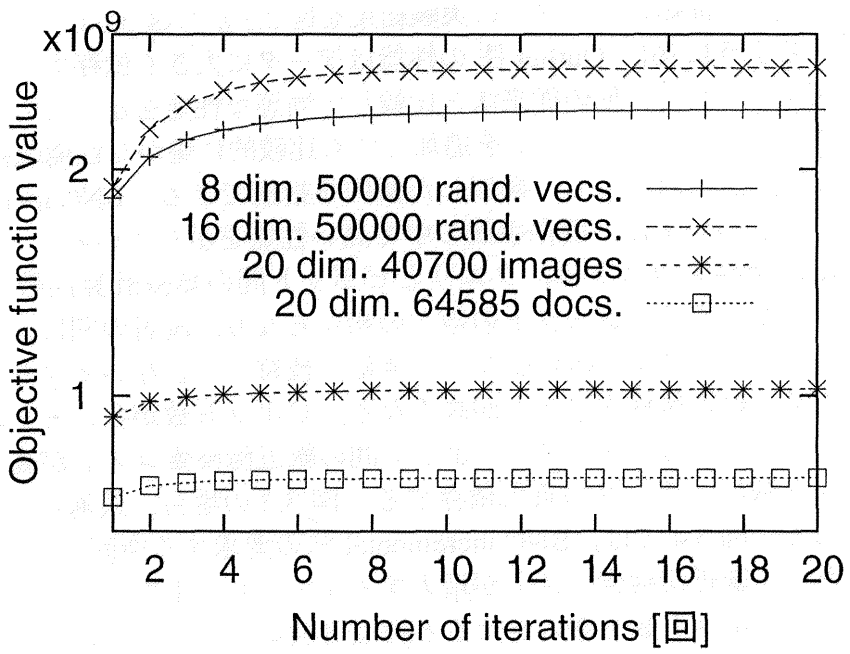


図 3.1: ピボット数 50 での提案手法の学習曲線

学習用データのオブジェクト数が十分に存在するため，それらでの性能と汎化性能が殆ど等しいことを意味すると考える．特に実データでは，オブジェクト数が限られている状況で，より大規模なデータでの比較実験を行うため，それら全てを学習とテストの両方に利用して実験を行った．

学習性能の評価

提案手法は逐次的に最適化を行うため，それぞれの実験データでのループ回数に対する目的関数（3.2）の値の変化を評価した．詳細には 4 種類のデータセットにおいて，最大ループ回数を $T = 20$ に設定し，ループ毎にそれぞれの目的関数（3.2）の値を求めた．ピボット数 50，100 での結果を，それぞれ，図 3.1，図 3.2 に示す．

人工データでは目的関数値が約 10 ループで十分に安定し，実データでは 5 ループ程度でも安定した値となっている．このことは，最初の数ループで効率よく最適化が行われていることを示唆している．以下では，提案手法のループ回数を $T = 10$ に設定し，実験を行った．

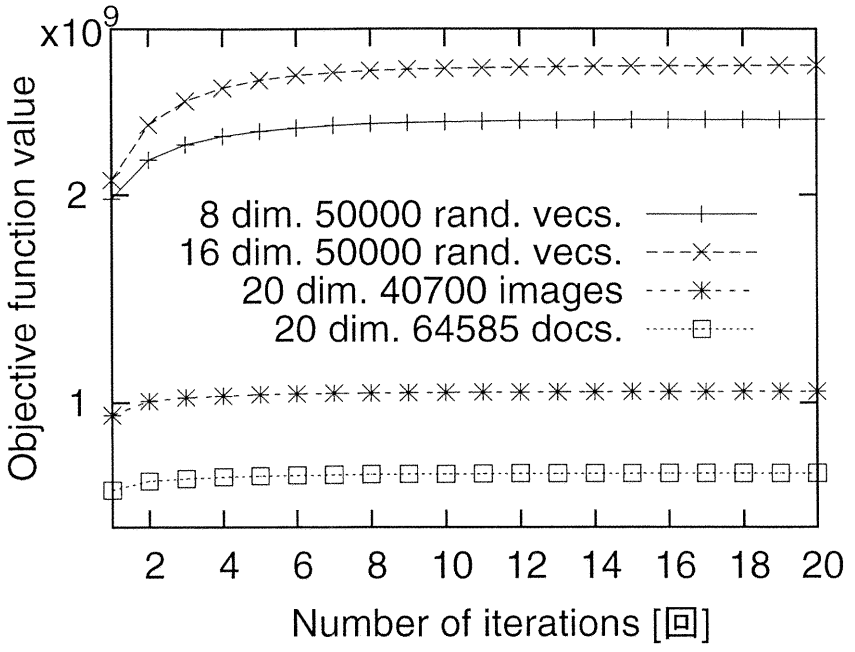


図 3.2: ピボット数 100 での提案手法の学習曲線

3.5.4 従来手法との比較評価

既に述べた従来手法（BNC incremental 法，MaxMin 法，Outliers 法）との比較実験により，提案手法の性能を評価した．

まず，共通の実験設定について記述する．Bustos ら [6] と同様に，データから 0.01% のオブジェクトを同定するレンジクエリで評価した．すなわち，それぞれのデータにおいて，$|\{u : d(q,u) \leq r\}| \approx 0.0001|\mathbb{U}|$ となる r によるレンジクエリである．K 近傍検索については，必要な距離計算回数が，K 番目の最近傍データオブジェクトとクエリの距離を r とするレンジクエリと同等である (詳細は付録 B を参照) ことから，評価は行わず，レンジクエリのみで評価した．我々の評価尺度は，総近傍検索コストの平均値である．つまり，leave-one-out cross-validation のように，データ中の各オブジェクトをクエリとみなし，それぞれの検索での総近傍検索コストを平均した値で評価した．ピボットの数は $10, 20, 30, \ldots, 90, 100$ と変化させた．ただし，16 次元人工データのみ $10, 50, 100, 150, \ldots, 400, 450$ と変化させた．BNC incremental 法 のパラメータの値は，Bustos らが示唆するよう $R = 50$ に設定した．また，実験条件を等しくするため，BNC incremental 法 のオブジェクトのペア集合 $\mathcal{A}$ をデータ集合 $\mathbb{U}$ の全てのペアとした．

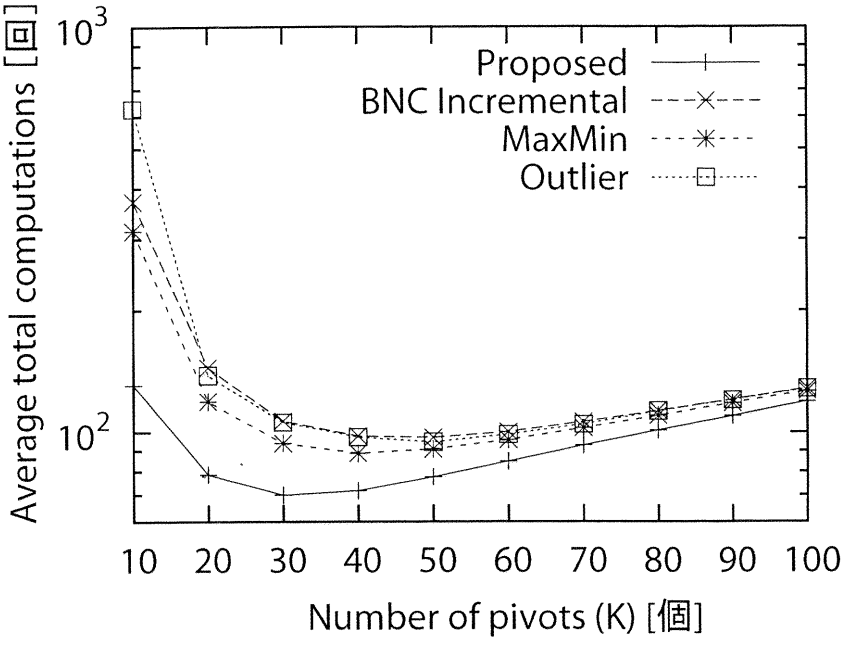


図 3.3: 8 次元ランダムベクトルでの比較

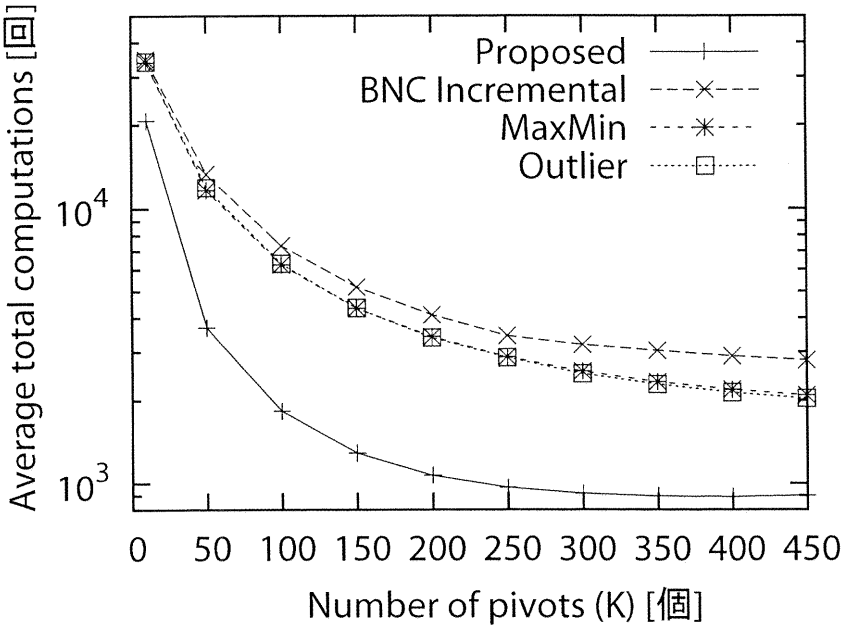


図 3.4: 16 次元ランダムベクトルでの比較

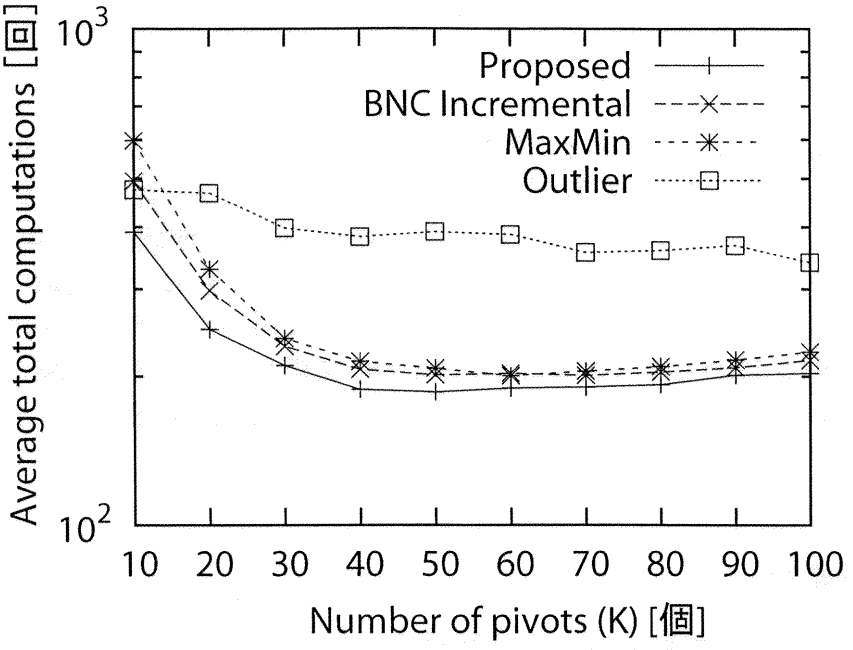


図 3.5: NASA の画像アーカイブでの比較

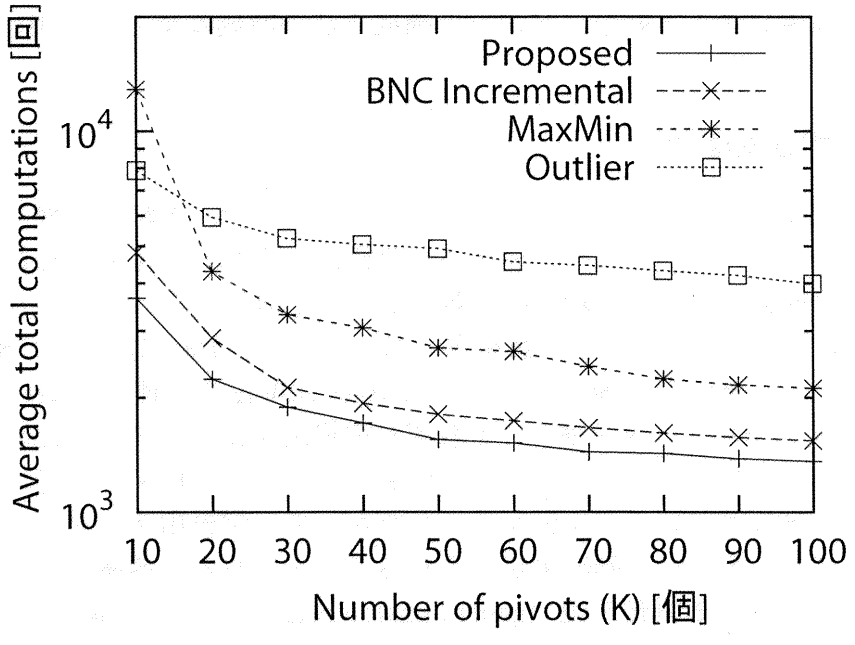


図 3.6: 毎日新聞での比較

前述の4種類のデータでの実験結果を図3.3から図3.6に示す．すでに指摘されているように[6]，Outlier法の性能はデータに大きく依存する．図3.3と図3.4の人工データでは，他の従来法と比較して同等な性能を示すものの，図3.5と図3.6の実データでは，ピボットを追加しても比較的に僅かな性能改善しか得られない．MaxMin法に対しても，類似した傾向を指摘できる．BNC incremental法は，実データにおいては，既存手法の中で高い性能を示している．

総近傍検索コストは，クエリとピボット間の類似計算回数と，下界で枝狩りできなかったオブジェクトの数との和からなり，両者の間にはトレードオフの関係がある．ピボット数が少ないときは，枝狩りできないオブジェクト数が多くなるため，総近傍検索コストが高くなる．ピボット数が多いときは，枝狩り数が増大するが，一方，クエリとピボット間の総近傍検索コストが高くなる．最適なピボット数は，データの質に依存する．図3.3の場合，最適なピボット数は30-40である．図3.4では，BNC incremental法の要求する空間コストが$O(|\mathbb{U}|^2)$と大きいため，総近傍検索コストが増大するまでプロットできなかったが，図3.3と同様に，ピボット数を増大させると総近傍検索コストは増大する．実際，ピボットの数$H = 750$，$H = 1000$としたとき，Outlier法の場合，$1.873 \times 10^3 (H = 750)$が$1.922 \times 10^3 (H = 1000)$となり，MaxMin法の場合，$1.869 \times 10^3 (H = 750)$が$1.919 \times 10^3 (H = 1000)$となり，総近傍検索コストが増大していることを確認している．

提案手法に関しては，図3.3と図3.4とで従来手法との総近傍検索コストの差が顕著であるが，図3.5と図3.6では顕著でない．

明らかに，手法の効果はデータの質に依存する．データの質を定量評価するために，固有次元(C.1)を計算する．固有次元ρの数値は，近傍検索の難しさに比例する．実験データに対する固有次元を表3.3に示す．表3.3より，提案手法は，既存手法にとって困難である固有次元が高いデータほど，既存手法よりも効率的に検索すると解釈できる．

また，8次元人工データ，16次元人工データ，NASAデータ，毎日新聞データに対する，ペアワイズオブジェクトの距離の分布及び下界の分布を図3.8，3.9，3.10，3.11に示す．ここで，下界の分布がオブジェクトの分布に重なっている面積が大きいほど，問題は簡単である．範囲r以下のオブジェクトを検索するとき，その枝狩り条件より，r以上の下界の数が多いほどより多くのオブジェクトを枝狩りできる．すなわち，r以上の下界の分布の面積が増えるように，下界の分布が実際の距離の分布により重なるほうが良い．

図3.8，3.9，3.10，3.11は，表3.3の固有次元の数値が示す問題の難しさの裏づけとなっている．表3.3の値が示すように，NASAデータと毎日新聞データは，

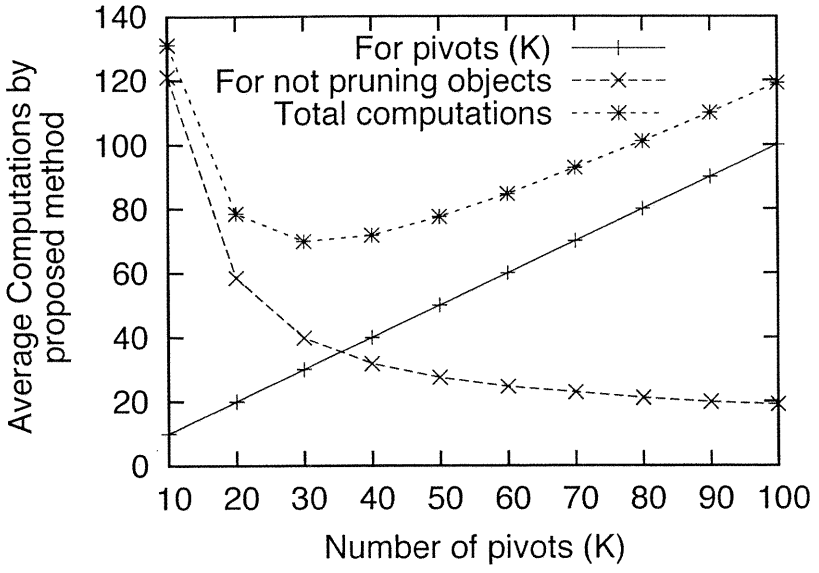


図 3.7: 3.7 の提案手法の分解

8，16 次元人工データよりも問題が簡単であるため，図 3.10 と図 3.11 のように，既存手法で選択したピボットによる下界の分布がペアオブジェクトの実距離の分布と十分に重なっている．逆に，8，16 次元人工データ，特に 16 次元人工データは問題が難しいため，図 3.8 と図 3.9 のように，既存手法で選択したピボットによる下界の分布はペアオブジェクトの実距離の分布に NASA データと毎日新聞データほど重なっていない．提案手法で学習したピボットによる下界の分布は，既存手法で選択したピボットによる下界の分布に比べ，NASA データ（図 3.10），毎日新聞データ（図 3.11）の場合小さく，8 次元データ（図 3.8），16 次元データ（図 3.9）の場合大きく右にシフトしている．これは，表 3.3 と同様に，提案手法が，既存手法にとって簡単な問題では既存手法を僅かながらでも上回り，困難な問題では既存手法をより上回る性能をもつことを示している．特に難しい問題である 16 次元人工データでは，より顕著なものになっている．

3.6 本章のまとめ

本章では，ピボットを利用した手法に基づく近傍検索の高速化に向けて，連続空間上でピボットを探索する問題を新たに設定し，機械学習アプローチにより効率的にピボットを構成する手法を提案した．特にユークリッド空間を対象とし，

表 3.3: 実験データの固有次元

データ	平均 μ	分散 σ^2	固有次元 ρ
8 次元人工データ	2.258	0.243	10.50
16 次元人工データ	3.231	0.238	21.90
NASA データ	1.478	0.212	5.163
毎日新聞データ	0.377	0.020	3.499

ピボット構成に非線形最適化手法を逐次的に適用する方法で効率化を実現した．二種類の人工データと二種類の実データを使った実験では，提案手法が，代表的な従来手法と比較して，総近傍検索コストの平均値を減らし高速化を実現できることを実証した．

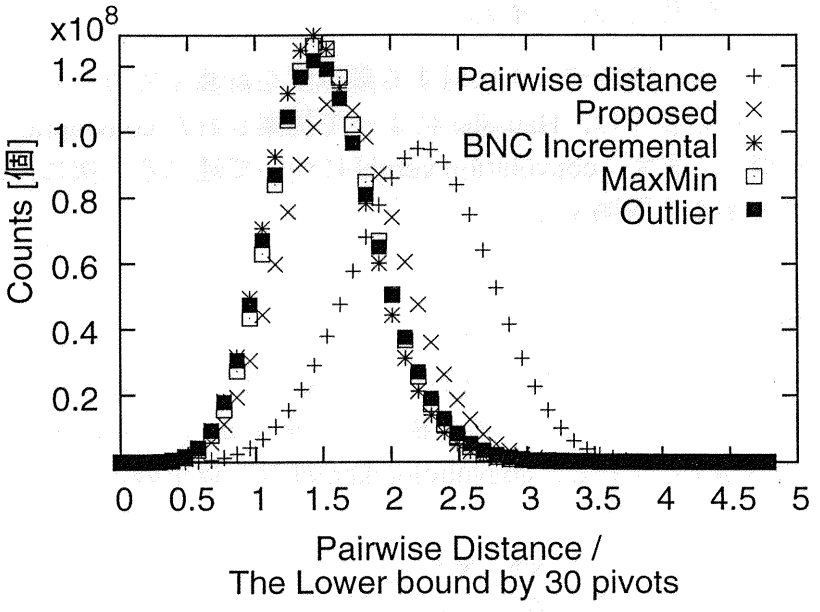


図 3.8: 8 次元人工データに対するペアワイズオブジェクトの距離分布と既存・提案手法による下界の分布

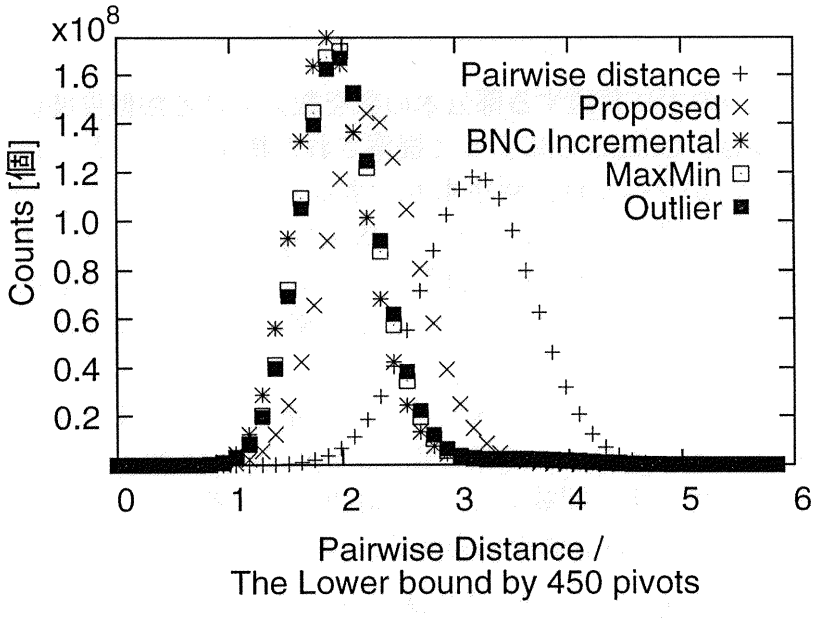


図 3.9: 16 次元人工データに対するペアワイズオブジェクトの距離分布と既存・提案手法による下界の分布

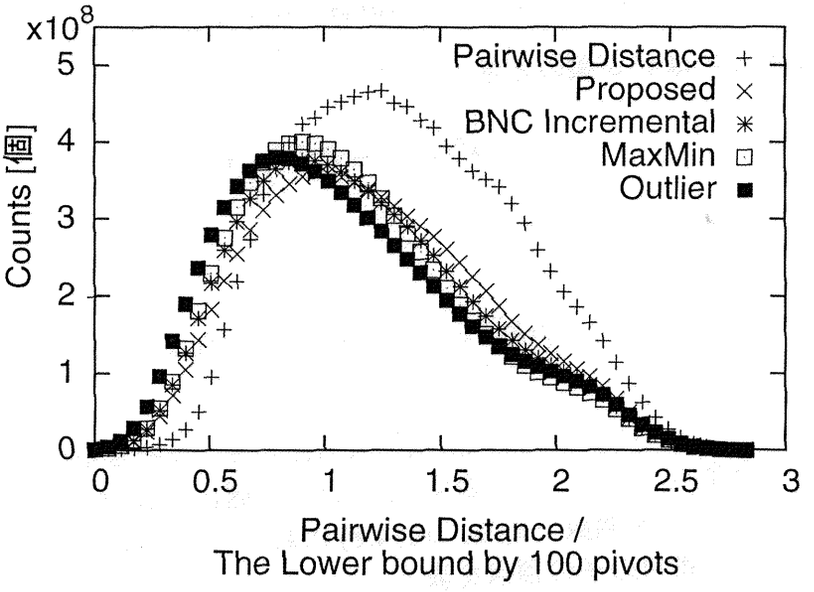


図 3.10: NASA データに対するペアワイズオブジェクトの距離分布と既存・提案手法による下界の分布

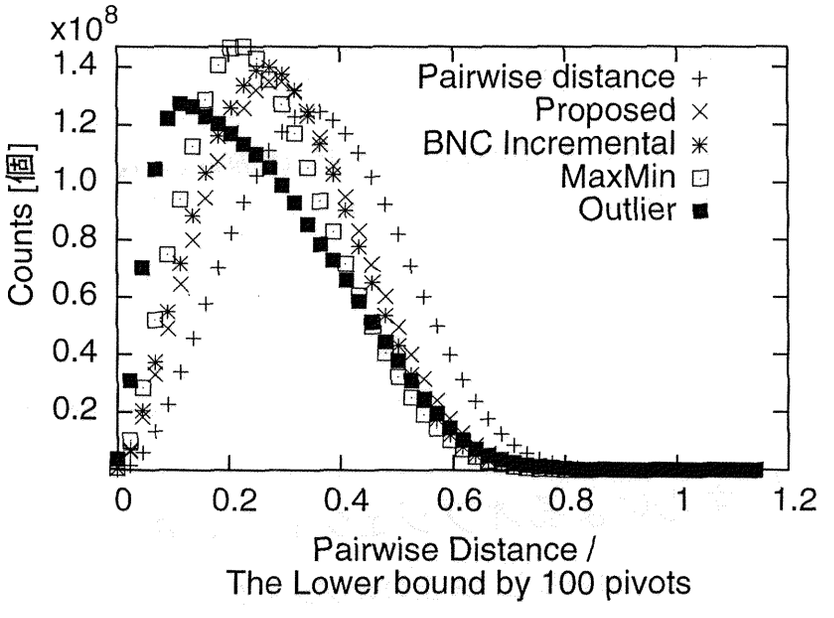


図 3.11: 毎日新聞データに対するペアワイズオブジェクトの距離分布と既存・提案手法による下界の分布

第4章　カーネルピボット学習法

我々の身の回りの電子データでは，ベクトルデータだけでなく文字列，木などの構造データが扱われているが多い．例えば，文章は文字列であり，最近は構文木，係り受け木等，木構造として扱われることも多い．これら構造データを扱えるように，ピボット学習法を拡張することが本章の目的である．つまり，ベクトルデータで検索の高速化に成功したピボット学習法を，構造データも高速に検索できるように一般化する．一般化した手法をカーネルピボット学習法と呼ぶ．

本章で新たに解く問題は，構造データ等の離散オブジェクトについてのピボットを，空間全体に配置する問題である．離散オブジェクトの空間は，ベクトル空間のように連続でない．そのため，非線形最適化手法（ニュートン法等）による最適化ができないので，3章のピボット学習法を直接適用することはできない．

離散オブジェクトについてのピボットを連続空間に配置する問題に対し，我々は，Kernel Principal Component Analysis[4]のようなカーネル法のアイデアを適用する．つまりは，非線形写像後のヒルベルト空間にピボットを定義する．そうすることで，ピボットはカーネル関数の重み付き線形和で計算でき，その重みについて非線形最適化手法で最適化できる．

4.1.2節で説明するようにカーネル関数の一部は，木や文字列，グラフ等の構造データ間の類似度を計算できる．カーネル関数を使用することで，構造データも十分に扱うことができる．

提案手法の性能を確認するため，人工ベクトルデータ，画像ベクトルデータ，係り受け木データ，以上3種類のデータで実験を行う．いずれのデータにおいても，提案手法が既存手法よりも高速に検索することを確認する．

4.1　ピボットの定義

$\mathbb{X}$を，木，文字列，グラフ等が存在するオブジェクト集合の空間とする．$\mathbb{X}$上には，データベースがあるとし，それを$\mathbb{U} = \{u_1, \cdots, u_i, \cdots, u_{|\mathbb{U}|} : u_i \in \mathbb{X}\}$とおく．

$\mathbb{X}$から 写像関数$\phi(\cdot)$によって 写像されたヒルベルト空間$\mathbb{H}$に，H個のピボット$\mathcal{P}_H = \{\phi(p_1), \cdots, \phi(p_h), \cdots, \phi(p_H)\}$を考える．具体的に，それぞれのピボット

$\phi(p_h)$ は，$\mathbb{U}$ から選ばれた少数 M 個のオブジェクト $\{u_{\xi(1)}, \cdots, u_{\xi(m)}, \cdots, u_{\xi(M)}\}$($\xi$ は，$\{1, \ldots, M\} \mapsto \{1, \ldots, |\mathbb{U}|\}$ であり単射関数) を，ϕ で $\mathbb{H}$ 上に写像し，ピボット各々のパラメータ $\{\alpha_{h,1}, \cdots, \alpha_{h,m}, \cdots, \alpha_{h,M} : \alpha_{h,m} \in \mathbb{R}\}$ で重み付けしたものとして定める．すなわち，

$$\phi(p_h) = \sum_{m=1}^{M} \alpha_{h,m} \phi(u_{\xi(m)}) \tag{4.1}$$

とする．

なお，式 (4.1) において，M は $M << |\mathbb{U}|$ とする．もし $\mathbb{U}$ すべてを基底とすると，ピボットのためにデータベース上の全てのオブジェクトを計算することになる．これは近傍検索を効率化する本研究の目的と反する．検索を効率化するためには，基底の数を抑えることが必須となる．

4.1.1 カーネル

$\phi(\cdot)$ を先述の $\mathbb{X}$ から $\mathbb{H}$ への非線形写像関数とする．カーネルは，非線形写像関数 $\phi(\cdot)$ による内積を写像せずに，$\mathbb{H}$ 上の u_1，u_2 の写像間の内積値を計算する以下のような関数である．

$$\kappa(u_1, u_2) =< \phi(u_1), \phi(u_2) > \tag{4.2}$$

ここで，$< \cdot, \cdot >$ は内積計算を表す．

ベクトルデータに対する主なカーネルを次に列挙する．

- ガウシアンカーネル [4]

$$\kappa(u_1, u_2) = \exp\left(-\frac{\|u_1 - u_2\|^2}{\sigma^2}\right) \tag{4.3}$$

  ここで，σ^2 は，非負の実数である．

- 多項式カーネル [4]

$$\kappa(u_1, u_2) = \left(u_1^T u_2 + c\right)^a \tag{4.4}$$

  ここで，a は実数，c は非負の実数である．

次に構造データを扱うカーネルを説明する．

4.1.2 構造データを扱うカーネル

木，文字列，グラフ等の構造データに対する類似度を計算するカーネルが幾つか提案されている．その多くは，Haussler によって提案された convolution kernel [13] の枠組みに基づく．まず，convolution kernel について述べる．次に，木への応用例である木カーネルを説明する．

Convolution kernel

Convolution kernel は，2つの離散的な構造データ構造に対する類似度を共通する部分構造数の総数として定義している．構造データを u，u'，それぞれの部分構造の集合を $S(u), S(u')$ とすると，convolution kernel は，以下のように書くことができる．

$$\kappa(u, u') = \sum_{s \in S(u)} \sum_{s' \in S(u')} \kappa^S(s, s') \tag{4.5}$$

ここで，$\kappa^S(s, s')$ は，2つの部分構造に対するカーネル関数である．$\kappa^S(s, s')$ は，他の部分構造を用いて再帰的に定義される．

木カーネル

木カーネルは，2つの木に共通する部分木の数を数え上げる類似度関数である．代表的な木カーネルは，Collins らによって提案された構文木カーネル [9]，鹿島らによって提案されたがラベル付き順序木カーネル [16] がある．u，u' を木とする．u，u' に関するノード数を $|u|$，$|u'|$ とおく．構文木カーネル，ラベル付き順序木カーネルは，動的計画法により，時間計算量，空間計算量ともに $O(|u||u'|)$ で計算される．以下では，構文木カーネルとラベル付き順序木カーネルの計算方法を紹介する．

まず，木カーネルで共通する計算を説明する．木 u，u' に含まれる頂点の集合をそれぞれ V，V' とする．木カーネルは，頂点 $v \in V$，$v' \in V'$ 毎に，v，v' を根とする部分木についてのカーネル関数 $\kappa^R(v, v')$ を計算する．そして，$v \in V$，$v' \in V'$ 全ての組に関する $\kappa^R(v, v')$ の合計値を出力する．つまり，次のように計算する．

$$\kappa(u, u') = \sum_{v \in V} \sum_{v' \in V'} \kappa^R(v, v') \tag{4.6}$$

これは，式 (4.5) の枠組みに収まる．この $\kappa^R(v, v')$ の計算によって，様々な木カーネルが定義される．

構文木カーネル [9] は，$\kappa^R(v,v')$ の計算を次のように行う．

- v あるいは v' が葉のとき

$$\kappa^R(v,v') = I(l(v) = l(v')) \tag{4.7}$$

- v と v' が葉でないとき

$$\kappa^R(v,v') = I(l(v) = l(v')) \cdot \prod_{i=1}^{\#ch(v)} \left(\kappa^R(ch(v,i), ch(v',i)) + 1\right) \tag{4.8}$$

式 (4.7)-(4.8) において，$l(\cdot)$ は，頂点を入力にとり，ラベルを返す関数である．つまり，$l(v)$ は頂点 v のラベルを表す．$I(\cdot)$ は，括弧内が成立するときに 1，成立しないときに 0 を返す関数である．式 (4.8) において，$\#ch(v)$ は頂点 v の子の数を，$ch(v,i)$ は v の i 番目の子の頂点を表す．

ラベル付き順序木カーネル [16] は，$\kappa^R(v,v')$ の計算を次のように行う．

- v あるいは v' が葉のとき

$$\kappa^R(v,v') = I(l(v) = l(v')) \tag{4.9}$$

- v と v' が葉でないとき

$$\kappa^R(v,v') = I(l(v) = l(v')) \cdot \bar{\kappa}^R_{v,v'}(\#ch(v), \#ch(v')) \tag{4.10}$$

$$\begin{aligned}\bar{\kappa}^R_{v,v'}(i,j) = {} & \bar{\kappa}^R_{v,v'}(i-1,j) + \bar{\kappa}^R_{v,v'}(i,j-1) - \bar{\kappa}^R_{v,v'}(i-1,j-1) \\ & + \bar{\kappa}^R_{v,v'}(i-1,j-1)\kappa^R_{v,v'}(ch(v,i), ch(v',j))\end{aligned} \tag{4.11}$$

式 (4.9)（v あるいは v' が葉のときの計算）は式 (4.7) と同じである．v と v' が葉でないとき，式 (4.10) を計算するために式 (4.11) を再帰的に計算する．

4.2 カーネルによるオブジェクトーピボット間の L_2 距離

定義したピボットを用いると，$\mathbb{H}$ 上のオブジェクトーピボット間の L_2 距離は，カーネル関数 κ のみを用いて計算できる．このことについて，本節で説明する．

ヒルベルト空間$\mathbb{H}$上に定義される距離を$d_{\mathbb{H}}$とする．$\mathbb{H}$上の写像間の距離は，クエリーピボット間（$q-p_h$間），データーピボット間（u_j-p_h間）ではϕで写像後に以下の計算をする．

$$\begin{aligned} d_{\mathbb{H}}(\phi(q),\phi(p_h)) &= \sqrt{\|\phi(q)-\phi(p_h)\|} \\ &= \left\{<\phi(q),\phi(q)>+<\phi(p_h),\phi(p_h)>-2<\phi(q),\phi(p_h)>\right\}^{\frac{1}{2}} \\ d_{\mathbb{H}}(\phi(u_j),\phi(p_h)) &= \sqrt{\|\phi(u_j)-\phi(p_h)\|} \\ &= \left\{<\phi(u_j),\phi(u_j)>+<\phi(p_h),\phi(p_h)>-2<\phi(u_j),\phi(p_h)>\right\}^{\frac{1}{2}} \end{aligned}$$

以後，$d_H(\phi(q),\phi(p_h))$を$d(q,p_h)$に，$d_H(\phi(u_j),\phi(p_h))$を$d(u_j,p_h)$に簡略化し話をすすめる．

内積$<\phi(q),\phi(q)>$，$<\phi(u_j),\phi(u_j)>$，$<\phi(q),\phi(p_h)>$，$<\phi(u_j),\phi(p_h)>$，$<\phi(p_h),\phi(p_h)>$は，カーネル関数κを用いて以下のように表すことができる．

$$<\phi(q),\phi(q)> = \kappa(q,q) \tag{4.12}$$

$$<\phi(u_j),\phi(u_j)> = \kappa(u_j,u_j) \tag{4.13}$$

$$<\phi(q),\phi(p_h)> = \sum_{m=1}^{M}\alpha_{k,m}\kappa(q,u_{\xi(m)}) \tag{4.14}$$

$$<\phi(u_j),\phi(p_h)> = \sum_{m=1}^{M}\alpha_{k,m}\kappa(u_j,u_{\xi(m)}) \tag{4.15}$$

$$\begin{aligned} <\phi(p_h),\phi(p_h)> &= \sum_{m=1}^{M}\alpha_{k,m}^2\kappa(u_{\xi(m)},u_{\xi(m)}) \\ &\quad +2\sum_{m_1=1}^{M-1}\sum_{m_2=m_1+1}^{M}\alpha_{k,m_1}\alpha_{k,m_2}\kappa(u_{\xi(m_1)},u_{\xi(m_2)}) \end{aligned} \tag{4.16}$$

式(4.14)-(4.16)の右辺の演算をそれぞれ，$\kappa(q_i,p_h)$，$\kappa(u_j,p_h)$，$\kappa(p_h,p_h)$とおくと，クエリーピボット間の距離関数$d(q,p_h)$及びデーターピボット間の距離関数$d(u_j,p_h)$は，式(4.12)-(4.16)を用いることにより，

$$d(q,p_h) = \left\{\kappa(q,q)+\kappa(p_h,p_h)-2\kappa(q,p_h)\right\}^{\frac{1}{2}} \tag{4.17}$$

$$d(u_j,p_h) = \left\{\kappa(u_j,u_j)+\kappa(p_h,p_h)-2\kappa(u_j,p_h)\right\}^{\frac{1}{2}} \tag{4.18}$$

となる．結局，$d(q,p_h)$，$d(u_j,p_h)$は，$\phi(\cdot)$による写像後のヒルベルト空間での内積計算をカーネル演算に置き換えることにより効率的に計算できる形式となる．

4.3 目的関数の定義と最適化手法

写像関数ϕによって写像された空間に存在するH個のピボット$\mathcal{P}_H$が最大化する目的関数は，3章と同じ目的関数$F_{\mathcal{B}}(\mathcal{P}_H)$である．つまり，以下の式である．

$$F_{\mathcal{B}}(\mathcal{P}_H) \stackrel{\text{def}}{=} \sum_{i=1}^{|\mathbb{U}|-1} \sum_{j=i+1}^{|\mathbb{U}|} \max_{1\le h\le H} |d(u_i, p_h) - d(u_j, p_h)|. \tag{4.19}$$

カーネルピボット学習法は，$\mathcal{P}_H$を従来のピボット学習法と同様に局所最適化する．まず，データペア集合を分割式

$$\mathcal{S}_h(\mathcal{P}_H) \stackrel{\text{def}}{=} \left\{(i,j) : h = \arg\max_{1\le h\le H} |d(p_h, u_i) - d(p_h, u_j)|, (u_i, u_j) \in \mathbb{U}\right\} \tag{4.20}$$

で分割し，h毎に$\mathcal{S}_h(\mathcal{P}_H)$に基づき

$$\begin{aligned} \lambda_h^+(i) &= \left|\{(i,j) \in \mathcal{S}_h(\mathcal{P}_H) : d(p_h, u_i) \ge d(p_h, u_j)\}\right|, \\ \lambda_h^-(i) &= \left|\{(i,j) \in \mathcal{S}_h(\mathcal{P}_H) : d(p_h, u_i) \le d(p_h, u_j)\}\right|, \end{aligned}$$

を求め，$\lambda_h(i) = \lambda_h^+(i) - \lambda_h^-(i)$を計算する．ここで，$\lambda_h(i)$は，整数，すなわち，$\{\lambda_h(1), \cdots, \lambda_h(i) \cdots, \lambda_h(|\mathbb{U}|) : \lambda_h(i) \in \mathbb{Z}\}$である．そして，ピボット$p_h$毎に$\{\lambda_h(i)\}$に基づいた目的関数

$$F_{\mathcal{B}}^{(h)}(\mathcal{P}_H) = \sum_{i=1}^{|\mathbb{U}|} \lambda_h(i) d(u_i, p_h) \tag{4.21}$$

を形成する．式(4.21)により目的関数(4.19)は

$$F_{\mathcal{B}}(\mathcal{P}_H) = \sum_{h=1}^{H} F_{\mathcal{B}}^{(h)}(\mathcal{P}_H) \tag{4.22}$$

とかける．式(4.21)を通常のピボット学習法と同様に，勾配法により局所最適化する．ただし，従来のピボット学習法がp_hの各次元の値を直接最適化するのに対し，本論文で紹介するカーネルピボット学習法は，$F_{\mathcal{B}}^{(h)}(\mathcal{P}_H)$を$\{\alpha_{h.\tau} : \tau = 1, \cdots, M\}$について局所最適化する．$F_{\mathcal{B}}^{(h)}(\mathcal{P}_H)$を$\alpha_{h.\tau}$について偏微分すると以下のようになる．

$$\frac{\partial F^{(h)}(\mathcal{P}_H)}{\partial \alpha_{h.\tau}} = \sum_{i=1}^{|\mathbb{U}|} \frac{\lambda_{h,i}}{d(p_h, u_i)} \left(\sum_{m=1}^{M} \alpha_{h,m} \kappa(u_{\xi(m)}, u_{\xi(\tau)}) - \kappa(u_i, u_{\xi(\tau)}) \right) \tag{4.23}$$

この偏微分から，準ニュートン法[30]に基づき$\alpha_{h.\tau}$のステップ幅$\Delta\alpha_{h.\tau}$を計算し，$\alpha_{h.\tau} \leftarrow \alpha_{h.\tau} + \Delta\alpha_{h.\tau}$として$\alpha_{h.\tau}$を更新する．$\{\alpha_{h.\tau} : \tau = 1, \cdots, M\}$を更新し，$\phi$によって写像された空間に存在する$p_h$を移動させた後は，式(4.20)以下の計算を再び行い$\{\lambda_h(i)\}$を更新する．そして，$\{\alpha_{h,\tau}\}$と$\{\lambda_h(i)\}$との更新を目的関数(4.19)が収束するまで繰り返す．

4.3.1 ピボットの初期化

目的関数の最適化において前節で説明しなかった，ピボットを初期化する方法についてここで述べる．各ピボットを初期化するには，式(4.1)に示す形でピボットを存在させるために(a)基底となるオブジェクトをデータベースの中からM個選択し，(b)基底となるオブジェクト群に重みをつける，という2つの手順が必要がある．本節では，効率的な近傍検索を実現するという最終目標を念頭に置き，(a)，(b)の具体的な初期化手順を提案する．以後，基底となるオブジェクトのことを基底ピボットと呼ぶ．

検索時にクエリとのカーネル関数の値を計算する基底ピボットの数はできるだけ減らしたい．そこで，本論文ではM個に制限された基底ピボットを全てのピボットで共有する制約をおく．この制約の元で，操作(a)を実現する最も簡単な方法は，基底オブジェクトをデータ集合$\mathbb{U}$からランダムに選択する方法である．ただ，過去にピボット選択手法として，Micóらの Outlier 法[27]，Brin の MaxMin 法[5]，Bustos らの BNC incremental 法[6]が提案されている．これら既存のピボット選択手法により選ぶピボットを基底ピボットとして使用することは，ランダムに選ぶ基底ピボットよりも効率的な検索を実現する可能性がある．以上，基底ピボットを選ぶために実施することが可能な方法を，表4.1に要約する．これら表4.1に書かれた手法を，本論文で提案する目的関数最適化アルゴリズムの一部に組み込む．

ピボットの数Hと基底ピボットの数Mとの関係は，同じ数$H = M$にする．$H > M$，$H < M$としないのは以下の理由による．$H > M$のときは，各ピボットの間で依存関係が生じるため採用しない．一方，$H < M$のとき，クエリと基底ピボットとの間で高コストで計算したカーネル値を全て活かすことができない．つまり，ピボットの数が多いほど枝刈り数が増加する可能性が高まるにもかかわらず，$H < M$として，ピボットの数を基底ピボットの数より少なくするのは非合理的と考える．

初期化(b)は，H個の各ピボットが$M(= H)$個の基底ピボットの中から異なる基底ピボットをそれぞれ一つ選ぶことにする．具体的には，重み$\{\alpha_{h,1}, \alpha_{h,2}, \cdots, \alpha_{h,M} :$

表 4.1: 基底ピボットの選択方法

本論文での呼称	方法
Rnd	ランダムに選択
Outlier	[27] の手法
Maxmin	[5] の手法
BNC incremental	[6] の手法

$h = 1, \cdots, H\}$ のうち $\alpha_{h,h} = 1, (h = 1, \cdots, H)$ とし，それ以外を 0 とおく．こうすることで，既存手法により基底ピボットを選んだ場合，既存手法に対して，提案アルゴリズムがどれくらいの効率化を実現したのか確認することが容易になる．

4.3.2 ピボット最適化アルゴリズム

提案するアルゴリズムの手順を以下に整理する．

step 1. 表 4.1 のいずれかの方法により $\mathbb{U}$ から H 個オブジェクトを選ぶ．そして，$\{\alpha_{h,1}, \alpha_{h,2}, \cdots, \alpha_{h,M} : h = 1, \cdots, H\}$ のうち $\alpha_{h,h} = 1, (h = 1, \cdots, H)$，それ以外を 0 とおき，ピボットを初期化する．ただし，$M = H$ である．

step 2. 以下のステップを収束するまで繰り返す．

step 2-1. それぞれの $\boldsymbol{p}_h$ について，分割式 (4.20) で $S_h(\mathcal{P}_H)$ を求め，$\lambda_h(i)$ を計算して式 (4.21) および式 (4.22) を定める．

step 2-2. $\alpha_{h.\tau} \leftarrow \alpha_{h.\tau} + \Delta\alpha_{h.\tau}$ により $\alpha_{h,\tau}$ $(\tau = 1, \cdots, M)$ を更新することで，各ピボット $\boldsymbol{p}_h$ $(h = 1, \cdots, H)$ を移動させる．ステップ幅 $\Delta\alpha_{h.\tau}$ は準ニュートン法 [30] により決定し，式 (4.23) に示す $F_{\mathcal{B}}^{(h)}(\mathcal{P}_H)$ に対する勾配ベクトルの各要素となる．

4.3.3 補足：正規化した L_2 距離に対するカーネルピボット学習法

提案したアルゴリズムは，正規化した距離にも対応できる．参考のためにここに記す．

正規化したピボット－オブジェクト間の距離$d(u_i, p_h)$は以下のように計算される．

$$d(u_i, p_h) = \sqrt{2}\left\{1 - \frac{\kappa(u_i, p_h)}{\sqrt{\kappa(u_i, u_i)}\sqrt{\kappa(p_h, p_h)}}\right\}^{\frac{1}{2}} \tag{4.24}$$

ここで，$\kappa(u_i, p_h)$と$\kappa(p_h, p_h)$は，式(4.15)及び式(4.16)の各右辺である．

正規化した$d(u_i, p_h)$に基づいた目的関数$F_{\mathcal{B}}^{(h)}(\mathcal{P}_H)$を$\alpha_{h.\tau}$により偏微分した式$\frac{\partial F^{(h)}(\mathcal{P}_H)}{\partial \alpha_{h.\tau}}$を以下に示す．

$$\begin{aligned}\frac{\partial F^{(h)}(\mathcal{P}_H)}{\partial \alpha_{h.\tau}} &= \sum_{i=1}^{|\mathbb{U}|} \frac{\lambda_{h,i}}{d(u_i, p_h)\sqrt{2\kappa(u_i, u_i)\kappa(p_h, p_h)}} \\ &\quad \times \left\{\frac{\kappa(u_i, p_h)}{\kappa(p_h, p_h)} \sum_{m=1}^{M} \alpha_{h,m}\kappa(u_{\xi(m)}, u_{\xi(\tau)}) - \kappa(u_i, u_{\xi(\tau)})\right\}\end{aligned} \tag{4.25}$$

正規化したL_2距離に対するピボット最適化アルゴリズムは，4.3.2に示したアルゴリズムを少しだけ修正したものになる．すなわち，オブジェクト－ピボット間の距離関数を式(4.18)から式(4.24)に変更するのに伴い，**step 2-2.** の準ニュートン法で用いる$F_{\mathcal{B}}^{(h)}(\mathcal{P}_h)$に対する勾配ベクトルが(4.23)から(4.25)に変更される．

4.4 実験

4.4.1 実験セット

実験では，人工ベクトルデータ，画像ベクトルデータ，係り受け木データ，以上3種類のデータでカーネルピボット学習法の性能を調査する．表4.2に実験データの要約を示す．

実験セット1である人工ベクトルデータは，通常のピボット学習法で検索が高速化できたタスクに対して，カーネルピボット学習法の有効性を確認するため導入する．カーネルピボット学習法で用いるカーネル関数は通常のベクトルの内積とし，カーネルピボット学習法で用いる距離関数を通常のピボット学習法と同じユークリッド距離とする．人工ベクトルデータの内容は，16次元の単位立方体内に一様に分布した20,000オブジェクトにより構成される．

実験セット2である画像ベクトルデータでは，画像データに対してカーネルピボット学習法の高速化の性能を確認するために導入する．カーネル関数は画像処理の分野で良く用いられるガウシアンカーネルとする．画像ベクトルデータの内

表 4.2: 実験で使用するデータセットとカーネル

名称	内容	クエリ数	データ数	カーネル	正規化
実験セット 1	ランダムベクトル 16 次元	5,000	15,000	内積	no
実験セット 2	画像データ 20 次元	1,1750	3,5250	ガウシアンカーネル	no
実験セット 3	Penn TreeBank 係り受け木	1,921	1,993	木カーネル	no
実験セット 4	Penn TreeBank 係り受け木	1,921	1,993	木カーネル	yes

容は，40,700 枚からなる NASA の画像データ[1]である．詳細には，Bustos ら [6] と同様に，20 次元のベクトルに変換された画像オブジェクトを実験に用いる．

実験セット 3 である係り受け木データは，自然言語処理分野で用いる木構造データである．自然言語処理分野及び構造データでカーネルピボット学習法の高速化の性能を確認するために導入する．係り受け木データの内容は，係り受け木変換ツール[2] により構文木から係り受け木に変換された Penn TreeBank 3,914 文である．カーネル関数は木カーネル [9] とする．また，類義語を吸収した検索を行うため，PLSI [14] により単語毎にトピック分布を求め，その内積値を単語間の類似度とする．

実験セット 4 は，実験セット 3 と同様の設定とし，カーネルに基づく距離関数を正規化し，実験をする．

4.4.2 学習性能の評価

ここでは，繰り返し毎に目的関数の値をプロットし，収束の様子を観測する．そのプロット図から収束の早さを評価する．そして，繰り返しの回数を決定する．

[1]Sixth DIMACS Implementation Challenge: Available Software. http://www.dimacs.rutgers.edu/Challenges/Sixth/Software.html

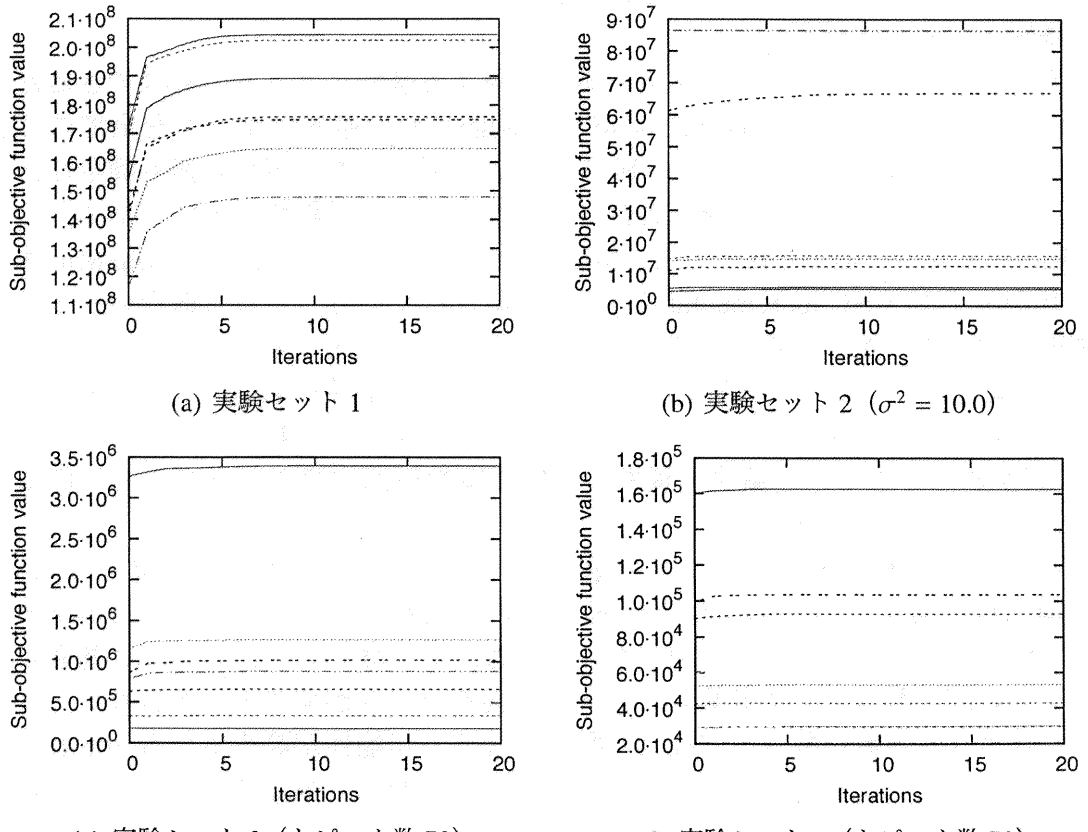
[2]The LTH Constituent-to-Dependency Conversion Tool for Penn-style Treebanks: http://nlp.cs.lth.se/software/treebank_converter

準ニュートン法の繰り返し回数の決定

提案するアルゴリズムは，繰り返しの都度，準ニュートン法（LBFGS法）[30]を用い，$F_{\mathcal{B}}^{(h)}(\mathcal{P}_H)$を収束させる．その収束の様子を観測し，繰り返し回数の決定することにする．

全ての実験セットで，準ニュートン法のループ回数に対する目的関数値$F_{\mathcal{B}}^{(h)}(\mathcal{P}_h)$（式(4.21)）の変化をピボット$p_h$毎に調べる．$K = M = 10$個の基底ピボットをランダムに選択し，目的関数(4.21)の値を20回目の繰り返しまでプロットする．実験セット1（ランダムベクトル）に関するグラフを図4.1(a)に，$\sigma^2 = 10$とした実験セット2（画像データ）に関するグラフを図4.1(b)に，トピック数70とした実験セット3（木データ）に関するグラフを図4.1(c)に示す．トピック数70とした実験セット4（木データ，正規化距離）に関するグラフを図4.1(d)に示す．

全ての実験において，20回目の繰り返しには，目的関数は大幅に更新されていない．そこで，本手法は，準ニュートン法の繰り返し回数を20とする．



(a) 実験セット1

(b) 実験セット2（$\sigma^2 = 10.0$）

(c) 実験セット3（トピック数70）

(d) 実験セット4（トピック数70）

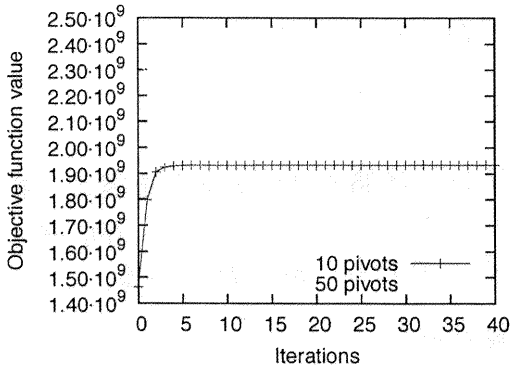
図4.1: ピボット毎の式(4.21)の学習曲線(ピボット数$K = M = 10$)

カーネルピボット学習法の繰り返し回数の決定

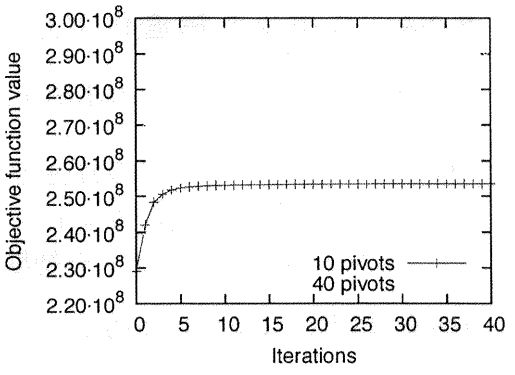
次は，目的関数値 $F_{\mathcal{B}}(\mathcal{P}_H)$（式(4.22)）の収束を調査する．その収束の様子から，ピボット最適化アルゴリズム（4.3.2節）が **step 2** を繰り返す回数を決定する．

ランダムに基底ピボットを選択し，目的関数値(4.22)を40回目の繰り返しまでプロットする．ピボット数は，実験セット1のみ10と50，それ以外を10と40設定した．実験セット1（ランダムベクトル）に関するグラフを図4.2(a)に，$\sigma^2 = 10$ とした実験セット2（画像データ）に関するグラフを図4.2(b)に，トピック数70とした実験セット3（木データ）に関するグラフを図4.2(c)に，トピック数70とした実験セット4（木データ，正規化距離）に関するグラフを図4.2(d)に示す，

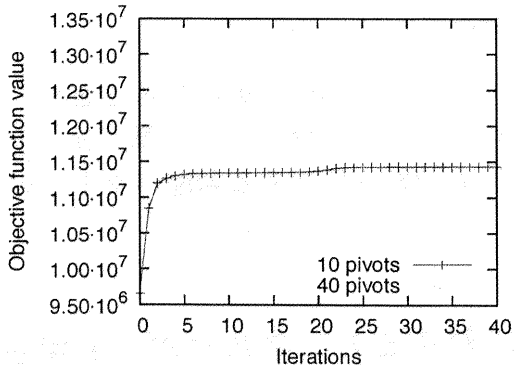
全ての実験において，40回目の繰り返しには，目的関数は大幅に更新されていない．そこで，本手法は，繰り返し回数を40とする．



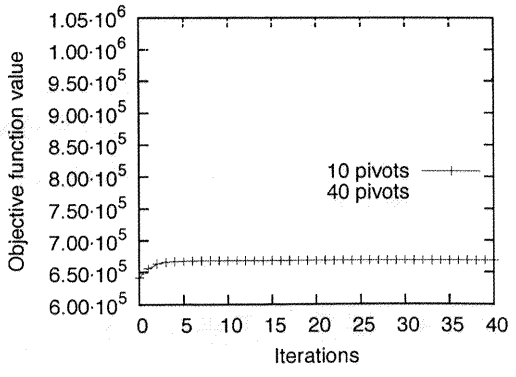
(a) 実験セット 1



(b) 実験セット 2（$\sigma^2 = 10.0$）



(c) 実験セット 3（トピック数 70）



(d) 実験セット 4（トピック数 70）

図 4.2: 式(4.22)の学習曲線(ピボット数10，40，ただし実験セット1のみ10，50)

4.4.3 比較する検索方法

カーネルピボット学習法は，インデキシングした情報によって，3種類の検索ができる．ここでは，その検索方法と計算コスト，比較指標について説明する．

カーネルピボット学習法は，枝刈りが可能か $|d(q,p_h)-d(u_i,p_h)|>r$ を判定するために $d(u_i,p_h)$ を保持し，$d(q,p_h)$ の計算に備える必要がある．そのため，$\{\kappa(u_{\xi(m)},u_i):m=1,\cdots,M,i=1,\cdots,|\mathbb{U}|\}$，$\{\alpha_{h,m}:h=1,\cdots,H,m=1,\cdots,M\}$，$\{\kappa(u_i,u_i):i=1,\cdots,|\mathbb{U}|\}$ は，インデキシング情報として事前に記録する．一方で，検索時に $d(q,p_h)$ を計算するために，クエリ q と $\{u_{\xi(m)}\}$ との間のカーネル値 $\{\kappa(q,u_{\xi(m)}):m=1,\cdots,M\}$ および $\kappa(q,q)$ を計算する必要がある．以上の計算したカーネル値を組み合わせることで3種類の検索を行うことができる．

1種類目の検索は，基底ピボットによる検索である．これは，$\{u_{\xi(m)}\}$ をピボット集合とみなす方法である．クエリ－データ（$q-u$）間の距離下界を

$$l_{(q,u)}(\{u_{\xi(m)}\})=\max_{1\le m\le M}|d(q,u_{\xi(m)})-d(u,u_{\xi(m)})|$$

ととる方法である．これにより，従来手法の検索も実現できる．

2種類目の検索は，式(4.1)によるピボットの検索である．本節の冒頭で述べた検索である．$q-u$ 間の下界値を $l_{(q,u)}(\mathcal{P}_H)$ ととる方式である．

3種類目の検索は，1種類目と2種類目とを組み合わせた検索である．つまり，$q-u$ 間の下界値を

$$l_{(q,u)}\left(\{u_{\xi(m)}\}\cup\mathcal{P}_H\right)=\max\left\{l_{(q,u)}(\{u_{\xi(m)}\}),l_{(q,u)}(\mathcal{P}_H)\right\}$$

ととる方式である．3種類目の検索は，基底ピボットすなわち従来手法と同等かそれ以上の枝刈りが実現できる．

2種類目の検索を pro1，3種類目の検索を pro2 とする．基底ピボットの選択方法と検索方法の組み合わせにより，表4.3に記述してあるインデキシング・検索方法が可能となる．

計算コスト

カーネル計算の多くは高コストである．例えば，構文木カーネル [9] の計算コストがノード数 N_1,N_2 に対し $O(N_1N_2)$ である．ゆえに，検索の計算コストとしてカーネルの計算回数を考える．

検索全体で必要なカーネルの計算回数を総カーネルコストと呼ぶことにする．総カーネルコストを，総カーネルコスト = 下界計算のためのカーネルコスト + 枝

表 4.3: カーネルピボット学習法が可能なインデキシング・検索方法

$\xi()$ の選択方法 (既存法)	検索方法	
	pro1	pro2
Rnd	Rnd+pro1	Rnd+pro2
Outlier	Outlier+pro1	Outlier+pro2
Maxmin	Maxmin+pro1	Maxmin+pro2
BNC inc.	BNC inc.+pro1	BNC inc.+pro2

刈りできないオブジェクトに対するカーネルコスト と分解する．それぞれのコストに対するカーネルの計算回数をみていく．

4.3.1 節で述べたように，各ピボットは $M(=H)$ 個の基底ピボットを共有する．クエリ q から M 個の基底ピボットへのカーネル値 $\{\kappa(q, u_{\xi(m)})\}$ と q 自身のカーネル値 $\kappa(q,q)$ とを計算し，インデキシング情報を利用することで，全ての距離下界 $\{l_{(q,u)}(u_{\xi(m)})\}$，$\{l_{(q,u)}(\mathcal{P}_H)\}$ を算出できる．したがって，距離下界全てを算出するには $1+M$ 回のカーネル計算が必要となる．この値が，下界計算のためのカーネルコストとなる．

一方，枝刈りできないオブジェクト $u \in \mathbb{U}$ に対しては，実際に $d(q,u)$ を計算する必要がある．$\kappa(q,q)$ は下界の計算で計算済みなので，枝刈りできないオブジェクト u に対し $\kappa(q,u)$ を計算すれば，インデキシング情報を利用することで $d(q,u)$ を計算できる．

前述の 1，2，3 種類目の検索は，枝刈りできないオブジェクトの数を $|\{l_{(q,u)}(\{u_{\xi(m)}\}) \leq r\}|$，$|\{l_{(q,u)}(\mathcal{P}_H) \leq r\}|$，$|\{l_{(q,u)}(\{u_{\xi(m)} \cup \mathcal{P}_H\}) \leq r\}|$ とそれぞれ書くことができる．総カーネルコストを $Cost^{base}_{Ker}$，$Cost^{pro1}_{Ker}$，$Cost^{pro2}_{Ker}$ とそれぞれ定義したとき，以下のように計算できる．

- $Cost^{base}_{Ker} = 1 + M + |\{l_{(q,u)}(\{u_{\xi(m)}\}) \leq r\}|$
- $Cost^{pro1}_{Ker} = 1 + M + |\{l_{(q,u)}(\mathcal{P}_H) \leq r\}|$
- $Cost^{pro2}_{Ker} = 1 + M + |\{l_{(q,u)}(\{u_{\xi(m)} \cup \mathcal{P}_H\}) \leq r\}|$

計算効率の比較指標

近傍検索の計算効率を測る比較指標を 2 つ導入する．

全検索に対する効率を測る絶対的な指標として，計算率（Calc rate）を導入する．

$$\text{計算率} = \frac{\text{総カーネルコスト}}{|\mathbb{U}|} \times 100 \quad [\%]$$

計算率が小さいほど，その手法は検索を高速化しているとわかる．

一方，提案手法が既存手法（基底ピボット）の計算から削減する計算コストを測る相対的な指標として以下の削減率（Pruning rate）を導入する．

$$\text{削減率} = \frac{Cost_{Ker}^{base} - \{\text{提案コスト}\}}{Cost_{Ker}^{base}} \times 100 \quad [\%]$$

{提案コスト}の部分には$Cost_{Ker}^{pro1}$もしくは$Cost_{Ker}^{pro2}$が入る．削減率が大きいほど，提案手法が既存手法に対し検索を高速化しているとわかる．

4.4.4 従来手法との比較評価

従来手法とカーネルピボット学習法との検索時の計算効率を比較する．比較する従来手法は，基底ピボット$\{u_{\xi(\cdot)}\}$による検索とする．すなわち，4.4.3節で述べた“1種類目の検索”とする．比較するカーネルピボット学習法は前述のpro1とpro2とする．従来手法の$\xi(\cdot)$の選択方法は，ランダム選択と，従来手法の中で性能が良いことが[18]で確認されているBNC incrementalとする．BNC incremental法のパラメータの値は，[6]に従い$R = 50$に設定した．また，実験条件を等しくするため，BNC incremental法のオブジェクトのペア集合をデータ集合$\mathbb{U}$の全てのペアとした．

次に，共通の実験設定について記述する．K近傍検索のKが1である，1-NNクエリで評価した．評価尺度には，計算率および削減率を導入し，クエリ全体でとった平均値で評価した．ピボットの数は，実験セット1では10から150，実験セット2では10から100，実験セット3，4では20から200と変化させた．

実験セット1での計算率，削減率によるプロットをそれぞれ，図4.3，図4.4に示す．同様に，実験セット2を図4.5，図4.6，実験セット3を図4.7，図4.8，実験セット4を図4.9，図4.10に示す．ただし，実験セット2では$\sigma^2 = 10$とし，実験セット3，4では，トピック数70として実験をした．

図4.3から図4.10の結果をみると，いずれの場合でも提案手法が従来手法よりも効率的な検索を実現していることがわかる．それぞれ詳細にみていく．

ベクトルデータに基づく，実験セット1，実験セット2による結果は，提案手法（Rnd+pro1，Rnd+pro2，BNC+pro1，BNC+pro2）がほとんどのピボット数で

既存手法（Rnd，BNC incremental）よりも効率的な検索を実現していることがわかる．実験セット1では実験セット2よりも計算率は高いが，その削減率の値はピボット数40以上のとき70%と圧倒的に高い．実験セット1のように実験データが一様に分布し，ピボットがオブジェクト集合間の任意の場所に配置し易いときに，提案手法は既存手法よりもピボットの質を向上させることがわかる．[18]でも同様の結果が観測されている．

係り受け木に基づく，実験セット3，実験セット4による結果は，BNC incremental法の結果が良いため，ベクトルデータに基づく結果とは異なるものになる．BNC incremtal法は，ランダム選択に基づく提案手法（Rnd+pro1，Rnd+pro2）よりも効率的な検索を実現している．これは，基底ピボットを選ぶことの重要性を示している．そのBNC incremental法よりも，BNC incremental法に基づく提案手法（BNC+pro1，BNC+pro2）は，効率的な検索を実現している．カーネルピボット学習法による最適化が上手くいっていることを示している．BNC incremental法の最良のピボット数は図4.7と図4.9ともに120であるが，BNC+pro2は，ピボット数80でそれよりも効率的な検索を実現する．

総カーネルコストは，4.4.3節でも述べたように，$1+M$と，下界で枝刈りできなかったオブジェクトの数との和からなる．両者の間にはトレードオフの関係がある．ピボット数Mが少ないときは，枝刈りできないオブジェクト数が多くなるため，総カーネルコストが高くなる．ピボット数Mが多いときは，枝刈り数が増大するが，一方，M自体負担になり総カーネルコストが高くなる．最適なピボット数は，データの質に依存する．例えば，図4.5の場合，最適なピボット数は40-60である．

カーネルに関するパラメータの影響を調べるため，実験セット2，3を異なるパラメータで実験する．実験セット2に関しては$\sigma^2 = 100$とし実験をする．図4.11，図4.12それぞれに，計算率，削減率に関するグラフをプロットする．得られるグラフの傾向は$\sigma^2 = 10$とほぼ同じとなる．ただ，図4.11の場合，計算率は全体的に低下する．実験セット3に関してはピボット数を40に固定し，トピック数を変化させ実験をする．図4.13，図4.14それぞれに，計算率，削減率に関するグラフをプロットする．得られるグラフから，トピック数が少ないときには，計算率が下がり，提案手法の既存手法に対する削減率が上昇すること読み取れる．一方，トピック数が多いときには，計算率が上昇する．これは，単語間の関係がスパースに近づき，無限次元に近づくことが原因である．そのため，ノードの組み合わせによって構成される部分木の空間は超高次元空間となり，枝刈りが難しい状態となる．したがって，計算率が上昇する．

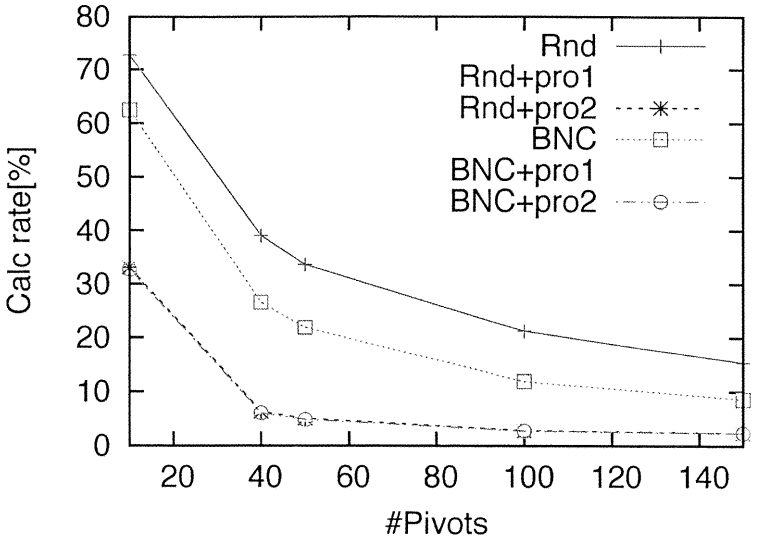


図 4.3: 実験セット 1（ランダムベクトル）で の計算率

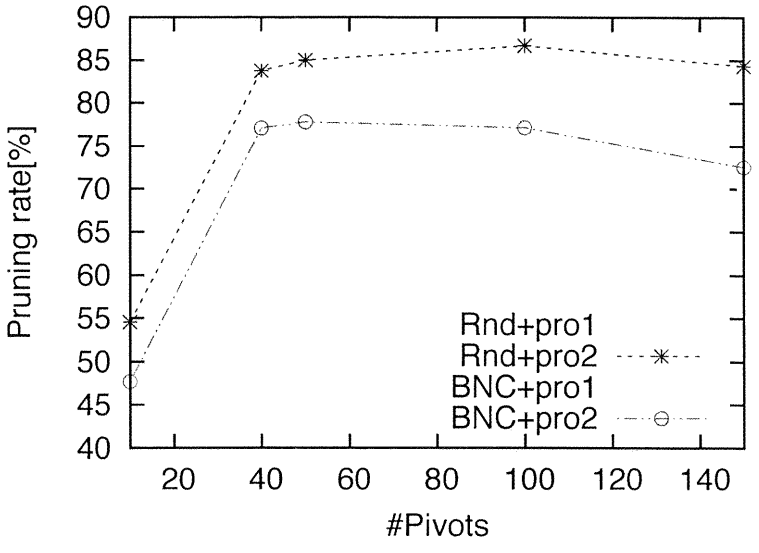


図 4.4: 実験セット 1（ランダムベクトル）での削減率

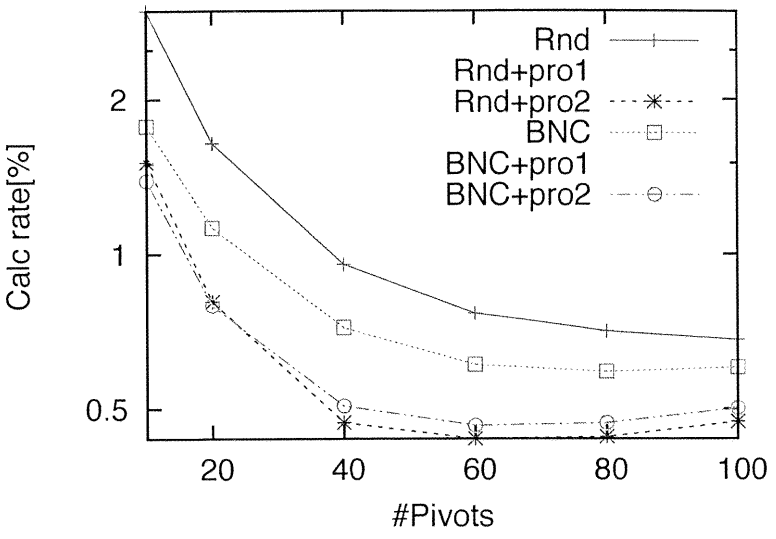


図 4.5: 実験セット 2（画像データ，$\sigma^2 = 10$）での計算率

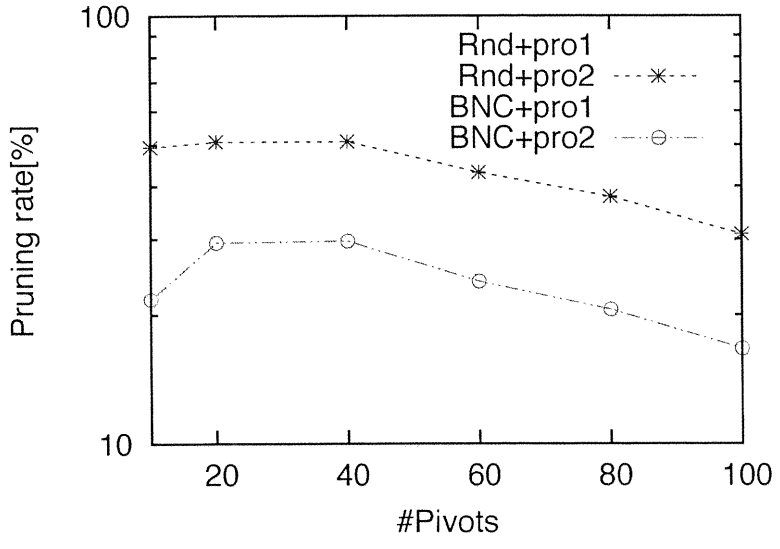


図 4.6: 実験セット 2（画像データ，$\sigma^2 = 10$）での削減率

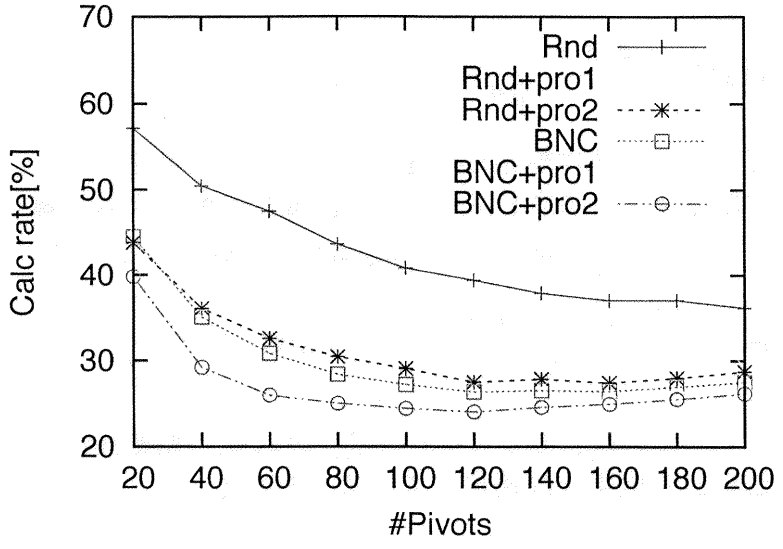


図 4.7: 実験セット 3（係り受け木，トピック数 70）での計算率

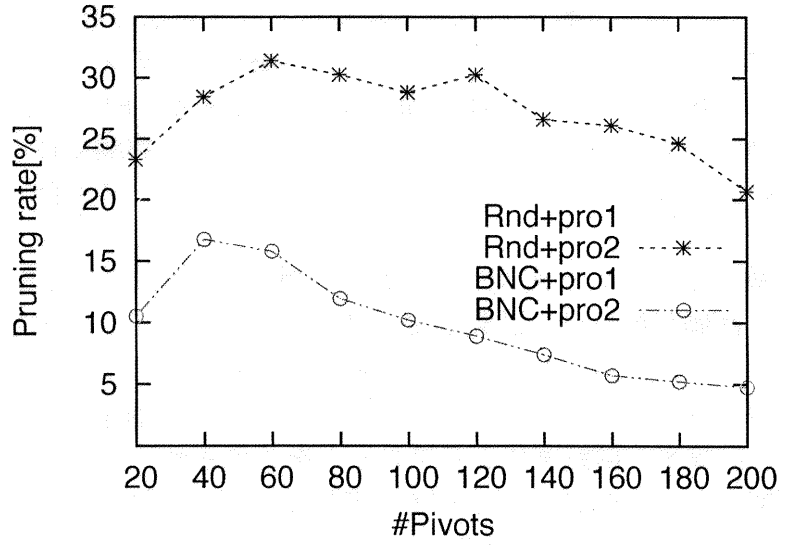


図 4.8: 実験セット 3（係り受け木，トピック数 70）での削減率

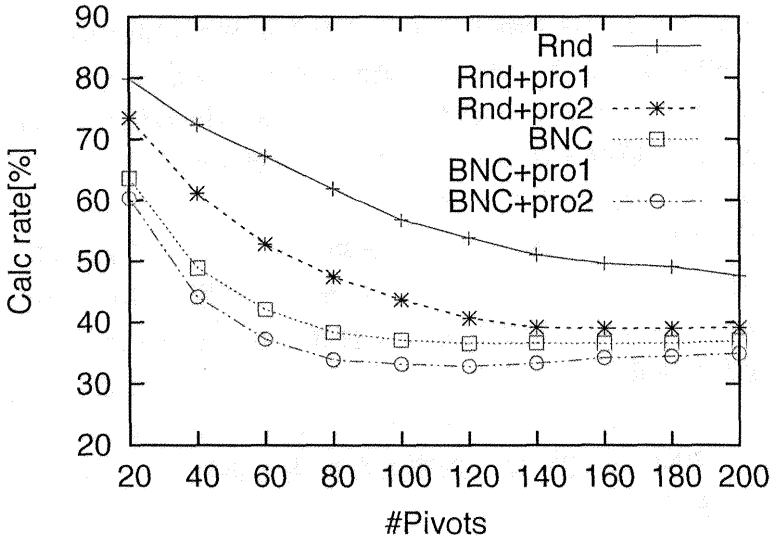


図 4.9: 実験セット 4（係り受け木，トピック数 70，正規化距離）での計算率

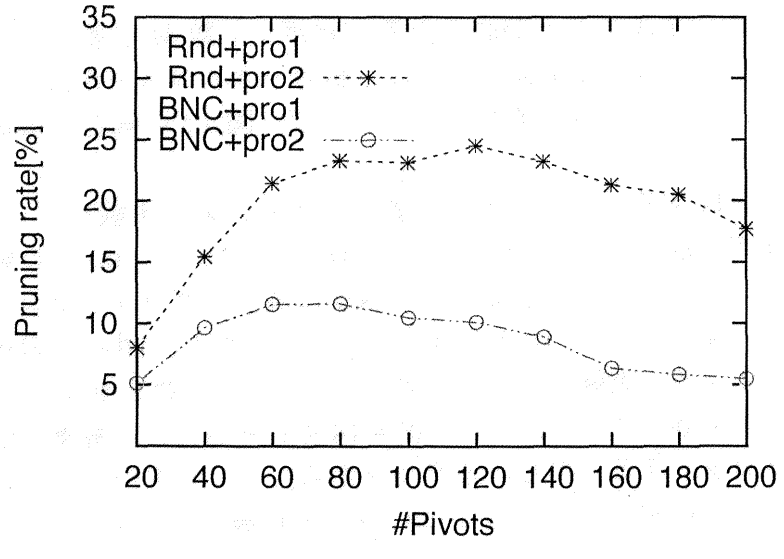


図 4.10: 実験セット 4（係り受け木，トピック数 70，正規化距離）での削減率

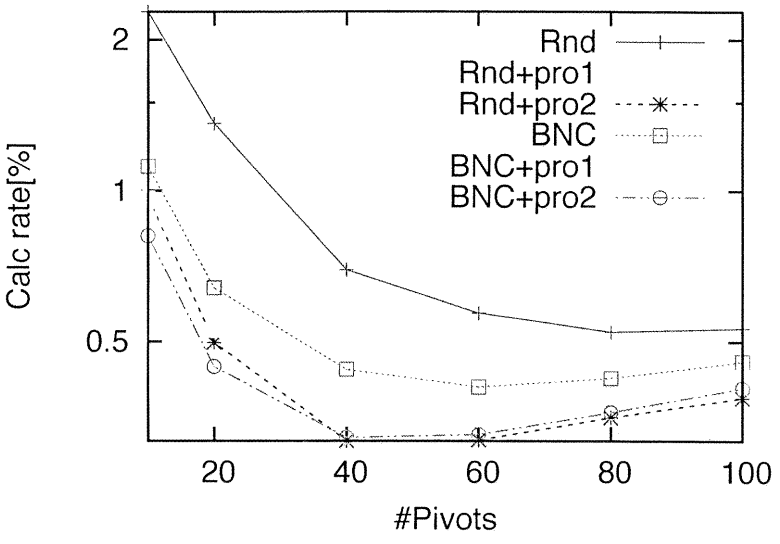


図 4.11: 画像データ，$\sigma^2 = 100$ での計算率

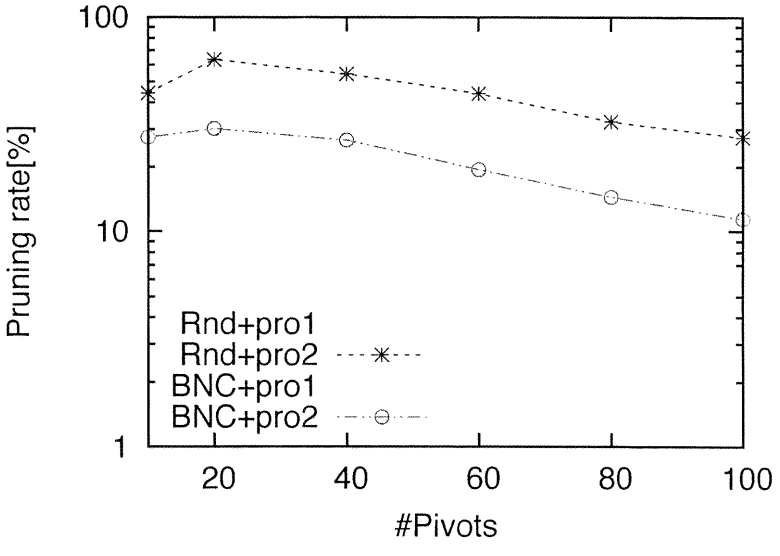


図 4.12: 画像データ，$\sigma^2 = 100$ での削減率

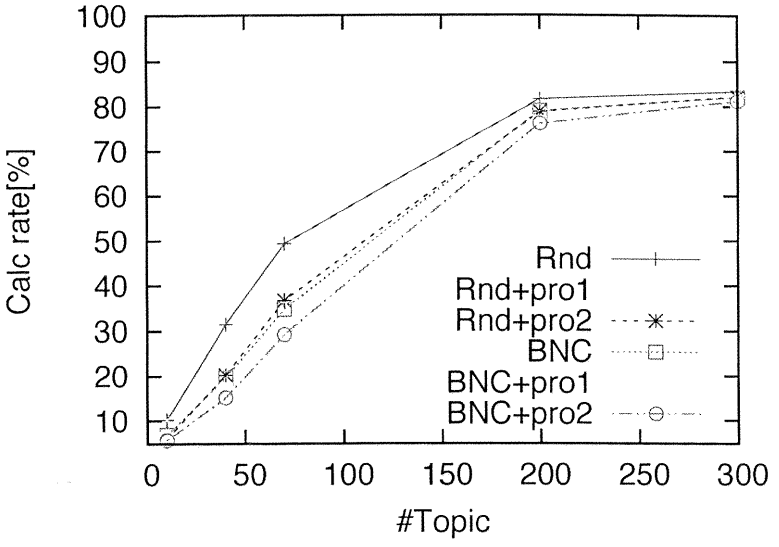


図 4.13: 係り受け木，ピボット数 40 でのトピックに対する計算率

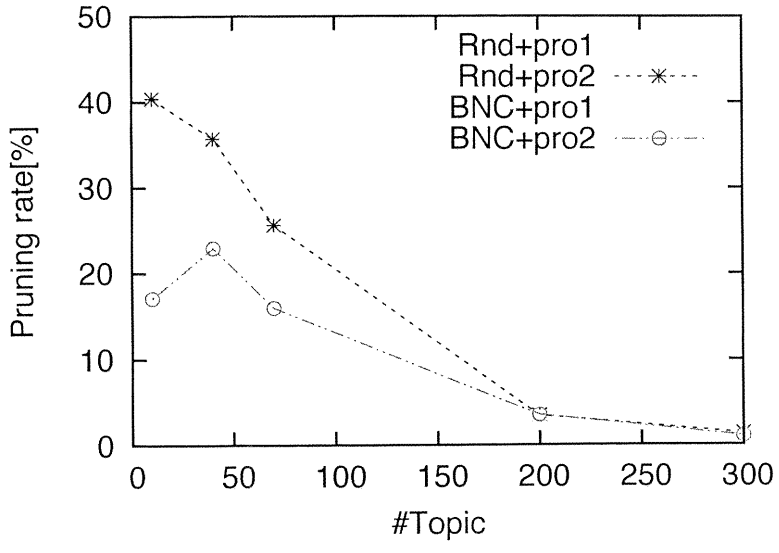


図 4.14: 係り受け木，ピボット数 40 でのトピックに対する削減率

4.5 本章のまとめ

本章では，カーネル関数に基づいた距離関数でのピボット学習法，カーネルピボット学習法を提案した．カーネル関数を正規化させた手法も提案し，ピボット学習法の応用先を広げた．

実験では，人工ベクトルデータ，画像ベクトルデータ，係り受け木データ，以上3種類のデータを使用した．カーネルには通常の内積，ガウシアンカーネル，木カーネルを用いた．その結果はいずれも既存手法よりも効率的な検索を実現した．

第5章　クエリとデータベースとの分布とが異なる場合のカーネルピボット学習法

現実の検索タスクでは，クエリとデータベースとの分布が異なることが多い．例えば，キーワードから文書データベースを検索する場合，キーワードの分布と文書データベースとの分布は異なる．本章では，クエリ集合とデータ集合とを異なるデータ分布と仮定しカーネルピボット学習法を定式化する．実験では，カーネルに木カーネルを導入し，データに構文木データベース (Penn TreeBank) を用いる．クエリのノード数を定数以下に限定し，クエリ集合とデータ集合とを異なるデータ分布に設定し実験をする．木カーネルの性能を調整するパラメータである，減衰係数が小さいときに，既存手法に対して提案手法が有効であることを確認する．

5.1　クエリ集合とデータ集合とを異なるデータ分布と仮定したときのカーネルピボット学習法

5.1.1　距離空間，オブジェクト，ピボット等の定義

$(\mathbb{X}, d)$ をオブジェクトの空間 $\mathbb{X}$ および $\mathbb{X}$ 上に定義された距離関数 d によって定まった距離空間とおく．d は非負性，対称性，三角不等式の性質を満たすとする．距離空間 $(\mathbb{X}, d)$ 上に，クエリとなる集合 $\mathbb{Q} = \{q_1, \cdots, q_i, \cdots, u_{|\mathbb{Q}|} : q_i \in \mathbb{X}\}$ とデータベース $\mathbb{U} = \{u_1, \cdots, u_j, \cdots, u_{|\mathbb{U}|} : u_j \in \mathbb{X}\}$ とが存在するとする．

$\mathbb{X}$ から 写像関数 $\phi(\cdot)$ よって 写像されたヒルベルト空間 $\mathbb{H}$ に，H 個のピボット $\mathcal{P}_H = \{\phi(p_1), \cdots, \phi(p_h), \cdots, \phi(p_H)\}$ を考える．具体的に，それぞれのピボットは，$\mathbb{V} = \{v_1, \cdots, v_m, \cdots, v_{|\mathbb{Q}\cup\mathbb{U}|} : v_m \in \mathbb{Q} \cup \mathbb{U}\}$ から選ばれた M 個のオブジェクト $\mathcal{V}_\xi = \{v_{\xi(1)}, \cdots, v_{\xi(m)}, \cdots, v_{\xi(M)}\}$ (ξ は, $\{1, \ldots, M\} \mapsto \{1, \ldots, |\mathbb{Q} \cup \mathbb{U}|\}$ であり単射関数)

を，ピボット各々のパラメータ $\{\alpha_{h,1},\cdots,\alpha_{h,m},\cdots,\alpha_{h,M} : \alpha_{h,m} \in \mathbb{R}\}$ で重み付けしたものである．

$$\phi(p_h) = \sum_{m=1}^{M} \alpha_{k,m}\phi(v_{\xi(m)}) \tag{5.1}$$

$v_{m_1}, v_{m_2} \in \mathbb{V}$ に対し，写像後の $\phi(v_{m_1}), \phi(v_{m_2})$ による内積 $< \phi(v_{m_1}), \phi(v_{m_2}) >$ は，カーネル $\kappa(v_{m_1}, v_{m_2})$ によって計算可能とする．

5.1.2 最大化したい目的関数

最大化したい目的関数 $F_{\mathcal{B}}(\mathcal{P}_H)$ は以下の定義である．

$$F_{\mathcal{B}}(\mathcal{P}_H) \stackrel{\text{def}}{=} \sum_{i=1}^{|\mathbb{Q}|}\sum_{j=1}^{|\mathbb{U}|} \max_{1\le h\le H} |d(q_i p_h) - d(u_j, p_h)|. \tag{5.2}$$

ここで，$d(q, p_h)$，$d(u_j, p_h)$ は，$\mathbb{H}$ 上の写像間の距離 $d_H(\phi(q), \phi(p_h))$，$d_H(\phi(u_j), \phi(p_h))$ を 4.2 節と同様に簡略化している．

5.2 に対し，

$$\mathcal{S}_h(\mathcal{P}_H) \stackrel{\text{def}}{=} \{(i,j) : h = \arg\max_{1\le h\le H} |d(q_i, p_h) - d(u_j, p_h)|\,, (q_i, u_j) \in \mathbb{Q}\times\mathbb{U}\} \tag{5.3}$$

とおくと，

$$F_{\mathcal{B}}(\mathcal{P}_H) = \sum_{h=1}^{H} \sum_{(i,j)\in\mathcal{S}_h(\mathcal{P}_h)} |d(q_i, p_h) - d(u_j, p_h)| \tag{5.4}$$

を得る．(5.2) は，各 (i,j) で最適な h での和を求めている．一方 (5.3) のように，各 (i,j) に対し最適な h を定義しておけば，(5.4) の計算で (5.2) を求めることができる．つまり等価な表現である．(5.4) の右辺で h に関する足し合わせ $\sum_{h=1}^{H}$ の内側の項を

$$F_{\mathcal{B}}^{(h)}(\mathcal{P}_H) \stackrel{\text{def}}{=} \sum_{(i,j)\in\mathcal{S}_h(\mathcal{P}_h)} |d(q_i, p_h) - d(u_j, p_h)| \tag{5.5}$$

とおく．(5.5) の右辺は，$\{d(q_i, p_h) : i = 1,\cdots,|\mathbb{Q}|\}$ の線形和と $\{d(u_j, p_h) : j = 1,\cdots,|\mathbb{U}|\}$ の線形和と，二つの線形和を足し合わせる形で書き直すことができる．それぞれの線形和の係数を，$\{\mu_h(i) : i = 1,\cdots,|\mathbb{Q}|\}$，$\{\lambda_h(j) : j = 1,\cdots,|\mathbb{U}|\}$ とおく．

$\mu_h(i)$ は $d(q_i, p_h)$ と $\{d(u_j, p_h) : j = 1, \cdots, |\mathbb{U}|\}$ 内の各要素 $d(u_j, p_h)$ との大小関係で計算でき，$\lambda_h(j)$ は $d(u_j, p_h)$ と $\{d(q_i, p_h) : i = 1, \cdots, |\mathbb{Q}|\}$ 内の各要素 $d(q_i, p_h)$ との大小関係で計算できる．

$d(q_i, p_h)$ が $d(u_j, p_h)$ 以上に大きいときの数を $\mu_h^+(i)$，$d(u_j, p_h)$ 以下に小さいときの数を $\mu_h^-(i)$，すなわち，

$$\begin{aligned} \mu_h^+(i) &= \left|\{(i, j) \in \mathcal{S}_h(\mathcal{P}_H) : d(q_i, p_h) \geq d(u_j, p_h)\}\right|, \\ \mu_h^-(l) &= \left|\{(i, j) \in \mathcal{S}_h(\mathcal{P}_H) : d(q_i, p_h) \leq d(u_j, p_h)\}\right|, \end{aligned}$$

とおくと，$\mu_h(i)$ は

$$\mu_h(i) = \mu_h^+(i) - \mu_h^-(i),$$

と計算できる．同様に，$d(u_j, p_h)$ が $d(q_i, p_h)$ 以上に大きいときの数を $\lambda_h^+(j)$，$d(q_i, p_h)$ 以下に小さいときの数を $\lambda_h^-(j)$，すなわち，

$$\begin{aligned} \lambda_h^+(j) &= \left|\{(i, j) \in \mathcal{S}_h(\mathcal{P}_H) : d(u_j, p_h) \geq d(q_i, p_h)\}\right|, \\ \lambda_h^-(j) &= \left|\{(i, j) \in \mathcal{S}_h(\mathcal{P}_H) : d(u_j, p_h) \leq d(q_i, p_h)\}\right|, \end{aligned}$$

とおくと，$\lambda_h(j)$ は

$$\lambda_h(j) = \lambda_h^+(j) - \lambda_h^-(j),$$

と計算できる．したがって，(5.5) は

$$F_{\mathcal{B}}^{(h)}(\mathcal{P}_H) = \sum_{i=1}^{|\mathbb{Q}|} \mu_h(i) d(q_i, p_h) + \sum_{j=1}^{|\mathbb{U}|} \lambda_h(j) d(u_j, p_h) \tag{5.6}$$

と変形できる．

5.1.3 計算アルゴリズム

アルゴリズムの手順を以下に記す．ピボットの初期化についての議論は，4.3.1 節と同様である．

step 1. 表 4.1 のいずれかの方法により，クエリ集合 $\mathbb{U}$ とデータベース $\mathbb{U}$ の和集合 $\mathbb{V} = \mathbb{Q} \cup \mathbb{U}$ から M 個オブジェクトを選び，重み $\{\alpha_{h,1}, \alpha_{h,2}, \cdots, \alpha_{h,M} : h = 1, \cdots, H\}$ を $\alpha_{h,h} = 1, (h = 1, \cdots, H)$，それ以外を 0 とおくことにより，ピボット $\{p_1, \cdots, p_H\}$ を初期化する．

step 2. 以下のステップを収束するまで回繰り返す.

step 2-1. それぞれの p_h について，分割式（5.3）で $S_h(\mathcal{P}_H)$ を求め，$\mu_h(i)$，$\lambda_h(j)$ を計算して目的関数（5.6）を定める.

step 2-2. 準ニュートン法で重み $\{\alpha_{h,1}, \alpha_{h,2}, \cdots, \alpha_{h,M} : h = 1, \cdots, H\}$ を更新し，各ピボット p_h を再計算する．ここでは，準ニュートン法に対し，偏微分 $\{\frac{\partial F_{\mathcal{B}}^{(h)}(\mathcal{P}_H)}{\partial \alpha_{h,\tau}} : \tau = 1, \cdots, M\}$ が必要となる.

5.1.4 偏微分 $\frac{\partial F_{\mathcal{B}}^{(h)}(\mathcal{P}_H)}{\partial \alpha_{h,\tau}}$

カーネル関数を保ったまま，$F_{\mathcal{B}}^{(h)}(\mathcal{P}_H)$ を $\alpha_{h,\tau}$ について偏微分すると以下のようになる.

$$\begin{aligned}\frac{\partial F_{\mathcal{B}}^{(h)}(\mathcal{P}_H)}{\partial \alpha_{h,\tau}} &= \sum_{i=1}^{|\mathbb{Q}|} \frac{\mu_{h,i}}{d(p_h, q_i)} \left(\sum_{m=1}^{M} \alpha_{h,m} \kappa(v_{\xi(m)}, v_{\xi(\tau)}) - \kappa(q_i, v_{\xi(\tau)}) \right) \\ &\quad + \sum_{j=1}^{|\mathbb{U}|} \frac{\lambda_{h,j}}{d(p_h, u_j)} \left(\sum_{m=1}^{M} \alpha_{h,m} \kappa(v_{\xi(m)}, v_{\xi(\tau)}) - \kappa(u_j, v_{\xi(\tau)}) \right) \end{aligned} \tag{5.7}$$

5.2 実験

ここで比較する検索方法や，計算コスト，計算効率の比較指標は，4.4.3節，4.4.3節，4.4.3節と同じである，実験設定や実験結果について述べる.

5.2.1 実験データ

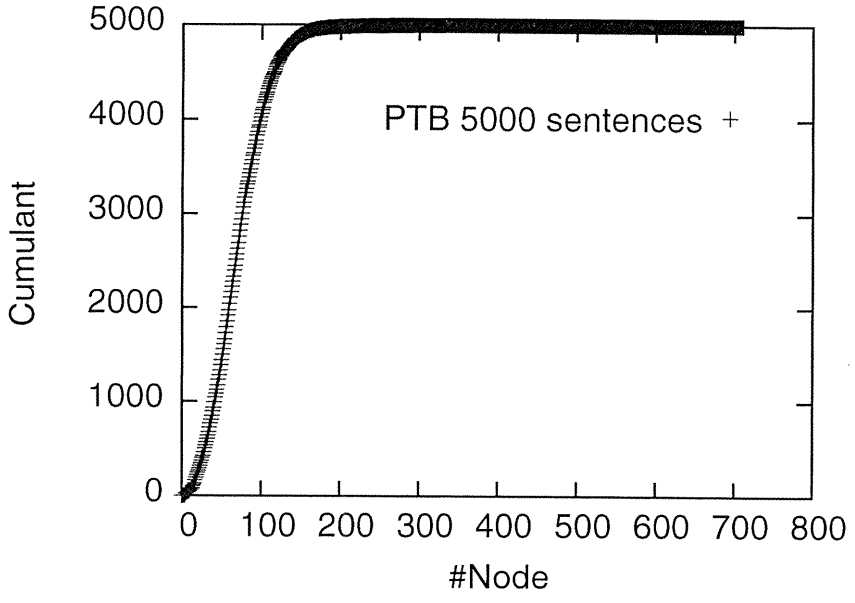


図 5.1: Penn TreeBank 5000 文のノード数のキュムラント

Penn TreeBank からランダムにサンプリングした 5000 文を実験に使う．参考のために，同データのノード数のキュムラント分布を図 5.1 に示す．この分布から，5000 文のデータを，3 種類のデータセットに排他的に分割する．分割したデータは，データベース用のデータ集合，学習用のクエリ集合，検索（テスト）用のクエリ集合として使われる．

まず，5000 文のデータを，データベース用のデータ集合とそれ以外のデータ集合（学習用のクエリ集合，検索（テスト）用のクエリ集合）に分割する．データベース用のデータ集合は，図 5.1 のようにあらゆるノード数を含むものにする．5000 文のうち，ノード数が 68 以上のもの 2487 個（データセット A），ノード数が 67 以下のもの 2513 個（データセット B）に分割する．ノード数が 67 以下の 2513 個（データセット B）をサンプリングすることで，1257 個（データセット BA），1256 個（データセット BB）に分割する．データセット A とデータセット BA とを合わせた 3744 個をデータベース用のデータ集合とする．

データベース用のデータ集合として使われなかったデータセット BB は，クエリ用の集合（学習用のクエリ集合，検索用のクエリ集合）である．学習用のクエリ集合と検索用のクエリ集合の間でデータに重なりがあると，答えを知った状態

で学習をしてしまうことになる．そこで，データセットBBをランダムにサンプリングし，628個を学習用のクエリ集合に，628個を検索用のクエリ集合に排他的に使用する．

5.2.2 実験結果

実験設定は次のものとする．ピボットに関するパラメータは，基底ピボットの数と線形結合ピボットの数は同じ数（$H = M$）とする．カーネルには構文木カーネル [9] を用い，減衰係数を変化させ実験をする．減衰係数は，カーネルの性能を特徴づけるパラメータである．1に近づくほど部分構造を重視し，0に近づくほど部分構造ではなくノードそのものを重視する．実験では，0.1，0.2，0.3，0.4，0.7，1.0と変化させた．0.4は，現実のタスク [28] に従う数値である．検索のタスクは，最近傍検索 (1-NN) とする．

減衰係数の変化に対する計算率，削減率

ピボットの数を固定し，減衰係数の変化に対する計算率，削減率を調べる．ピボット数$H = M = 20$のときの，計算率のグラフを図5.2，削減率のグラフを図5.3に示す．ピボット数$H = M = 40$のときの，計算率のグラフを図5.4，削減率のグラフを図5.5に示す．

また，ピボット数$H = M = 20$のとき，基底ピボット+線形和で導出したピボットは，ピボット数40とみなすことができる．ピボット数$H = M = 20$のときの基底ピボット+線形和で導出したピボット，ピボット数$H = M = 40$のときの基底ピボット，ピボット数$H = M = 40$のときの線形和で導出したピボットを計算率の比較のグラフ：図5.6，削減率のグラフ：図5.7に示す．

図5.3，5.5，5.7より，減衰係数が小さいとき（減衰係数0.1 – 0.4）に，基底ピボットに対する削減率が0より大きいことから，提案手法が既存手法よりも高速に検索できると言える．現実のタスク [28] に従う数値0.4でもそれを達成している．

また，図5.2，5.4，5.6から，計算率は，減衰係数の値が0.4 – 0.7の間にピークがあり，そのピークから離れていくと，計算率は低くなる．特に，減衰係数を1としたとき，計算率が最も低い結果が得られる．しかしながら，そのとき計算率の値は既存手法と同等である．次節では，既存手法のみでも有効な結果が得らる原因を調査するため，追加実験を行う．

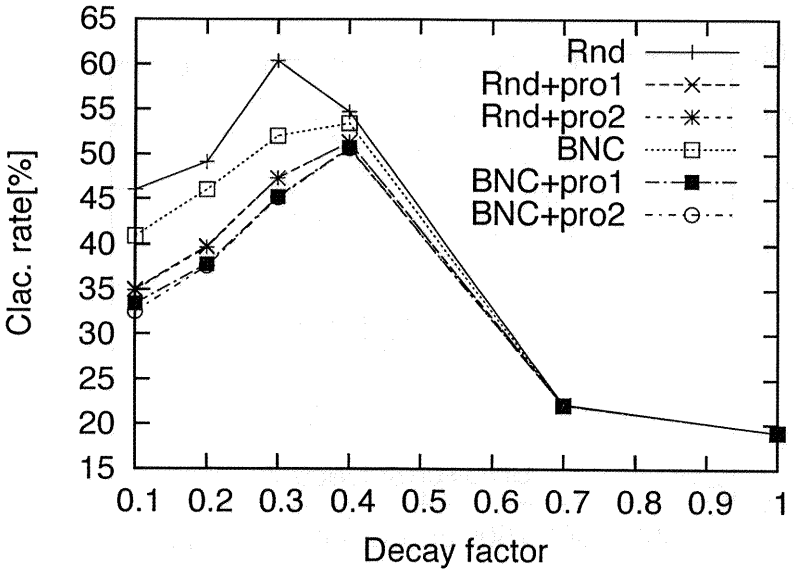


図 5.2: 減衰係数-計算率（ピボット数 20）

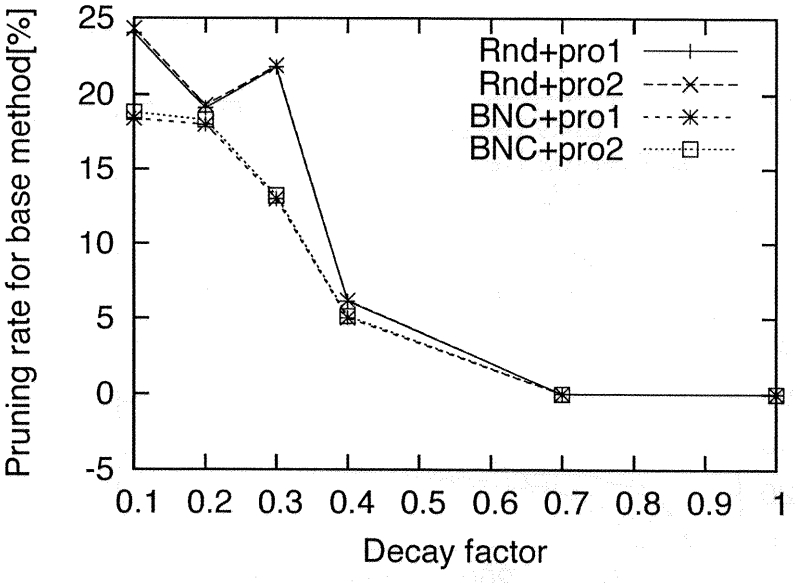


図 5.3: 減衰係数- 基底ピボットに対する提案手法の削減率（ピボット数 20）

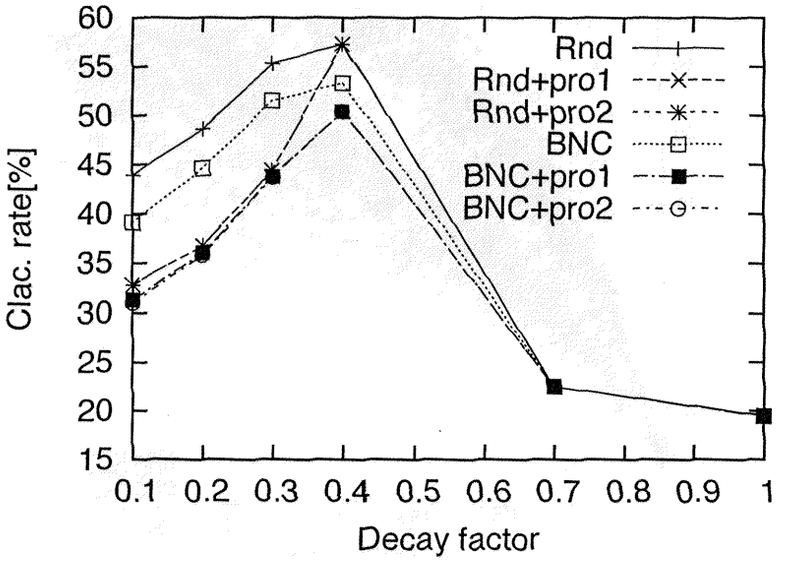


図 5.4: 減衰係数-計算率（ピボット数 40）

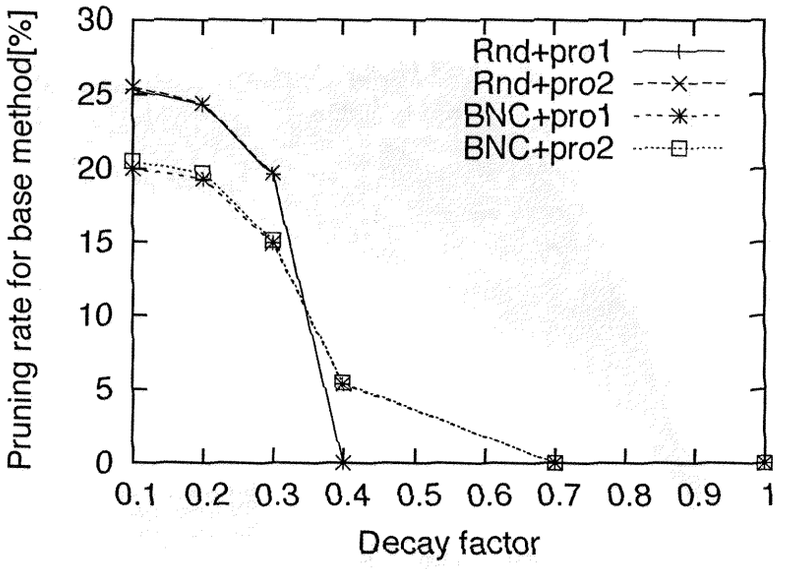


図 5.5: 減衰係数-基底ピボットに対する提案手法の削減率（ピボット数 40）

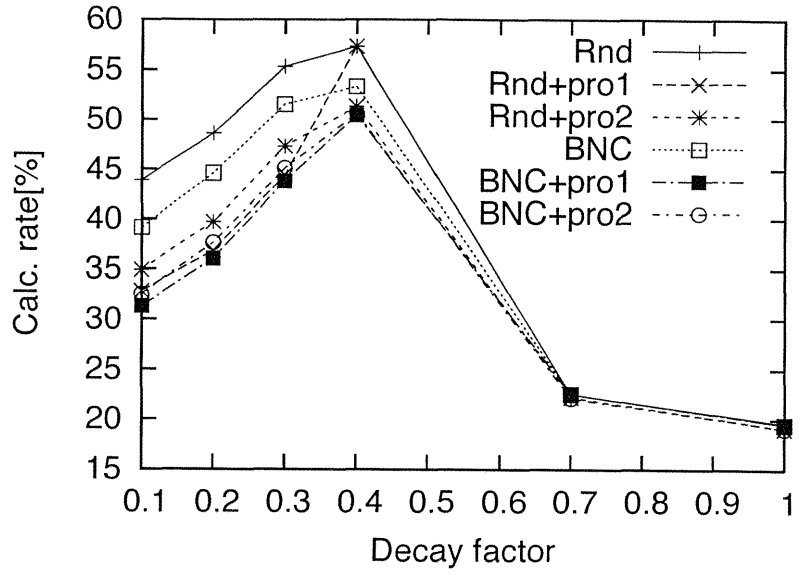


図 5.6: 減衰係数-計算率（基底ピボット 40，線形結合ピボット 40，基底ピボット 20 ＋線形結合ピボット 20 の比較）

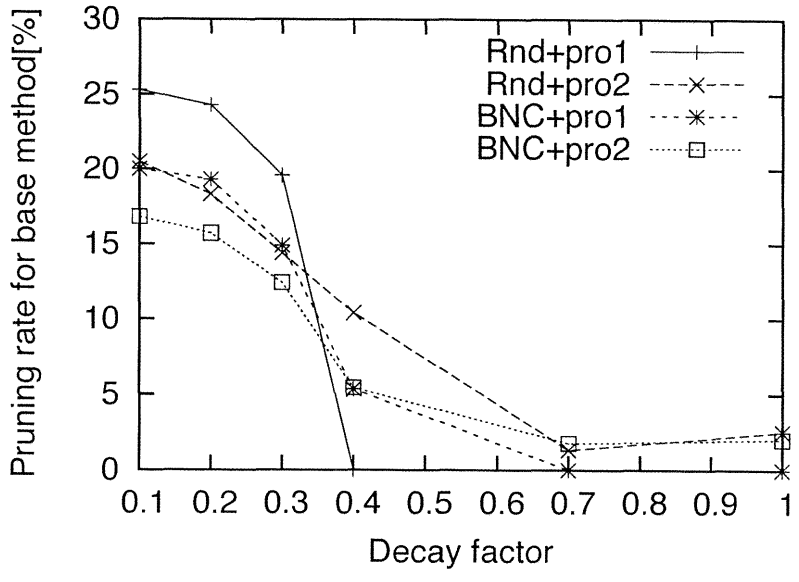


図 5.7: 減衰係数-基底ピボットに対する提案手法の削減率（基底ピボット 40，線形結合ピボット 40，基底ピボット 20 ＋線形結合ピボット 20 の比較）

5.3 追加実験

節 5.2.2 の結果を導く要因を調査するために，本節では様々な実験を行う．

5.3.1 カーネルの計算回数に関する分布

本節では，カーネルの計算回数を軸に，減衰係数ごとのクエリに関する情報量の傾向をみていく．

カーネルの計算回数に関するクエリのキュムラント分布を図 5.8，図 5.9 にプロットする．図 5.9 は図 5.8 のカーネルの計算回数に関する区間 [0,500] をズームアップしたものである．

図 5.8，5.9 のキュムラント分布はカーネルの計算回数と比例関係に近い．ノード数に関するキュムラント分布の図，図 5.1 においても，ノード数 67 個以下では，ノード数とキュムラント分布とは比例関係に近い．このことから，カーネルの計算回数とクエリのノード数の傾向をみていくことにする．図 5.10，5.11 に，それぞれ，既存法（BNC 法），提案法（BNC 法+提案法 2）による，各ピボットの＜カーネル計算回数，ノード数＞の関係をプロットする．

図 5.10，5.11 を概観すると，減衰係数毎に，分布に傾向があることがわかる．減衰係数 0.4 のときに最も相関が強い傾向が見られ，減衰係数 0.7，0.1 はそれよりも相関が弱いことがわかる．特に，0.1 のときは，最も相関が弱い．

次に，図 5.10 と図 5.11 とを比較し，提案法 2 を適用することで，プロットの集団がどのように変化するか観察する．減衰係数 0.1 のとき，図 5.11 のプロットの集団は図 5.10 よりも全体が左に移動している．例えば，プロットの密度は，カーネル計算の区間 [0.500] で高くなっているのに対し，区間 [2500,3000] で低くなっている．一方，減衰係数 0.4 のとき，集団はあまり動いていない．減衰係数 0.7 のときは，面白い傾向が出ている．図 5.10 で，二股の木のように，原点から 2 つに分かれている集団のうち，片方の計算回数が少ない集団が図 5.11 ではその分布に変化がないのに対し，もう片方の計算回数が多い集団は計算回数が少ない集団に近づいている．

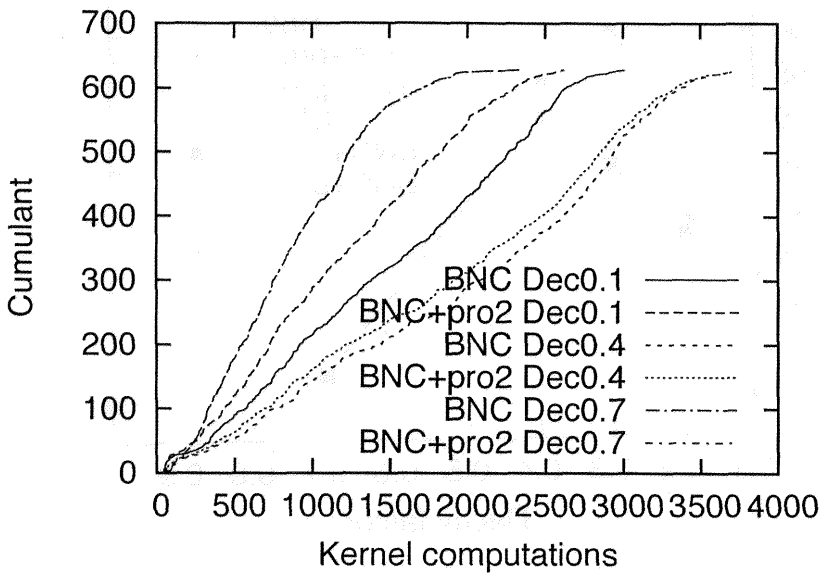


図 5.8: BNC 法に基づくインデックスでの計算回数のキュムラント（基底ピボット 40，基底ピボット 40 ＋線形結合ピボット 40）

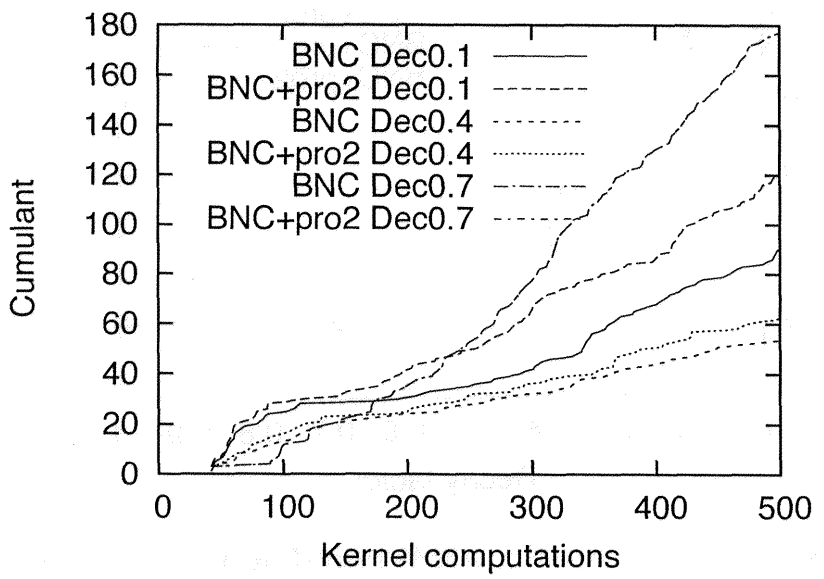


図 5.9: 区間 [0,500] を拡大した図 5.8

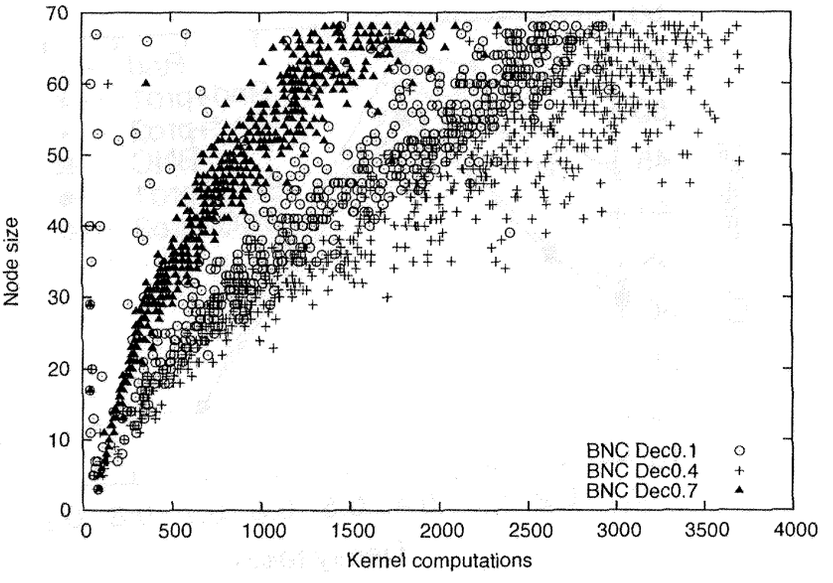


図 5.10: BNC 法によるインデックスでの各クエリに対する＜計算回数，ノード数＞のプロット（ピボット 40）

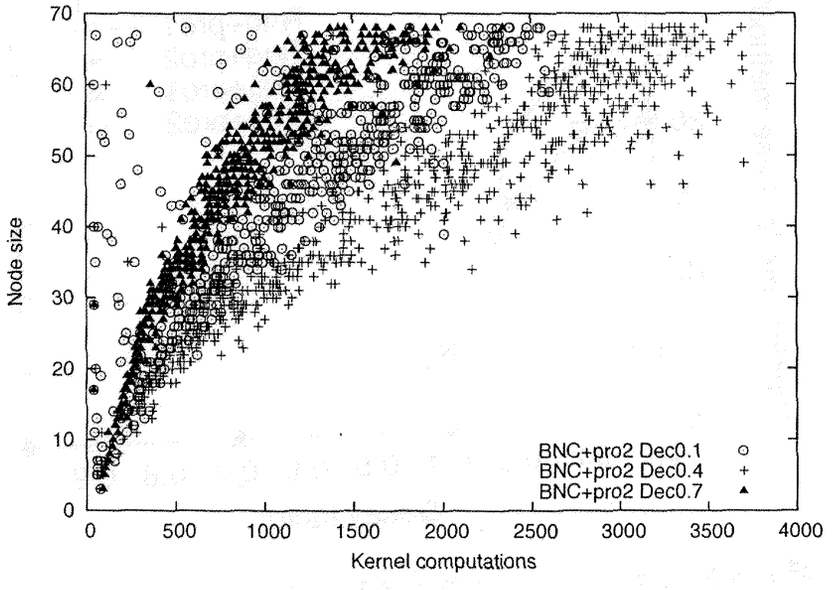


図 5.11: BNC 法+提案法 2 よるインデックスでの各クエリに対する＜計算回数，ノード数＞のプロット（ピボット 40）

5.3.2 各減衰係数の結果

本節では，各減衰係数ごとに，以下を調べる．

1. クエリとクエリに対するデータベースの最近傍の距離

2. 最近傍検索をしたときに，クエリがデータベース内の同一のデータを最近傍として参照する回数

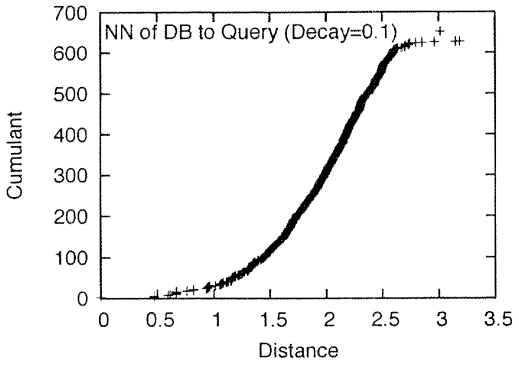
1. に関しては，キュムラント分布としてまとめる．2. に関しては，横軸にデータベース内の各データがクエリから最近傍として参照される回数，縦軸にその頻度をとるヒストグラムとしてまとめる．ただし，2. では，クエリから最近傍としての参照回数が0となるデータベース内のデータの頻度はプロットしない．つまり，2 に関するヒストグラムの累積数は全クエリ数と一致する．

減衰係数 0.1，0.4，0.7，1.0 の変化に対し，1. に関するキュムラント分布の変化を図 5.12 に，2. に関するヒストグラムを図 5.13 にまとめる．

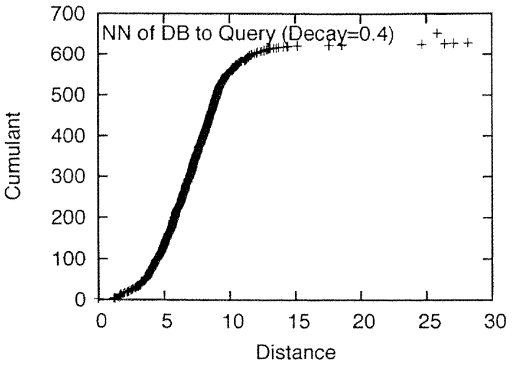
減衰係数 1.0 の場合を例にとり，減衰係数が高いときに基底ピボットのみの検索（従来手法）でさえ，著しい高速化が可能になる理由を考察する．図 5.13(d) より，全クエリ 625 個のうち約 350 個そして 150 個から最近傍として検索されるデータベース内のデータが一つずつ存在することがわかる．つまり，大半のクエリは，この 2 つのデータを近傍，それ以外のデータを遠方にあるものとして分布している．最近傍に関する枝刈りは

$$\left|\text{クエリからピボットへの距離} - \text{データからピボットへの距離}\right| > \text{クエリから最近傍への距離} \qquad (5.8)$$

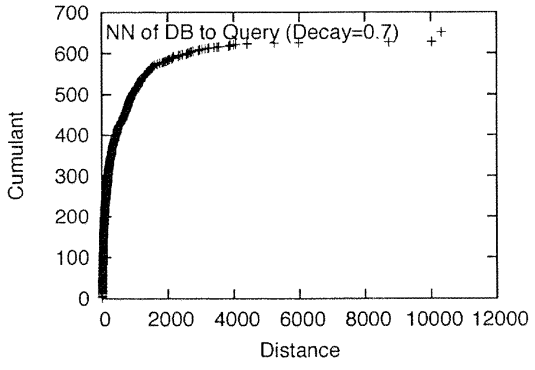
により行われる．減衰係数 1.0 の場合，クエリから最近傍への距離は他よりも近いため，式（5.8）の右辺は小さくなり，左辺が近似したい，クエリ－データ間の距離は多くのクエリ－データペアで大きくなる．実際，基底ピボットとして選ばれるピボットはノード数が小さいものばかり選ばれていた．また，データベースのデータのノード数は，クエリのノード数のよりも多い．そのため，多くのクエリ－データペアで，クエリからピボットへの距離は必然的に小さくなり，一方で，データからピボットへの距離も必然的に大きくなる．以上，式（5.8）の条件式を満たす，クエリ－データの数が多いと推測されるため，基底ピボットだけで計算率が低いと考えられる．



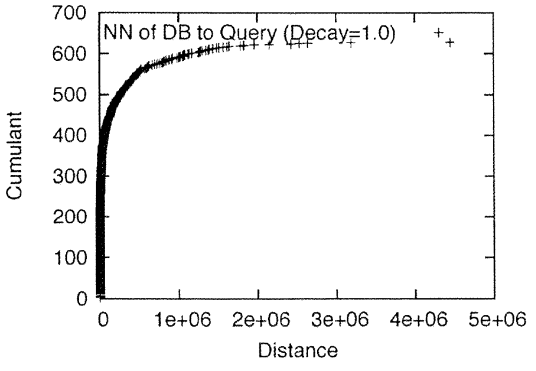
(a) 減衰係数=0.1



(b) 減衰係数=0.4

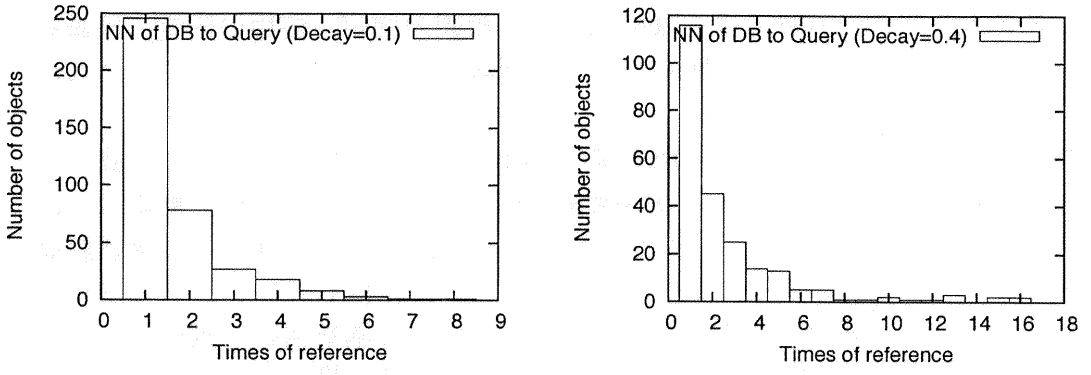


(c) 減衰係数=0.7



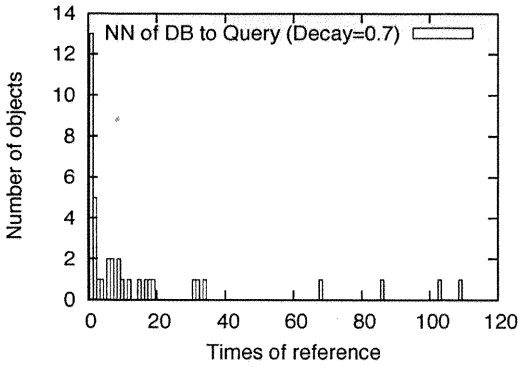
(d) 減衰係数=1.0

図 5.12: クエリとの最近傍距離のキュムラント分布

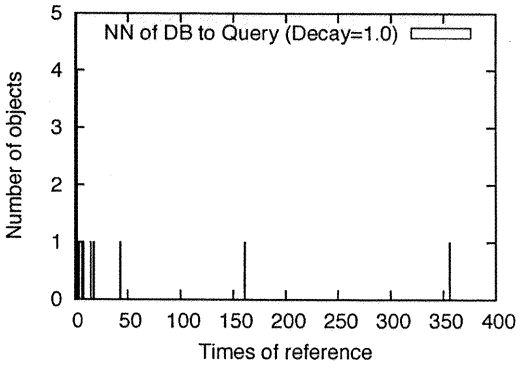


(a) 減衰係数=0.1

(b) 減衰係数=0.4



(c) 減衰係数=0.7



(d) 減衰係数=1.0

図 5.13: クエリから最近傍して参照された回数のヒストグラム（ピボット数 K=M=20）

5.4 本章のまとめ

本章では，クエリ集合とデータ集合とを異なるデータ分布と仮定したときのカーネルピボット学習法を定式化し，木カーネルによる構文木データベース (Penn TreeBank) に対する実験結果を示した．減衰係数が小さいときに，既存手法に対して提案手法が有効であることを示した．

第6章　おわりに

本論文は，近傍検索を高速化することを目的に，ピボットアプローチによるインデキシング手法を研究した．ピボットアプローチが検索を効率化する鍵はピボットの位置にある．ピボットの探索範囲をデータベース上のオブジェクト集合から距離空間全体に拡大することを提案し，近傍検索を高速化する可能性を広げた．

探索範囲を拡大したことで，現実的な時間でピボットを構成することが可能かどうか検証する必要があった．そこで，距離区間全体からピボットを構成する問題に対し，ピボット学習法を提案した．ピボット学習法が出力するピボットは，オブジェクト集合から選択する既存手法に比べ，効率的な検索を実現することができた．つまり，効果的なピボットを距離空間全体から同定することが十分に可能であることを示した．

ピボット学習法がとるアプローチは，機械学習手法に基づく．オブジェクトのペアに関する距離下界の総和を目的関数とし，それをピボット集合について最適化する．最適化の方法は，*K*-means クラスタリング法に類似しており，以下のメリットを持つ．

- 並列処理が容易
- 追加データへの対応が容易

両者は，ピボットをインクリメンタルに選ぶ既存手法の問題点を克服する．並列化が容易であることは，コア数が多い現在の計算機環境に適合する．実際，本論文に掲載した実験結果も並列処理の結果得られたものである．

ピボット学習法は，ユークリッド距離を扱い，ニュートン法，準ニュートン法を用いることで，距離空間内から効率的にピボットを探索する．さらに，様々な構造データを扱えるように，適用可能な距離尺度をユークリッド距離からカーネル関数に基いて定義された特徴空間上の距離に拡張した（カーネルピボット学習法）．両手法（ピボット学習法，カーネルピボット学習法）とも，実験により有効性を確認した．

現実の検索タスクでは，クエリとデータベースとの分布が異なることが多い．クエリとデータベース上のオブジェクトの分布が異なることを仮定した拡張も行っ

た（クエリカーネルピボット学習法）．ノード数が少ない木をクエリ，ノード数が多い木をデータベースとし，木カーネルを用い実験を行い，有効性を確認した．

今後の研究課題について述べる．4種類の課題を考えている．

一つ目の課題は，アルゴリズムを大域的最適化するものに変更することである．本論文で提案したピボット学習法は，局所最適化をするのみである．通常のK-means クラスタリング法と同様に，ピボットの初期位置次第で結果は変わってくる．大域的最適化するアルゴリズムを提案し，この問題を解決していきたい．

二つ目の課題は，性能保証を与えてやることである．今のところピボット学習法には，検索時間や検索コストに対する理論的な保証はない．また，学習時の収束の早さに対する保証もない．後者に関しては，提案手法とアルゴリズムが類似したEMアルゴリズムを参考に保証を与える定理をつくっていきたい．

三つ目の課題は，近似近傍検索手法 [1, 20, 19, 38] との比較である．その中でも，カーネルピボット学習法を共通項の多い [20, 19] と比較実験したい．[20, 19] は，カーネルピボット学習法と同様にカーネル関数に特化し，カーネル関数についての重み付き線形和をとる．一方で，下界による枝刈りを行っておらず，構造データについての実験を論文中で行っていない．構造データについて，[20, 19] とカーネルピボット学習で比較実験を行い，実験結果からより優れた手法をさらに提案していきたい．

近傍検索は，様々な分野で使われている．四つ目の課題は，ピボット学習法の有効性をそれらの分野で確認することである．以下では，ピボット学習法の有効性を確認したい分野について，近傍検索の適用例を説明する．情報検索，機械学習，データマイニング，推薦システムについて説明する．

情報検索の分野では，コンテンツ検索，スペル訂正などで用いられる．コンテンツ検索は，画像，CAD，文書，時系列情報，などを対象に，コンテンツが類似したデータを検索する．画像に関しては，近年，多くのサービスが開始されている[1]，スペル訂正は，文字列データを対象に行う．ユーザ書いた文字列に類似した文字列を辞書中の単語列から同定することで行う．

機械学習の分野では，K-Nearest Neighbor 分類，関数近似，クラスタリング，スペクトラルクラスタリング，manifold learning，link analysis などで用いられる．K-Nearest Neighbor 分類は，未知のデータを分類する，機械学習の分野での標準的な分類手法である．訓練データの中から未知のデータに対する近傍 K 個同定し，その近傍 K 個のデータのラベル及び距離情報を基に，未知のデータのラベルを決定する．関数近似は，入力変数に近い基底群を同定し，その基底群を基に出

[1] TineEye: http://www.tineye.com/, GazoPa:http://www.gazopa.com/, Google similar images labs: http://similar-images.googlelabs.com/

力を計算する．入力に近い基底を同定する際に用いられる．クラスタリング，例えば，*K*-means クラスタリング法 [12] では，各データは複数のオブジェクトの中から一番近いセントロイドと呼ばれるオブジェクトを同定する．セントロイドを同定する際に用いられるこのセントロイドの同定は，収束するまで何度も行われる．スペクトラルクラスタリング，manifold learning，link analysis などの分野では，*K*-NN グラフを作成する必要がある．*K*-NN グラフを高速につくることは，これらの分野の研究で重要な課題である．

データマイニングの分野では，近傍検索が，複製の同定，盗用の同定，同義語の同定に用いられる．近似近傍検索手法を，同義語の同定に行う研究も進められている．

推薦システムの分野では，近傍検索は商品をユーザに推薦するために，商品に対する近傍検索，ユーザに対する近傍検索を行う．商品に対する近傍検索は，前述のコンテンツ検索と同様のことを行う．ユーザに対する近傍検索は，購買履歴が類似したユーザを検索する．手順としては，まず，推薦したいユーザの購買履歴をクエリとして，莫大な顧客のデータベースの中から購買情報が類似したユーザを検索する．そして，検索したユーザが購入しているが推薦したいユーザが購入していない商品を，推薦したいユーザへの推薦商品として決定する．

以上の分野に限らず，様々な分野でピボット学習法の有効性を確認していきたい．

付録 A　距離関数

この章では代表的な距離尺度について述べる．

A.1　Minkowski 距離（L_c 距離）

Minkowski 距離（L_c 距離）は，ベクトル空間のデータに対して定義される距離であり，パラメタ c により，その形が変化する．

2 つの n 次元ベクトルデータ $\forall x, y \in \mathbb{R}^n$ を考える．各ベクトルの要素は，$\boldsymbol{x} = \{x_1, x_2, \cdots, x_n\}$，$\boldsymbol{y} = \{y_1, y_2, \cdots, y_n\}$ と表されるものとする．$\boldsymbol{x}$ と $\boldsymbol{y}$ の Minkowski 距離 $d(\boldsymbol{x}, \boldsymbol{y})$ は以下のように定義される．

$$d(\boldsymbol{x}, \boldsymbol{y}) = \left\{ \sum_{i=1}^{n} (x_i - y_i)^c \right\}^{1/c} \tag{A.1}$$

$c = 1$ のとき，以下のマンハッタン距離となる．

$$d(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{n} |x_i - y_i| \tag{A.2}$$

$c = 2$ のとき，以下のユークリッド距離となる．

$$d(\boldsymbol{x}, \boldsymbol{y}) = \left\{ \sum_{i=1}^{n} (x_i - y_i)^2 \right\}^{1/2} \tag{A.3}$$

ユークリッド距離は，正規化すると以下の形になる．

$$d(\boldsymbol{x}, \boldsymbol{y}) = \left\{ 2 - 2\frac{\boldsymbol{x}^T \boldsymbol{y}}{\|\boldsymbol{x}\|^{1/2} \|\boldsymbol{y}\|^{1/2}} \right\}^{1/2} \tag{A.4}$$

ここで式中の $\frac{\boldsymbol{x}^T \boldsymbol{y}}{\|\boldsymbol{x}\|^{1/2} \|\boldsymbol{y}\|^{1/2}}$ は，コサイン類似度である．ユークリッド距離は距離尺度（小さい程似ている）であり，コサイン類似度は類似度尺度（大きい程似ている）という違いがあるが，その関係に対応がとれる．

$c = \infty$ のとき，以下のように，チェビシェフ距離となる．

$$d(\boldsymbol{x}, \boldsymbol{y}) = \max_{i=1,\cdots,n} |x_i - y_i| \tag{A.5}$$

各次元で，$\boldsymbol{x}$ と $\boldsymbol{y}$ の差が最も大きい値を関数の出力とする．

A.2 マハラノビス距離

2つの n 次元ベクトルデータ $\forall x, y \in \mathbb{R}^n$ を考える．$n \times n$ の半正定値行列 A に対して，マハラノビス距離は，以下のように書ける．

$$d_A(\boldsymbol{x}, \boldsymbol{y}) = \left\{(\boldsymbol{x} - \boldsymbol{y})^T A(\boldsymbol{x} - \boldsymbol{y})\right\}^{1/2} \tag{A.6}$$

近年，マハラノビス距離は，機械学習コミュニティの中の Metric Learning という分野で，距離を学習するモデルとしてよく扱われている [15, 34]．データやデータ間のラベル，データの性質から，A をパラメタとみなし学習する．

また，A の対角成分以外を 0 とおくと，重み付きユークリッド距離となる．

$$d_A(\boldsymbol{x}, \boldsymbol{y}) = \left\{\sum_{i=1}^{n} a_{i,i}(x_i - y_i)^2\right\}^{1/2} \tag{A.7}$$

A.3 カーネルに基づく距離

カーネル [4] とは，写像関数による内積を写像をせずに効率的に計算する類似度関数である．カーネルは木や文字列，グラフ等の構造データも扱うことができる．構造データに対するカーネルの定義は，4.1.1 節でも説明する．まず，カーネルの説明をし，次に，カーネルに基づく距離を説明する．

$\mathbb{X}$ をオブジェクトの空間とする．$\mathbb{X}$ からヒルベルト空間 $\mathbb{H}$ への写像関数を $\phi(\cdot)$ とおく．オブジェクト x, $y \in \mathbb{X}$ は，$\mathbb{H}$ 上に $\phi(x)$, $\phi(y)$ と写像される．カーネルを $\kappa(,)$ とおく．x, y に対して，カーネル $\kappa(x, y)$ は，次のように定義される．

$$\kappa(x, y) = < \phi(x), \phi(y) > \tag{A.8}$$

ここで，$< \phi(x), \phi(y) >$ は，$\phi(x)$，$\phi(y)$ に対する内積を表す．カーネル $\kappa(x, y)$ は，写像 $\phi(x)$，$\phi(y)$ 及び 内積 $< \phi(x), \phi(y) >$ を実際に計算せずに，効率よくその値を計算する．

$\mathbb{H}$ 上で L_2 距離 $d_{\mathbb{H}}(\cdot,\cdot)$ を定義する．次のように定義する．

$$\begin{aligned} d_{\mathbb{H}}(\phi(x),\phi(y)) &= \sqrt{\|\phi(x)-\phi(y)\|} \\ &= \left\{ <\phi(x),\phi(x)> + <\phi(y),\phi(y)> -2<\phi(x),\phi(y)> \right\}^{\frac{1}{2}} \end{aligned} \tag{A.9}$$

$d_{\mathbb{H}}(\phi(x),\phi(y))$ はカーネル $\kappa(\cdot,\cdot)$ を用いることで次のように計算できる．

$$\begin{aligned} d_{\mathbb{H}}(\phi(x),\phi(y)) &= \sqrt{\|\phi(x)-\phi(y)\|} \\ &= \left\{ \kappa(x,x) + \kappa(y,y) - 2\kappa(x,y) \right\}^{\frac{1}{2}} \end{aligned} \tag{A.10}$$

これをカーネルに基づく距離とする．

また，カーネルに基づく距離は，マハラノビス距離を一般化したものとみなせる．2つの n 次元ベクトルデータ $\forall \boldsymbol{x},\boldsymbol{y} \in \mathbb{R}^n$ に対し，式 (A.6) を分解すると，

$$d_A(\boldsymbol{x},\boldsymbol{y}) = \left\{ \boldsymbol{x}^T A \boldsymbol{x} + \boldsymbol{y}^T A \boldsymbol{y} - 2\boldsymbol{x}^T A \boldsymbol{y} \right\}^{1/2} \tag{A.11}$$

となる．$\boldsymbol{x},\boldsymbol{y}$ に対し，$\boldsymbol{x}^T A \boldsymbol{y}$ は，対称性，半正定値性を満たし，カーネル $\kappa(x,y)$ の性質を満たす．つまり，$\boldsymbol{x}^T A \boldsymbol{y}$ はカーネル関数とみなすことができる．逆に，カーネルに基づく距離はマハラノビス距離を一般化したものとみなせる．

A.4 編集距離 (Levenshtein 距離)

編集距離は，2つの文字列の間での違いを測る距離である．

編集距離を計算するには，以下の操作を一方の文字列に行い，もう一方の文字列と同一になるまで繰り返す．

挿入 文字列に1文字挿入する

削除 文字列から1文字削除する

置換 文字列から1文字別の文字に置き換える

この変換操作の最小回数が編集距離である．

編集距離は，スペルミスを発見，訂正する際に用いられる．

A.5 Jaccard係数

Jaccard係数は，2つの集合の間の類似度を測る尺度である．$\mathscr{A}$ と $\mathscr{B}$ を集合とする．Jaccard係数は，以下のように定義される．

$$\frac{|\mathscr{A} \cap \mathscr{B}|}{|\mathscr{A} \cup \mathscr{B}|} \tag{A.12}$$

このJaccard係数に対する距離は，次のように定義される．

$$d(\mathscr{A}, \mathscr{B}) = 1 - \frac{|\mathscr{A} \cap \mathscr{B}|}{|\mathscr{A} \cup \mathscr{B}|} \tag{A.13}$$

Jaccard係数は，情報検索の分野でよく用いられる [24].

バイナリ値をとるベクトルデータに対して，Jaccard係数に基づく距離（Tanimoto距離）を計算できる．バイナリ値をとるベクトルデータ $\boldsymbol{x}$ と $\boldsymbol{y}$ に対して，Tanimoto距離は以下のように定義される．

$$d(\boldsymbol{x}, \boldsymbol{y}) = 1 - \frac{\boldsymbol{x} \cdot \boldsymbol{y}}{\|\boldsymbol{x}\|^2 + \|\boldsymbol{y}\|^2 - \boldsymbol{x} \cdot \boldsymbol{y}} \tag{A.14}$$

付録B　レンジクエリとKNNクエリとの計算コストの一致

ここではレンジクエリの計算コストとKNNクエリの計算コストの関係性を述べる．データベース中のデータオブジェクトで，クエリqからK番目の最近傍及びそれに対応する距離を$u_{Rank_d(K)}$，$d(q, u_{Rank_d(K)})$とする．$d(q, u_{Rank_d(K)})$をレンジとしてレンジクエリによる検索をする場合，その計算コストは，K個の最近傍を検索するKNNクエリの計算コストと一致する．

それを示すために図B.1を導入する．図B.1には，クエリqからデータベース上のオブジェクトu_iへの距離$d(q, u_{Rank_d(\cdot)})$についてのヒストグラム，クエリqからデータベース上のオブジェクトu_iへの距離の下界$d(q, u_{Rank_l(\cdot)})$についてのヒストグラムが示してある．それぞれのヒストグラムの図B.1中の表記は，前者がReal distance: $Rank_d(\cdot)$，後者がLower bound:$Rank_l(\cdot)$である．$d(q, u_{Rank_l(i)})$の分布は，$d(q, u_{Rank_d(i)})$の下界であるので$d(q, u_{Rank_d(i)})$の分布より左にシフトしている．図中の太い縦の黒い線が，クエリqからK番目の最近傍への距離$d(q, u_{Rank_d(K)})$である．

レンジクエリの検索に必要な計算コストはレンジrに対して枝刈りできないオブジェクト，すなわち，$|d(q, p_h) - d(u_i, p_h)| \leq r$を満足するオブジェクトである．$r = d(q, u_{Rank_d(K)})$で検索したとする．図B.1では，斜線部の面積が$|d(q, p_h) - d(u_i, p_h)| \leq d(q, u_{Rank_d(K)})$を満足するオブジェクトである．レンジクエリでは，斜線部の面積が計算コストとして必ず必要となる．

KNNクエリの場合を説明する．KNNクエリのアルゴリズムにより，下界が小さいオブジェクトから実距離を計算し，アルゴリズムの終了条件である$d(q, u_{Rank_d(K)})(= r)$以下となるオブジェクトまで計算したとする．このとき，検索の間に計算した全てのオブジェクトは，$\{u_{Rank_l(i)} : l(q, u_{Rank_l(i)})) \leq d(q, u_{Rank_d(K)})\}$を満足するオブジェクトであり，図B.1では斜線部の面積である．したがって，KNNアルゴリズムによる検索に必要な計算コストは，レンジクエリで$r = d(q, u_{Rank_d(K)})$以下のオブジェクトを検索するのに必要な計算コストと一致する．

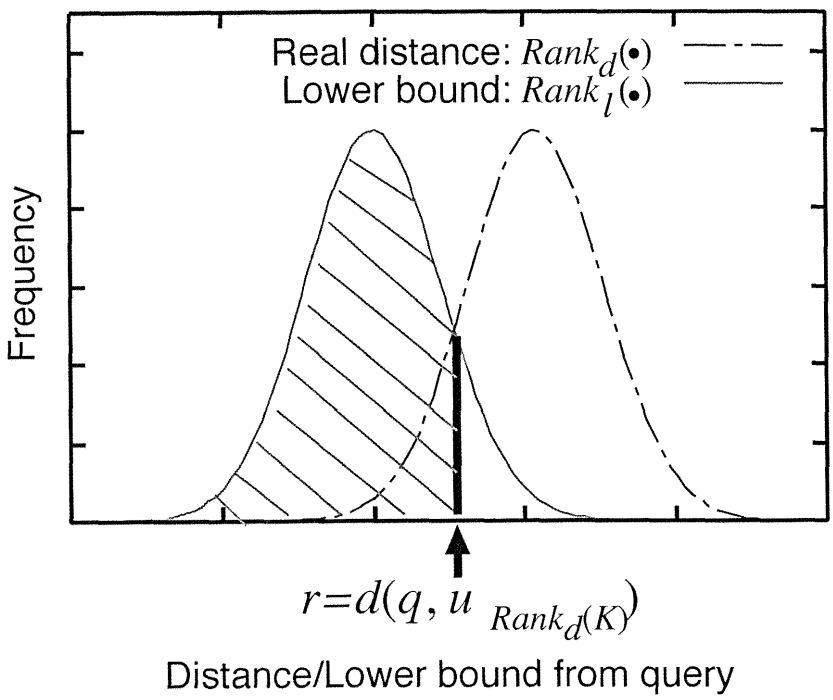


図 B.1: レンジクエリと KNN クエリとの関係性

付録 C　固有次元：近傍探索問題の困難さの指標

近傍検索を高速化できるかどうかは，データベース内のデータオブジェクトの分布に結局は依存する．データオブジェクトの分布を定量評価する尺度として，“距離空間の固有次元（intrinsic dimensionality）”が提案されている [7]．“距離空間の固有次元”とは，既存の近傍検索手法の難しさを表す指標であり，図 C.1 のようなオブジェクト対の距離の分布に対して，次式で定義される．

$$\rho = \frac{\mu^2}{2\sigma^2} \tag{C.1}$$

ここで，μ，σ は，ペアワイズオブジェクトの距離の分布に対する平均と分散である．固有次元 ρ の数値は，近傍検索の難しさに比例するとされている [7]．

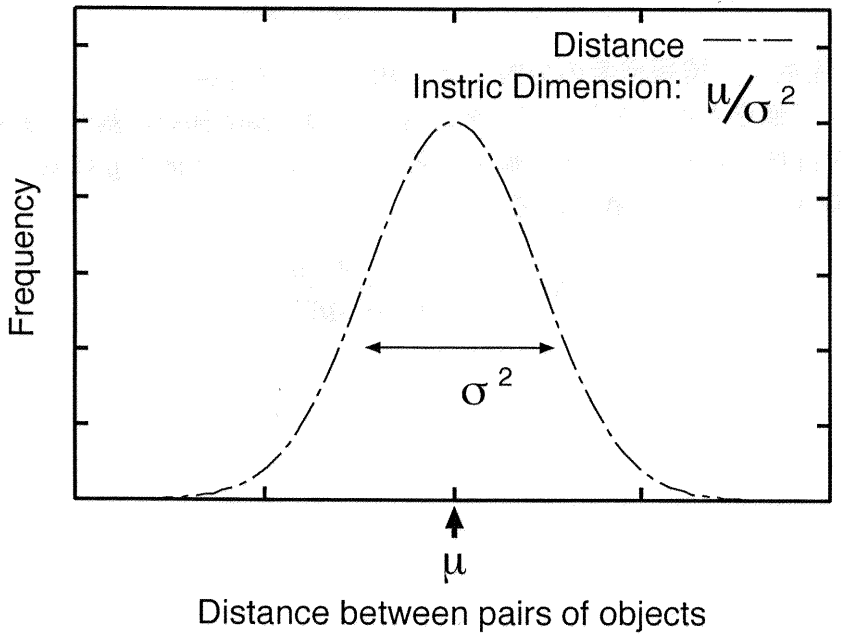

Distance
Instric Dimension: μ/σ 2
Frequency
σ 2
μ
Distance between pairs of objects

図 C.1: 固有次元

謝辞

本研究に加え生活面（整理整頓含む）でもご指導をくださいました新保仁助教に感謝いたします．新保さんには，思いついたばかりの訳の分からない話にもつきあっていただきました．ありがとうございました．

お忙しい中，論文審査委員をお引き受けくださいました論理生命学講座の池田和司教授に感謝いたします．池田先生からは論文や発表のわかりにくい部分をご指摘いただきました．ありがとうございました．

主指導教官として研究室で温かく見守ってくださいました松本裕治教授に感謝いたします．松本先生からは，研究設備や金銭面でのご支援をいただきました．時には，教授という立場でありながら，先生の方から雑談をしていただき，私も含め学生との距離を縮めてくださいました．ありがとうございました．

副指導教員として研究室でご指導くださいました乾健太郎准教授に感謝いたします．研究室で私を研究員として雇用した予算は，乾さんが獲得した研究資金によるものでした．おかげで3年間大病をせずに研究生活を送ることができました．ありがとうございました．

新保さんと同様に私の話を聞いてくださった原一夫研究員に感謝いたします．原さんには何のメリットも無いにもかかわらず，いろんな面でサポートしていただきました．ありがとうございました．

本研究を始めるきっかけをつくってくださった静岡県立大学の斉藤和巳教授，そして，NTTコミュニケーション科学基礎研究所の上田修功博士に感謝いたします．私がジャーナルを作成する際に　お二人は遠方にいながらもご指導をじっくりとしてくださいました．ありがとうございました．

これまで一緒に研究してきた松本研究室の皆さん，そして，松本研究室のOBの皆さんに感謝いたします．皆さんには，公私にわたり，とてもお世話になりました．ありがとうございました．

陰ながら私を支えてくださった私の父と母に感謝いたします．27歳という働きざかりの年齢でありでながら，学生であることを認めていただきました．これからは，何らかの形で恩返しをしたいと思います．ありがとうございました．

最後になりましたが，奈良先端科学技術大学院大学，そして，職員の皆さんに

感謝を述べたいと思います．ありがとうございました．

参考文献

[1] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun. ACM*, 51(1):117–122, 2008.

[2] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The r*-tree: An efficient and robust access method for points and rectangles. In H. Garcia-Molina and H. V. Jagadish eds., *SIGMOD Conference*, pp. 322–331. ACM Press, 1990.

[3] S. Berchtold, D. A. Keim, and H.-P. Kriegel. The x-tree : An index structure for high-dimensional data. In T. M. Vijayaraman, A. P. Buchmann, C. Mohan, and N. L. Sarda eds., *VLDB*, pp. 28–39. Morgan Kaufmann, 1996.

[4] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[5] S. Brin. Near neighbor search in large metric spaces. In U. Dayal, P. M. D. Gray, and S. Nishio eds., *VLDB*, pp. 574–584. Morgan Kaufmann, 1995.

[6] B. Bustos, G. Navarro, and E. Chávez. Pivot selection techniques for proximity searching in metric spaces. *Pattern Recognition Letters*, 24(14):2357–2366, 2003.

[7] E. Chávez, G. Navarro, R. A. Baeza-Yates, and J. L. Marroquín. Searching in metric spaces. *ACM Comput. Surv.*, 33(3):273–321, 2001.

[8] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In M. Jarke, M. J. Carey, K. R. Dittrich, F. H. Lochovsky, P. Loucopoulos, and M. A. Jeusfeld eds., *VLDB*, pp. 426–435. Morgan Kaufmann, 1997.

[9] M. Collins and N. Duffy. Convolution kernels for natural language. In T. G. Dietterich, S. Becker, and Z. Ghahramani eds., *NIPS*, pp. 625–632. MIT Press, 2001.

[10] R. A. Finkel and J. L. Bentley. Quad trees: A data structure for retrieval on composite keys. *Acta Inf.*, 4:1–9, 1974.

[11] A. Guttman. R-trees: A dynamic index structure for spatial searching. In B. Yormark ed., *SIGMOD Conference*, pp. 47–57. ACM Press, 1984.

[12] T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning*. Springer, August 2001.

[13] D. Haussler. Convolution Kernels on Discrete Structures. Technical Report UCSC-CRL-99-10, UC Santa Cruz, 1999.

[14] T. Hofmann. Probabilistic latent semantic indexing. In *SIGIR*, pp. 50–57. ACM, 1999.

[15] P. Jain, B. Kulis, I. S. Dhillon, and K. Grauman. Online metric learning and fast similarity search. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou eds., *Advances in Neural Information Processing Systems 21*, pp. 761–768. Nips Foundation (http://books.nips.cc), 2009.

[16] H. Kashima and T. Koyanagi. Kernels for semi-structured data. In *ICML '02: Proceedings of the Nineteenth International Conference on Machine Learning*, pp. 291–298, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

[17] N. Katayama and S. Satoh. The sr-tree: An index structure for high-dimensional nearest neighbor queries. In J. Peckham ed., *SIGMOD Conference*, pp. 369–380. ACM Press, 1997.

[18] 木村, 斉藤, 上田. 効率的な類似検索のためのピボット学習法. 情報処理学会論文誌, 50(8):1883–1891, 2009.

[19] B. Kulis and T. Darrell. Learning to hash with binary reconstructive embeddings. In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta eds., *Advances in Neural Information Processing Systems 22*, pp. 1042–1050. Nips Foundation (http://books.nips.cc), 2009.

[20] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing for scalable image search. In *The Twelfth IEEE International Conference on Conputer Vison(ICCV 2009)*. IEEE Computer Society, 2009.

[21] K.-I. Lin, H. V. Jagadish, and C. Faloutsos. The tv-tree: An index structure for high-dimensional data. *VLDB J.*, 3(4):517–542, 1994.

[22] D. B. Lomet and B. Salzberg. The hb-tree: A multiattribute indexing method with good guaranteed performance. *ACM Trans. Database Syst.*, 15(4):625–658, 1990.

[23] D. G. Luenberger. *Introduction to Linear and Nonlinear Programming.* Addison-Wesley, 1973.

[24] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval.* Cambridge University Press, 1 edition, July 2008.

[25] C. D. Manning and H. Schtze. *Foundations of Statistical Natural Language Processing.* The MIT Press, June 1999.

[26] C. D. Meyer ed. *Matrix analysis and applied linear algebra.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[27] L. Micó, J. Oncina, and E. Vidal. A new version of the nearest-neighbour approximating and eliminating search algorithm (aesa) with linear preprocessing time and memory requirements. *Pattern Recognition Letters*, 15(1):9–17, 1994.

[28] A. Moschitti. Making tree kernels practical for natural language learning. In *EACL.* The Association for Computer Linguistics, 2006.

[29] J. Nievergelt, H. Hinterberger, and K. C. Sevcik. The grid file: An adaptable, symmetric multikey file structure. *ACM Trans. Database Syst.*, 9(1):38–71, 1984.

[30] J. Nocedal and S. J. Wright. *Numerical Optimization.* Springer, August 1999.

[31] J. T. Robinson. The k-d-b-tree: A search structure for large multidimensional dynamic indexes. In Y. E. Lien ed., *SIGMOD Conference*, pp. 10–18. ACM Press, 1981.

[32] E. V. Ruiz. An algorithm for finding nearest neighbours in (approximately) constant average time. *Pattern Recogn. Lett.*, 4(3):145–157, 1986.

[33] T. K. Sellis, N. Roussopoulos, and C. Faloutsos. The r+-tree: A dynamic index for multi-dimensional objects. In P. M. Stocker, W. Kent, and P. Hammersley eds., *VLDB*, pp. 507–518. Morgan Kaufmann, 1987.

[34] C. Shen, A. Welsh, and L. Wang. Psdboost: Matrix-generation linear programming for positive semidefinite matrices learning. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou eds., *Advances in Neural Information Processing Systems 21*, pp. 1473–1480. Nips Foundation (http://books.nips.cc), 2009.

[35] J. K. Uhlmann. Satisfying general proximity/similarity queries with metric trees. *Inf. Process. Lett.*, 40(4):175–179, 1991.

[36] E. Vidal. New formulation and improvements of the nearest-neighbour approximating and eliminating search algorithm (aesa). *Pattern Recognition Letters*, 15(1):1–7, 1994.

[37] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In A. Gupta, O. Shmueli, and J. Widom eds., *VLDB*, pp. 194–205. Morgan Kaufmann, 1998.

[38] Y. Weiss, A. B. Torralba, and R. Fergus. Spectral hashing. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou eds., *NIPS*, pp. 1753–1760. MIT Press, 2008.

[39] P. N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *SODA*, pp. 311–321, 1993.

[40] 武本, 加藤, 和田. 効率的な距離計算戦略による高次元最近傍探索の高速化 (テーマセッション (2),cv のためのパターン認識・学習理論の新展開). 電子情報通信学会技術研究報告. PRMU, パターン認識・メディア理解, 105(673):49–56, 20060309.

研究業績

学術論文

1. 木村, 斉藤, 上田. 効率的な類似検索のためのピボット学習法. 情報処理学会論文誌, 50(8):1883-1891, 2009.

国際会議

1. M. Kimura, K. Saito, and N. Ueda. Pivot learning for efficient similarity search. In B. Apolloni, R. J. Howlett, and L. C. Jain eds., *KES (3)*, Vol. 4694 of *Lecture Notes in Computer Science*, pp. 227-234. Springer, 2007.