修士論文

クラスタリングを利用した Top-k 組み合わせ検索の 効率化に関する研究

A Study to Improve Top-k Combinatorial Search Using Clustering

48106418

鈴木 貴敦

東京大学大学院 情報理工学系研究科 電子情報学専攻

指導教員 安達 淳

2012 年 2 月 8 日提出

概要

これまでに，データベースに対する検索システムは，精度，再現度，速度ともにめざましく進歩した．検索は生活のあらゆるレベルに浸透し，コモディティ化したため，検索に対する要求も性能面はもちろんのこと，結果の質においても高い水準のものが求められている．本論文では，検索結果の質の高度化を行うべく，データの組み合わせを対象とした検索手法について検討する．この検索には，組み合わせ爆発という言葉があるとおり，検索対象が指数関数的に増えてしまう問題がある．そこで，検索結果の上位だけを高速に求める Top-k 検索技術に注目し，高速化を行う．本論文では，性質の異なる 2 種類の Top-k 組み合わせ検索に対して，クラスタリングを利用した高速化手法を提案し，有効性を示した．

目次

図目次

表目次

第 1 章
序論

1.1 研究の背景と目的

人類が生み出す情報が爆発的に増加しているのは周知の事実である．統計によれば，地球全体で人間が作り出すデータの総量は 2020 年に 35ZB(= 35×10^9TB= $35 \times 10^{21} B$) に達すると予想されている (図 1.1)[10]．特に近年では，“Big Data” をキーワードとし，情報分野だけでなく，医療，土木，流通等様々な分野で大量に蓄積したデータから情報を抜き出し，活用しようとする動きが活発になってきた．頻出パターン抽出や異常値検出などの知識発見の技術等，大量のデータから必要な情報を抜き出すための技術は様々あるが，最も身近で最もよく使われている技術は，必要なデータを探し出す情報検索技術である．

検索技術は，Web 検索をはじめとして，全文検索，類似文字列検索等，我々の生活に深く入り込んでいる．そのため，検索に対する要求水準と，それに応えるための技術の水準は日々高くなっている．処理効率面では，大規模な商用検索システムともなれば，1 秒間に数万以上のリクエストをリアルタイムに処理することができる [5]．また，結果の精度や質の面においても，スパム除去はもちろんこと，ユーザの嗜好に合わせた検索結果を出力することで，検索技術はユーザのためだけではなく，商品の宣伝媒体としても重要な意味を持つようになった．

進歩が著しい検索技術であるが，未だ不得手とする部分が存在する．その 1 つが，データの組み合わせを対象とした検索である．以下にクエリ例を挙げる．

クエリ例 (1) 自分の行動ログから，健康的な生活を送るために何が欠けているのか知りたい．

クエリ例 (2) 自分の専門分野の学会に参加しつつ，海でのんびり休養するにはどこに行けばよいか．

クエリ例 (1) に答えるためには，まずは行動を数値化し，理想的な行動と自分の行動の差を取る．そして，その差を埋めることができるような行動の組み合わせを結果として出力すればよい．クエリ例 (2) に答えるためには，学会のデータベースと観光地のデータベースの両方を調べ，それぞれ得られた結果を統合すればよい．

このようなクエリに対するシステム側からのアプローチとして，既存の検索システムを統合して回答を作成するメタサーチが挙げられる [11][3]．メタサーチの利点は，既存の検索システムを並行して利用できる点にある．しかし，これらはあくまで既存の検索システムへのゲートウェイとしての性格が強く，得られた結

果を提示するにとどまる．そのため，既存の検索システムから得られた結果を組み合わせて，クエリに合致する結果を作りだすことはできない．

ユーザ側からのアプローチとして，ウェブサービスのマッシュアップが挙げられる [27]．熟練ユーザであれば，各々のニーズに合わせたシステムを作り出すことが可能であるかもしれないが，やはり一般ユーザにとっては敷居が高いものである．

以上を踏まえて，検索システムにおいて組み合わせ処理を行い，クエリへの回答を作成する検索を考える．組み合わせを扱う上で最も大きな問題となるのは組み合わせ爆発による解候補の増加である．そこで，本論文では Top-k 検索と呼ばれる技術に注目した [7]．Top-k 検索とは，検索結果のなかで最も重要なものの上位 k 件のみを高速に探すための手法である．たいていの場合，我々人間が必要とする検索結果は，上位の一部で充分であると考えられるため，検索候補が多い組み合わせを対象とした検索を高速化させるための手法としては，この方法が適していると言える．

本論文では，組み合わせを対象とする検索の実現のため，以下の異なる 2 種類の Top-k 組み合わせ検索について取り組んだ．それぞれ，テーマ 1 がクエリ例 (1)，テーマ 2 がクエリ例 (2) に対応する．

テーマ 1 Euclid 距離を評価尺度とした場合のベクトル空間における Top-k 組み合わせ検索

テーマ 2 成立しない組み合わせが存在する場合の Top-k Join

本論文の貢献は，これら 2 つの Top-k 組み合わせ検索において，クラスタリングを利用した高速化手法を提案し，その有効性を示したところにある．

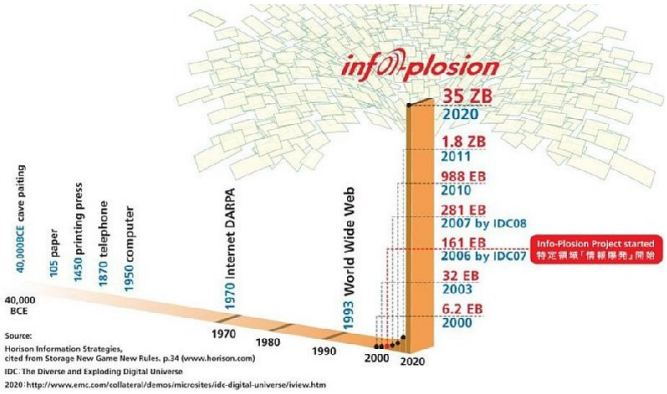

図 1.1: 人類が生み出すデータ量の急激な増加.2020 年には 35ZB に達すると予測されている.

1.2 本論文の構成

本論文の構成は以下のとおりである．まず，第 2 章で，これまでに研究されてきた Top-k 検索の手法について紹介する．第 3 章で，評価関数として Euclid 距離を利用した場合のベクトル空間上の Top-k 組み合わせ検索に対する，クラスタリングを利用した高速化手法の詳細について述べる．第 4 章で，第 3 章で提案した手法をベースに，Top-k Join の高速化手法の提案，およびクエリ処理の最適化に必要なコスト予測手法の詳細について述べる．最後に第 5 章で，本研究のまとめと今後の展望について述べる．

第 2 章
関連研究

2.1 一般的な Top-k 検索の手法

Top-k 検索では，何らかの基準によってデータベース内のオブジェクトに対して順位付けを行う．評価関数とは，データベース内のオブジェクトを引数として，それに対応する得点を返す働きを持つ．評価関数を利用して得た得点を利用して順位付けを行い，上位 k 件のオブジェクトのみを求めるのが Top-k 検索である．

この得点付けを行う評価関数の性質によって，Top-k 検索の戦略が大きく異なる．本節では，評価関数の性質によって Top-k 検索技術を分類した．

2.1.1 評価関数が単調変化する場合

多変数の評価関数 f が単調変化であるとは，単調増加 (減少) である場合のことを指す．以下に単調増加 (減少) の定義を示す．単調変化である関数の例としては，平均，最大，最小が挙げられる．

定義 1 単調増加 *(減少)*
関数 $f : \mathbf{R}^m \rightarrow \mathbf{R}$ が単調増加 *(減少)* であれば，$f(x_1, x_2, \ldots, x_m) \geq f(x'_1, x'_2, \ldots, x'_m)$ が $\forall i x_i \geq x'_i$ ($f(x_1, x_2, \ldots, x_m) \leq f(x'_1, x'_2, \ldots, x'_m)$ が $\forall i x_i \leq x'_i$) について成立する．

単調変化である場合の Top-k 検索の方針は以下の通りである．

1. 評価関数に応じて，ベクトルの各要素ごとに整列させた列を作る
2. 先頭から順次アクセスを行い，評価関数を適用する
3. 上位 k 番目の評価値を利用し，枝刈りや探索終了判定を行う

枝刈りのための閾値を考えやすいため，単調変化の条件を前提としたり，また，評価関数が単調変化でなくとも，何らかの方法で単調変化へ変換させてから枝刈りを行う場合は多く，類似検索や文書検索のためのクエリ拡張など様々なところで応用されている [23][13][19][20].

例として，図 2.1 に示す家の価格データベースと，必要な教育費のデータベースから，引越し先を決める場合について考える．A さん一家は，A さん，妻，子ども 2 人の 4 人家族である．子どもたちの成長に伴い，現在住んでいる家が手狭になったので，広い家へ引越を考えている．子どもは 2 人とも来年から小学生であるため，これから先 12 年間の教育費と家の価格が最小になるような場所に新居を

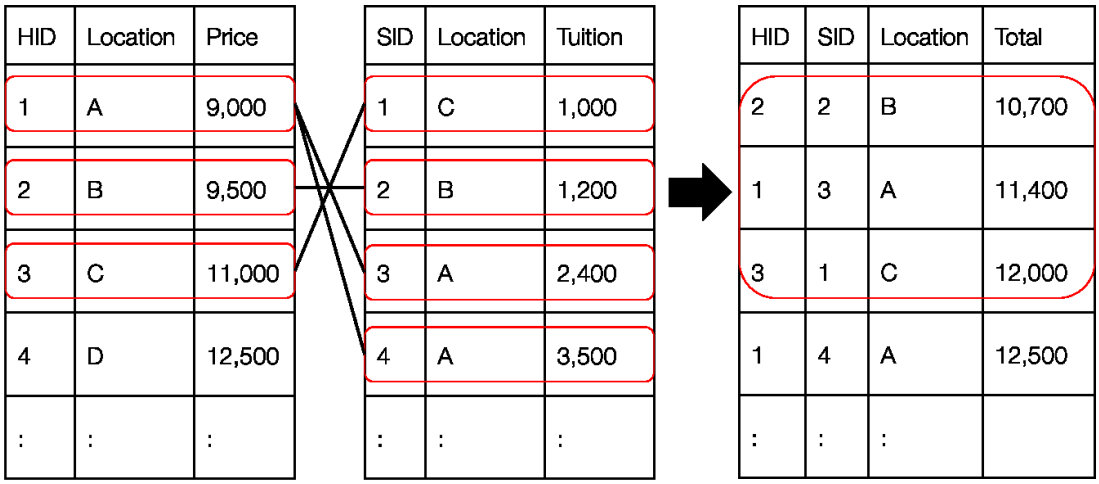

図 2.1: 家の価格と教育費の例

構えたい．そこで，検索を行い，家の価格と教育費の和が小さい場所の上位 3 件を調べて，引越し先を検討することにした．

家の価格と場所のデータベースと，教育費と場所のデータベースが存在し，それぞれ値段の低い順に整列されているとする (*1*)．先頭から順にアクセスを行い，評価関数 f を適用する (*2*)．評価関数 f は，f = (家の価格) + (教育費) の 2 変数の単調増加関数となるため，一方のデータベースで見つかった値に関して，他方のデータベースから，足りない値を補完する．評価関数が単調増加のため，上位 3 番目の解が現在注目している位置の和，すなわち，見つけていない要素でつくることのできる最小の値よりも小さければ探索終了である (*3*)．図 2.1 にデータベースと結果を示す．

単調増加関数を利用する代表的な汎用アルゴリズム　ここでは，Fagin らによって提案された，最もよく使われている 3 つの Top-k 検索アルゴリズムについて述べる [7]．3 つはデータへのアクセス方法で分類されている．全てのアルゴリズムで，各属性ごとに整列済みのリストが別々に管理されており，評価関数として，オブジェクトの属性値を引数とする，単調変化関数を使用する．図 2.2 に，3 つのアルゴリズムが仮定しているデータベースのモデルを示す．オブジェクトの属性値とは，図 2.2 中の $attr_i$ 列のことであり，今回は各 $attr_i$ の持つ値が実数である場合について扱う．

1 つ目の Threshold Algorithm(TA) では，整列済みリストの先頭から順にアクセスするシーケンシャルアクセスと，評価に足りない値を補完するためのランダム

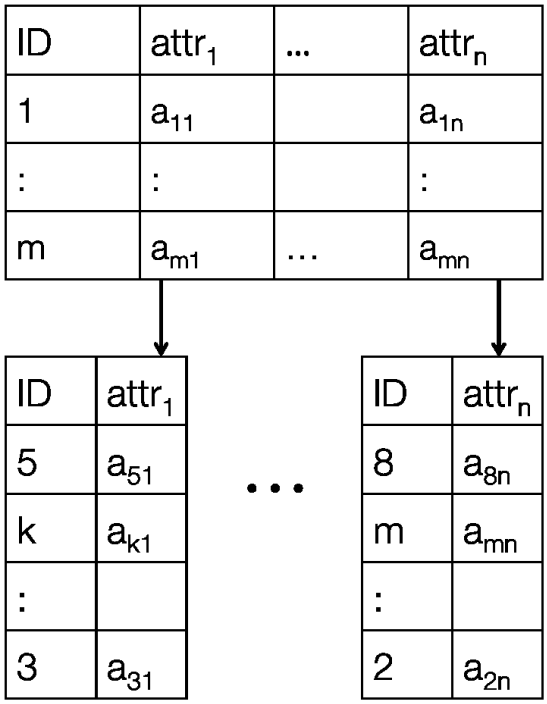

図 2.2: TA，NRA，CA で前提とするデータベースモデル．各属性は独立に評価関数に従って整列されている．この場合は評価関数が属性のもつ値の和であるため，大きいほど得点が高い．したがって，値の大きいものから順に並べられている．

アクセスの両方が許されている場合，2 つ目の No Random Algorithm ではシーケンシャルアクセスのみが許されている場合，3 つ目の Combined Algorithm ではシーケンシャルアクセスとランダムアクセスのコスト比を考慮した場合について研究されている．

Threshold Algorithm(TA)　まず，基本となる Threshold Algorithm(TA) とは，整列済みリストの先頭から順にアクセスするときのコストと，リストへのランダムアクセス時のコストが同じであるという前提のもと考えられたアルゴリズムである．探索の効率化のため，閾値 τ を設定し，上位 k 番目の得点が τ 以上になった場合は探索を終了する．評価関数が単調変化であれば，解が正しいことが保証されている．

アルゴリズムを以下に示す．

1. n 個の整列済みリストに先頭から順次アクセスを行う

   (a) 見つかったオブジェクトの要素に関して，各リストにランダムアクセス

を行い，オブジェクト内で見つかっていない要素を全て埋める

(b) 得点計算を行う

(c) オブジェクトと得点の組み合わせを上位 k 個だけ保持する

2. 各リストの最後にアクセスした値 $\underline{x_i}$ を元に，閾値 $\tau = f(\underline{x_i}, \ldots)$ を計算する

3. τ が上位 k 番目のスコア以下ならば探索終了

No Random Algorithm(NRA)　前述の TA では，ランダムアクセスとシーケンシャルアクセスのコストを同程度として扱っていた. ここで，例えば TA を文書検索へ適用することを考える．単語と文書の紐付けを行う転置リストへのランダムアクセスは不可能ではないものの非常にコストがかかってしまうため，現実的でない．また，一般にランダムアクセスはシーケンシャルアクセスよりもコストがかかる場合が多い．

そこで，ランダムアクセスをせずに上位 k 件を求める方法として No Random Algorithm(NRA) が提案された．以下にアルゴリズムを示す．

1. n 個の整列済みリストに先頭から順次アクセスを行う．現在注目している深さを d とし，各 d において以下のことを行う．

   - 各リストにおける，深さ d までの最小値 $\underline{x}_1^{(d)}, \underline{x}_2^{(d)}, \cdots, \underline{x}_m^{(d)}$ を更新する．
   - 全てのオブジェクト R について，見つかっている部分だけを用いて，深さ d における最小値 $W^{(d)}(R)$，最大値 $B^{(d)}(R)$ の計算を行う (見つかっていない項目に関しては，$W^{(d)}$ は 0,$B^{(d)}$ は $\underline{x}_i^{(d)}$ で補完する)．見つかっていないオブジェクト R' に関しては，$W^{(d)}(R') = f(0, \ldots, 0)$，$B^{(d)}(R') = f(\underline{x}_1^{(d)}, \underline{x}_2^{(d)}, \cdots, \underline{x}_m^{(d)})$ として扱う．
   - $T_k^{(d)}$ を深さ d の,$W^{(d)}$ に関する Top-k リストとする．2 つのオブジェクトが等しい $W^{(d)}$ の値である場合は，$B^{(d)}$ の大小で順位を決定する．ここで，$M_k^{(d)}$ を k 番目に大きい $W^{(d)}$ の値とする．

2. 以下の 2 つの条件を満たしたときに探索を終了する．$T_k^{(d)}$ が目的の Top-k のリストとなる．

   - $T_k^{(d)}$ が k 個のオブジェクトを保持している

- $T_k^{(d)}$ に含まれていない全てのオブジェクト R に関して，$B^{(d)} \leq M_k^{(d)}$ が成立する

Combined Algorithm(CA)　Combined Algorithm(CA) とは，TA と NRA を組み合わせたアルゴリズムであり，ランダムアクセスの回数を最小化することを目的としている．

シーケンシャルアクセスのコストを c_S，ランダムアクセスのコストを c_R とし，$h = \frac{c_R}{l} c_S$ とする．CA は，基本的には NRA と同じく各リストに対して並列にシーケンシャルアクセスを行うが，アクセス h 回ごとにリストへのランダムアクセスを行い，スコアの上限値 $B^{(d)}$ と下限値 $W^{(d)}$ を更新する．以下にアルゴリズムを示す．

1. m 個の整列済みリストに先頭から順次アクセスを行う．現在注目している深さを d とし，各 d において以下のことを行う．

   - 各リストにおける，深さ d までの最小値 $\underline{x}_1^{(d)}, \underline{x}_2^{(d)}, \cdots, \underline{x}_m^{(d)}$ を更新する．
   - 全てのオブジェクト R について，見つかっている部分だけを用いて，深さ d における最小値 $W^{(d)}(R)$，最大値 $B^{(d)}(R)$ の計算を行う (見つかっていない項目に関しては，$W^{(d)}$ は 0,$B^{(d)}$ は $\underline{x}_i^{(d)}$ で補完する)．見つかっていないオブジェクト R' に関しては，$W^{(d)}(R') = f(0, \ldots, 0)$，$B^{(d)}(R') = f(\underline{x}_1^{(d)}, \underline{x}_2^{(d)}, \cdots, \underline{x}_m^{(d)})$ として扱う．
   - $T_k^{(d)}$ を深さ d の,$W^{(d)}$ に関する Top-k リストとする．2 つのオブジェクトが等しい $W^{(d)}$ の値である場合は，$B^{(d)}$ の大小で順位を決定する．ここで，$M_k^{(d)}$ を k 番目に大きい $W^{(d)}$ の値とする．

2. h 回シーケンシャルアクセスを行うごと (すなわち，深さが h 深くなるごとに) に，ランダムアクセスを行う．具体的には，$B^{(d)}(R) < M_k^{(d)}$ を満たすオブジェクトで，全ての項目が未探索のものに関して，ランダムアクセスを行い，項目を埋める．

3. 以下の 2 つの条件を満たしたときに探索を終了する．$T_k^{(d)}$ が目的の Top-k のリストとなる．

   - $T_k^{(d)}$ が k 個のオブジェクトを保持している
   - $T_k^{(d)}$ に含まれていない全てのオブジェクト R に関して，$B^{(d)} \leq M_k^{(d)}$ が成立する

2.1.2 評価関数が単調変化しない場合

評価関数は常に単調変化するものを選べるわけではない．例えば，自乗誤差を最小にするようなものを検索したい場合を考える．評価関数 f は $f = (x - \alpha)^2$ の形をとるため，単調変化する関数ではなくなってしまう．そのため，先程述べた方針を直接適用することはできない．

このような単調変化しない場合の方策として，評価関数の代わりに限界値を与えてくれる単調変化関数を利用する方法，可能最大値を利用して枝刈りを行う方法，効率的な探索候補を生成する方法が挙げられる [25][14][9][29].

木構造で索引付けを行う例を 1 つ紹介する．この方法は 2007 年に Xin らによって提案された方法で，B 木や R 木などを利用して，各要素ごとに木を作成し，それらを利用して順次探索候補を生成しつつ，上位 k 件を求める [24]．評価関数 f が，変数のドメイン Ω 上における上限値もしくは下限値が理論的に導けるものであれば，この手法が適用可能となる．方針は以下の通りである．

1. オブジェクトの各要素に関して索引付けを行う
2. 各索引の統合を行い，評価関数に応じた探索候補を順次生成していく
3. 生成された探索候補に評価関数を適用し，上位 k 件を求める

編集距離を利用した木マッチングへの応用　ここでは，単調変化しない評価関数を用いた Top-k 検索を XML の部分木マッチングへ応用した例の 1 つである，Augsten らによって提案された Top-k Approximate Subtree Matching(TASM) を紹介する [1].

XML の部分木マッチングの従来手法では，時間計算量が $O(n^3)$，空間計算量が $O(n^2)$ であるため，DBLP のような巨大な木を対象にした検索は困難であった [18].

Nicolas らが提案した TASM-postorder では，空間計算量を大幅に削減しており，探索候補を限定するためのバッファの大きさは入力文書の大きさによらず，求めたい解の数 k とクエリ木の大きさのみに依存する．そのため，既存手法で扱うことができなかった大きなサイズの文書にも適用できる．

問題定義　条件を表 2.1 にまとめる．$R = (T_{i_1}, T_{i_2}, \ldots, T_{i_k})$ が，Q に関して T の top-k 部分木であるとは，以下の 2 つの条件をみたすことである．

1. R はクエリに最も近い k 個の木を保持している
   $\forall T_j \notin R : d(Q, T_{i_k}) \leq d(Q, T_j)$

表 2.1: 表記のまとめ

Q	クエリ木
T	整列済みラベル付き木
n	T のノード数
T_i	t_i を根とし，t_i の子ノードを全て含む T の部分木
$d(\cdot,\cdot)$	整列済みラベル付き木間の距離
k	$k \leq n$ を満たす整数
R	部分木列

2. R 中の部分木は，クエリからの距離によって整列されている
   $\forall j(1 \leq j < k) : d(Q, T_{i_j}) \leq d(Q, T_{i_j+1})$

Top-k Approximate Subtree Matching は，文書木 T が与えられたときに，クエリ木 Q に関して，距離が小さいもの上位 k 件の部分木を求めるという問題を扱う．

用語定義　木 T とは，循環がなく，全てのノードが連結されている空でない有向グラフであり，木を構成するノード t_i は (identifier,label) のペアとする．T のノードを $V(T)$ とし，エッジを $E(T)$ とする．集合 $V(T)$ に空ノードを加えた集合を $V_\epsilon(T)$ とする．identifier は木で一意に決定され，ノード t_i の identifier は $id(t_i)$ と表す，ラベルは有限アルファベットの集合であり，$\lambda(t_i)$ と表す．

根をノード $t_i \in V(T)$ とする部分木 $T_i \in T$ とは，t_i と，t_i の子ノード全てで構成される．

postorder queue とは，ラベルと大きさのペアが postorder で格納されているキューである．大きさは，各ノードを根とした場合の部分木のノード数で表される．図 2.3 の木の場合，postorder queue は $((sauce, 1), (salt, 2), (pepper, 1), (olive, 1), (mint, 3), (sugar, 6))$ となる．

TASM-postorder　クエリ Q と k が与えられると，解候補となる部分木の大きさの閾値 τ を決めることができる．τ を与える定理は以下の 3 つの補題より導くことができる (証明は省略する)．

補題 1 木の大きさと編集距離の関係
Q, T をそれぞれ整列済みラベル付き木とすると，$|T| \leq \delta(Q, T) + |Q|$ が成立する．

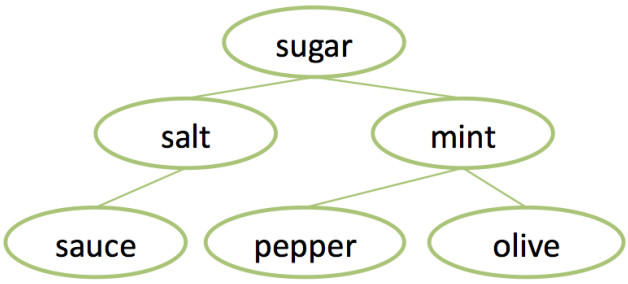

図 2.3: 木の例

補題 2 中途順位が与える部分木の大きさの上限
$R' = (T_{i'_1}, T_{i'_2}, \ldots, T_{i'_k})$ をクエリ Q に関するある時点での順位とする．R を最終的な順位とすると，全ての部分木 $T_i \in T$ に対して，$\forall T_{i_j}(|T_{i_j}| \in R \rightarrow |T_{i_j}| \leq \delta(Q, T_{i'_j}) + |Q|)$ が成立する．

補題 3 T の先頭から k 番目までのノードで構成された部分木 T_i と編集距離の上限
Q, T を整列済みラベル付き木 $(k \leq |T|)$，c_Q, c_T をそれぞれ Q, T のノードにおける最大のコスト，t_i を T の i 番目のノードとする．このとき，$1 \leq i \leq k$ において，$|T_i| \leq k \wedge \delta(Q, T_i) \leq |Q|c_Q + kc_T$ が成り立つ．

以上より，ノード数の閾値に関する定理を導くことができる．この定理で求められる閾値により，探索候補の限定を行うことができる．

定理 1 上位 k 番目以内に順位付けされる部分木の大きさの上限
Q, T をそれぞれ整列済みラベル付き木とし，c_Q, c_T をそれぞれ Q, T のノードにおける最大のコストとする．$R = (T_{i_1}, T_{i_2}, \ldots, T_{i_k})$ をクエリ Q に関する最終的な順位だとすると，R 内の全ての部分木は $\tau = |Q|(c_Q + 1) + kc_T$ で大きさの上限が定められる．

検索候補の限定 閾値 τ を利用して探索候補となる部分木を postorder queue から抽出する．木 T の探索候補 $cand(T, \tau)$ は以下の式 (2.1) により定義される．なお，$|T_i|$ はノード t_i を根とする部分木 T_i を構成するノードの数, $anc(t_i)$ はノード t_i の親ノードを表す．

$$cand(T, \tau) = \{T_i \mid t_i \in V(T), |T_i| \leq \tau, \forall t_a \in anc(t_i) : |T_a| > \tau\} \tag{2.1}$$

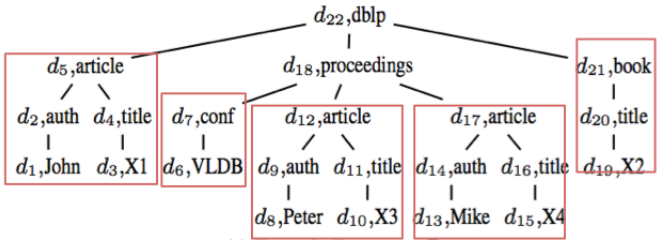

図 2.4: 探索候補の例 [1]

$\tau = 6$ のときの探索候補の例を図 2.4 に示す．図中の赤枠で囲まれた部分木が探索候補となる．

Ring Buffer Pruning　前パラグラフで定義した探索候補を，実際に postorder queue から取り出すための手法が Ring Buffer Pruning である．$\tau + 1$ 個のノードが入るリングバッファを利用して，探索候補となる部分木を取り出す．リングバッファの大きさはクエリ木 Q の大きさ k とだけに依存し，文書の大きさによらないため，既存手法では扱えないような大きな文書にも適用できる．

手順を以下に示す．

1. リングバッファが埋まるか，キューが空になるまで取り出す
2. リングバッファ内の最も番号の小さいノードが
   (a) 葉ノードのとき：そのノードを含む部分木を探索候補とする
   (b) 葉ノードでないとき：バッファから取り除く
3. キューとバッファの状態で場合分け
   (a) キューが空でない：1 へ
   (b) キューが空でバッファは空でない：2 へ
   (c) 全て空：終了

TASM-postorder　TASM-postorder アルゴリズムの概略を以下に示す．R は上位 k 件の部分木列，τ', τ は部分木 T_i のノード数の閾値である．

1. postorder queue からリングバッファへ読み込む

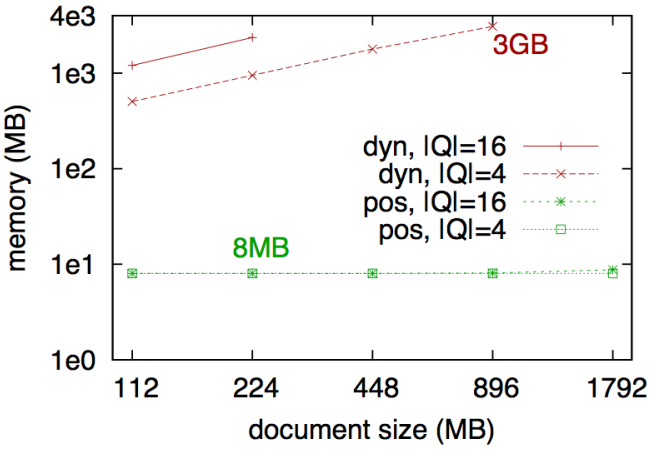

図 2.5: メモリ使用量の比較 [1]

2. $|R| = k$ のとき：閾値 τ' を更新.
   $\tau' = min(\tau, max(R) + |Q|)$

3. $|R| < k$ または $|T_i| < \tau'$ のとき，$\delta(Q, T_i)$ を計算し，解の整理を行う

4. postorder queue とリングバッファが空でない場合：1 へ
   両方とも空の場合：終了

結果　図 2.5 に文書の大きさを変化させた場合のメモリ使用量のグラフを示す．使用したデータセットは，XML のベンチマークに利用される XMark の文書である [26].

TASM ではメモリ使用量は入力の大きさによらないため，動的計画法では処理し切れないような大きなサイズの文書も扱えることがわかる．

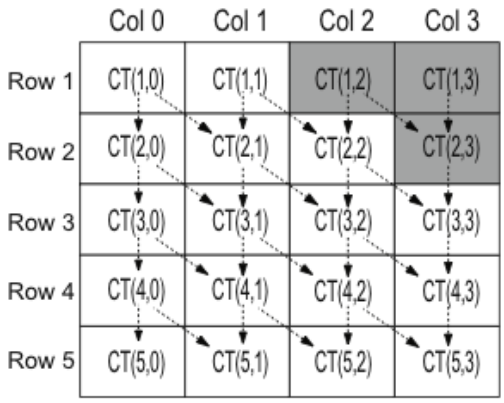

図 2.6: Solution Table

2.2 組み合わせを対象とした Top-k 検索手法

2.2.1 Skyline Query との併用

Skyline 検索 Skyline 検索とは，Börzsönyi らによって提案された，多次元の数値データを対象とした，支配されていない点の集合を求める検索である [2]．点 p が点 q を支配していることの定義を以下に示す．

定義 2 支配
点 $p = (p_1, p_2, \ldots, p_m)$ が，点 $q = (q_1, q_2, \ldots, q_m)$ を支配しているとき，以下の 2 つの条件が成り立つ．

1. 全ての $i(1 \leq i \leq m)$ に対して，$p_i \leq q_i$
2. ある $j(1 \leq j \leq m)$ に対して，$p_j < q_j$

Combinatorial Skyline Queries 組み合わせ Skyline 検索とは，Su らが提案した手法で，Skyline の条件を満たすオブジェクトの組み合わせの集合を求めるものである [17]．応用先としては金融資産のポートフォリオが想定されている．

オブジェクトの組み合わせ方の定義を以下に示す．

定義 3 オブジェクトの組み合わせ方
オブジェクトの集合 $p = \{t_{p1}, t_{p2}, \cdots, t_{pN}\}$ と，関数の集合 $F = \{f_1, f_2, \cdots, f_m\}$ が与

えられるとする．t_{pi} は，データの集合 p における i 番目の要素を表し，$t_{pi}[s_l]$ は，t_{pi} の第 l 番目の値を表す $(1 \leq l \leq m)$．f_h は関数の集合 F に含まれる関数であるとし，f_h は，複数の値 $t_{p1}[s_h], t_{p2}[s_h]$，$\cdots$，$t_{pk}[s_h]$ をスカラーにマッピングする関数とする $(1 \leq h \leq m)$．

このとき，データの組み合わせ g_p は以下のように計算できる．

$$
\begin{aligned}
g_p &= \{t_{p1}, t_{p2}, \cdots, t_{pk}\} \\
&= (f_1(t_{p1}.s_1, t_{p2}.s_1, , t_{pk}.s_1), \cdots \\
&\quad f_2(t_{p1}.s_2, t_{p2}.s_2, , t_{pk}.s_2), \cdots \\
&\quad f_d(t_{p1}.s_d, t_{p2}.s_d, , t_{pk}.s_d))
\end{aligned}
$$

Skyline となるデータには，本来順位という概念が存在しないため，以下のように順位付けの仕方を定義する．

定義 4 *Skyline* となるデータの組み合わせの順位付け
$S = s_1, s_2, \cdots, s_d$ を，ユーザが重視する順に並べ替えた次元の集合であるとする．すなわち，$i < j$ のとき，s_i は s_j よりも重視される．このとき，g_p，g_q をそれぞれ *Skyline* の条件を満たすオブジェクトの組み合わせであるとしたとき，g_p が g_q よりも高い評価値を得るのは，以下の 2 つの条件のうちいずれかを満たした場合である．

1. $f_1(g_p.s_1) > f_1(g_q.s_1)$ が成り立つ
2. $i = 1, \cdots, l (l < d)$ に関して，$f_i(g_p.s_i) = f_i(g_q.s_i)$ が成り立ち，$f_{l+1}(g_p.s_l) > f_{l+1}(g_q.s_l)$ が成り立つ

今回は，評価値のよいオブジェクトの組み合わせ上位 k 組を求めるのではなく，Skyline となる組み合わせに対して順位付けを行わなければならない．そのため，各データを評価値の高い順に並び替え，計算途中で閾値を求めることで終了判定を行う方法は難しいと言える．そこで，オブジェクトの組み合わせを効率よく生成する方針をとる．

組み合わせの生成は，Solution Table と呼ばれる 2 次元の表を利用して行う (図 2.6)．n 個のオブジェクトから，c 個を使ったオブジェクトの組み合わせの生成は，以下のように表される．

$$CT(n, c) = CT(n-1, c) \cup \Big(CT(n-1, c-1) \oplus t_n\Big) \tag{2.2}$$

式 (2.2) は，組み合わせの計算式 ${}_nC_c = {}_{n-1}C_c + {}_{n-1}C_{c-1}$ を元にしたものである．$CT(n,c)$ とは，$n(1 \leq n \leq N)$ 番目までの n 個のオブジェクトから c 個のオブジェクトを組み合わせてできる組み合わせを表し，$n < c$ もしくは $c = 0$ のとき，$CT(n,c) = \phi$ となる．$CT(n-1,c-1) \oplus t_n$ は，$n-1$ 番目までの $n-1$ 個のオブジェクトから $c-1$ 個選んでできる全ての組み合わせに，t_n を要素として加えたものを表す．例としてデータセットが $\{t_1, t_2, t_3 t_4\}$ ときを示す．$CT(3,2) = \{(t_1,t_2),(t_1,t_3),(t_2,t_3)\}$ となり，$CT(3,2) \oplus t_4 = \{(t_1,t_2,t_4),(t_1,t_3,t_4),(t_2,t_3,t_4)\}$ となる．

組み合わせの生成は，図 2.6 で表されるように，オブジェクト数が少ないものから進んでいく．全ての組み合わせを生成するのは非常にコストがかかってしまうため，親子関係を利用した枝刈りを行う．親子関係の定義を以下に示す．

定義 5 親子関係
g'_p が g_p の子 *(descendant)* であるとは，以下の条件を満たすものである．

1. g_p 自身
2. g_p に他のオブジェクトを追加したもの

ある組み合わせ g から生成される子のなかで，最も評価値の高い MPD(Most Prefered Descendant) と呼ばれる特別な子を考える．現在までに求められた解と MPD を比較し，もし現在の解よりもよい解が得られるのならば MPD 以外の子の計算を行い，そうでなければ枝刈りする．MPD の数は実際の解の数よりも少なく，高速に生成できるため，計算コストを小さく抑えることができる．

2.2.2 確率的に Join が成立する場合の Top-k Join

概要 Complex Search Task とは，複雑なクエリによる高度な検索を実現することを目標とした検索タスクである [4]．ここでの複雑なクエリとは，例えば，“どこに行けば自分の専門分野の学会に参加しつつ，海でのんびり休養できるか” や，“年少人口の間で流行っている病気について，最も危険なリスク要因は何か”，のように，回答するためには複数のデータベースを横断的に調べる必要のあるもののことを言う．一般的な検索システムとの違いは，複数のデータベースを調べて回答を作成する点である．

前述した TA，NRA，CA も Top-k Join のアルゴリズムである．しかし，これらはどのオブジェクトも必ず Join 可能である場合を想定したアルゴリズムである．今

回は，複数のデータベースに対して Join を行うため，必ずしも Join 可能であるとは限らない．先ほどの例で言えば，東京で開催される学会に対して沖縄のビーチを Join して一緒に提示することに相当する．そのため，これまでの Top-k Join の手法をそのまま適用することはできない．それに対して Shalem らは，Join が可能でないことを考慮した Top-k Join の手法を提案した [16].

Complex Search Task では，クエリ Q が複数のサブクエリ (ある単一のデータベースに対して発行するクエリ)$q_1, q_2, \ldots, q_m$ に分解可能である．また，クエリへの回答はサブクエリの回答の組み合わせで表される．このとき，Top-k Join では，各サブクエリの回答リストから最大で 1 つずつオブジェクトを選んで組み合わせを作り，スコアの高い上位 k 件を求める．これは，既存の限られた範囲に関して行う検索技術が大きく進歩したため，与えられたクエリを，それぞれのデータベースにあった形に変えることができれば，各部分解から回答を構築することができるというアイデアに基づくものである．

複合クエリ　基本クエリ q は，ある単一のデータベースを対象としたクエリであり，1 つのデータベースを調べることで解が得られるものであるとする．複合クエリ Q は，基本クエリの集合で表される．すなわち，$\mathrm{Q} = (q_1, q_2, \ldots q_m)$ となる．複合クエリ Q を構成する各 q_i をサブクエリと呼ぶ．サブクエリ q_i の回答のリストを L_i で表す．L_i 中のオブジェクトは，各オブジェクトに対する得点の降順に整列されており，j 番目のオブジェクトを o^i_j で表す．サブクエリの回答リストの集合を $PreAns$ と呼び，$PreAns(\mathrm{Q}) = (L_1, L_2, \ldots, L_m)$ とする．

以上より，複合クエリ $\mathrm{Q} = (q_1, q_2, \ldots q_m)$ への回答は，サブクエリの回答リストの集合 $PreAns(\mathrm{Q}) = (L_1, L_2, \ldots, L_m)$ から作られる．

組み合わせについて　組み合わせ C は，$PreAns(\mathrm{Q}) = (L_1, L_2, \ldots, L_m))$ が与えられたときに，各 L_i から最大で 1 つずつオブジェクトを選び作られる．組み合わせが成立するためには，2 つのオブジェクトが Join 可能な場合は True を返す $Jcon(o^x_{i_x}, o^y_{j_y})$ が与えられるので，任意の 2 つのオブジェクトについて，$Jcon(o^x_{i_x}, o^y_{j_y})$ が True となる必要がある．また，組み合わせ C のスコアは，ある単調変化の評価関数が与えられるので，各オブジェクトのスコアから評価関数を利用して求める．

組み合わせ C が完全 (Complete) であるとは，$PreAns(\mathrm{Q}) = (L_1, L_2, \ldots, L_m))$ が与えられたときに，C が各 L_i から 1 つずつオブジェクトを選び作られるときである (式 (2.3))．組み合わせ C が最大 (Maximal) であるとは，$PreAns(\mathrm{Q}) = (L_1, L_2, \ldots, L_m))$

が与えられたときに，C が各 L_i から最大で 1 つずつオブジェクトを選んで作られ，かつ C の上位集合となるような組み合わせが存在しない場合である (式 (2.4)，式中の $\perp$ はオブジェクトを選ばない場合を表す)．オブジェクトを選ばない場合，そのリストから得られる得点は 0 であるとする．

$$C = (o^1_{i_1}, o^2_{i_2}, \ldots o^m_{i_m}) \in L_1 \times L_2 \times \cdots \times L_m \tag{2.3}$$

$$C = (o^1_{i_1}, o^2_{i_2}, \ldots o^m_{i_m}) \in (L_1 \cup \perp) \times (L_2 \cup \perp) \times \cdots \times (L_m \cup \perp) \tag{2.4}$$

問題定義　Top-k Join とは，これらの前提のもと，Q, $PreAns$(Q), k, $Jcon$，評価関数 f が与えられたときに，以下の条件を満たすオブジェクトの組み合わせ列 $G = (g_1, g_2, \ldots, g_k)$ を求めることである．

1. 各 g_i は最大組み合わせ
2. G は得点の降順に $g_1, \ldots, g_k$ に整列している．すなわち，$\forall 1 \leq i < j \leq k$: $score(g_i) > score(g_j)$ が成り立つ．
3. 任意の可能なオブジェクトの組み合わせ $g' \notin G$ について，$score(g_k) > score(g')$ が成り立つ．

解法　ここでは，Shalem らが提案した Top-k Join を解くためのアルゴリズムである VNL(The Vertical Nested Loop Algorithm)，HNL(The Horizontal Nested Loop Algorithm)，VNL と HNL を合わせた手法である VHNL を紹介する．

VNL　VNL は，組み合わせを生成するための基本的な手法である Nested Loop を改良した手法である (アルゴリズム 1)．ループの途中で逐時 Join が成立するかを調べ，成立すれば次のループへ進み，そうでなければやり直す．アルゴリズム 1 中の $MPS(C)$ は，現在注目している組み合わせ C が取り得る最大スコア (Maximum Possible Score) を表す．$MPS(C)$ が解候補における k 番目に大きい得点 $score(g_k)$ よりも小さければ，ループを抜ける．

VNL はシンプルな Nested Loop に近い手法であるため，Join が成功する確率が高いとき，後述する HNL とは処理速度の面で大きく水を開けられてしまう．しかし，オブジェクトを選ばない選択肢が早い段階で出現するため，Join する確率が低い場合に有利であると言える．

HNL　HNL では，各 L_i 中の得点の高いものから順に Join を行う (アルゴリズム 2)．そのため，Join が成功する確率が高いほど高速に処理が可能である．アルゴリズム 2 中の T_j^i は，L_i において，先頭から j 番目までのオブジェクトの列を示す．

しかし，Join が成功する確率が低く，組み合わせ中にオブジェクトが選ばれないリストが存在する場合，処理効率が落ちてしまう．なぜならば，オブジェクトを選ばない選択を得点 0 のオブジェクトとして扱うので、得点の高い順に組み合わせを生成していくため，出現するのは組み合わせ生成の最後であるからだ．

VHNL　VHNL は，ある閾値を設け，前半部分は VNL，閾値以降の後半部分は HNL を利用する手法である．

今，Join が成功する確率を P_c とすると，P_c は式 (2.5) により求められる．

$$P_c = \frac{Join\text{ が成功する組み合わせの数}}{PreAns(\mathrm{Q})\text{ から 1 つずつオブジェクトを選んでできる全組み合わせ数}} \tag{2.5}$$

切り替えのための閾値 h は，P_c を利用して以下のように求められる．各リストについて，先頭から h 個調べた場合に成立する組み合わせ数の期待値 X は，サブクエリ数を m として，$X = P_c h^m$ と表せる．必要な組み合わせの数は k 個であるため，$X = k$ を代入して，h に関する式にすると，式 (2.6) となる．

$$h = \sqrt[m]{\frac{k}{P_c}} \tag{2.6}$$

この式 (2.6) から得られる h は，少なくとも先頭から h 個調べなければ，組み合わせを k 個得ることはできない，ということを意味している．ここで得られることが期待される k 個の組み合わせはそのまま解となる可能性が高い．したがって，各リスト先頭から h 個は虱潰しに調べ，以後は枝刈りを狙う方針をとることで，処理効率をあげることが期待できる．以上より，閾値 h 以前は VNL を行うことで全て探索し，閾値以後は枝刈りを行うべく HNL を利用して組み合わせを求める．

Algorithm 1 VNL

Input: Query Q, $PreAns(\mathrm{Q}) = (L_1, L_2, \ldots, L_m)$, the number of answer k, $Jcon$, scoring function f

Output: The k highest score combinations of objects $G = (g_1, g_2, \ldots, g_k)$

```
C = {} , G = {}
for i_1 = 1 to length(L_1) do
    add o^1_{i_1} to C
    if score(C) < MPS(C) then
        break
    end if
    for i_2 = 1 to length(L_2) do
        add o^2_{i_2} to C
        if there is at least one pair that Jcon returns False then
            delete o^2_{i_2}
        end if
        if score < MPS(C) then
            break
        end if
        ⋮
        for i_m = 1 to length(L_m) do
            add o^m_{i_m} to C
            if there is a pair that Jcon returns False then
                delete o^m_{i_m}
            end if
            if score(C) < MPS(C) then
                break
            end if
            if score(C) > score(g_k) then
                delete g_k
                add C to G
            end if
        end for
    end for
    ⋮
end for
```

Algorithm 2 HNL

Input: Query Q, $PreAns(\mathrm{Q}) = (L_1, L_2, \ldots, L_m)$, the number of answer k, $Jcon$, scoring function f

Output: The k highest score combinations of objects $G = (g_1, g_2, \ldots, g_k)$

```
C = {}, G = {}
for i = 1 to maxlen(L_1, L_2, ..., L_m) do
    for j = 1 to m do
        for all C ∈ (T_i^1, T_i^2, ..., {o_i^j}, T_{i-1}^{j+1}, ... T_{i-1}^m) do
            if there are no pair that Jcon returns False in C then
                if |G| = k and score(C) > score(g_k) then
                    delete g_k and add C to G
                else
                    add C to G
                end if
            end if
        end for
        if ∀j : MPS(o_{i+1}^j) < score(g_k) then
            break
        end if
    end for
end for
```

第 3 章
クラスタリングを利用した距離尺度の Top-k 組み合わせ検索

3.1 概要

本章では，クエリとの類似度を測る尺度として Euclid 距離を利用した，ベクトル空間での Top-k 組み合わせ検索について取り組む．組み合わせを対象にしない Top-k 検索では，オブジェクトの各属性に対して，評価関数として単調変化する性質のものを前提とするものがほとんどであるため，既存の手法をそのまま適用することは難しい．そこで，クラスタリングを利用した枝刈り手法を提案する．

本章の構成を以下にまとめる．まず始めに，本章の扱う問題の範囲を定義する．そして，提案手法であるクラスタリングを応用した枝刈り手法について説明した後，2 種類のデータを使用して実験を行い，提案手法の有効性を示す．

3.2 問題定義

本章では，以降の議論で使う記号，記法の定義を行う．大文字，太字，筆記体 ($\mathcal{D}$) をそれぞれ，集合，ベクトル，集合の集合を表すとする．データベース中にオブジェクトの集合 S があるとする．データベース中の各オブジェクトは d 次元の特徴ベクトルとして表される．2 つのオブジェクト $\mathbf{v}$，$\mathbf{u}$ 間の距離は，$\|\mathbf{v}-\mathbf{u}\|$ とする．オブジェクト同士の組み合わせは，各オブジェクトの特徴ベクトルの和として表す．d 次元のクエリベクトル $\mathbf{q}$ が与えられたとき，オブジェクトの組み合わせ $O \subseteq S$ との距離は，以下の式 3.1 で与えられる．

$$d(\mathbf{q}, O) := \|\mathbf{q} - \sum_{\mathbf{v} \in O} \mathbf{v}\| \tag{3.1}$$

本研究では，クエリ $\mathbf{q}$ との距離が小さいもののうち，最も近い上位 k 件のオブジェクトの組み合わせ $(O_1, O_2, \cdots, O_k)$ を求める問題について取り組む．$(O_1, O_2, \cdots, O_k)$ は，$1 \leq i \leq k$ について，以下の 2 つの条件を満たす．

- 組み合わせ O_i の要素数 $|O_i|$ は，$|O_i| \leq h$
- $O \in 2^S - \{O_1, \cdots, O_{i-1}\}$ について，$d(\mathbf{q}, O_i) \leq d(\mathbf{q}, O)$

ここで，2^S はオブジェクト集合 S のべき集合であるとする．この問題を，以後，*top-k combinatorial search* と呼ぶ．

3.3 既存解法と問題

Top-k combinatorial search は，単純には，以下のように全ての組み合わせを生成することにより解くことができる，この手法を *naive method* と呼ぶ．

step (1): 全てのオブジェクトの組み合わせ $\{O \mid O \subseteq S, |O| \leq h\}$ に対して，クエリベクトル $\mathbf{q}$ との距離を計算する

step (2): オブジェクトの組み合わせ O を，距離 $d(\mathbf{q}, O)$ をキーとして昇順に並べかえる．

step (3): 上位 k 組を出力する

具体的には，ループを入れ子にする nested loop と呼ばれる手法を用いて組み合わせの計算を行う [28].

この手法には以下の 2 つの点で問題がある．

1. 非常に大量の組み合わせを全て生成しなければならない
2. 評価関数が距離であるため，既存の単調増加関数を評価関数に利用している高速化手法を応用することが難しい

オブジェクト数を $|S|$ とすると，naive method における組み合わせ数 N は式 (3.2) で得られる．式 (3.2) からわかるとおり，時間計算量は $\mathrm{O}(|S|^h)$ となるため，1 つの組み合わせ内に含まれるオブジェクトの上限 h が大きくなると，N が爆発的に増加してしまう．そのため，何らかの対策が必要である．

$$N = \sum_{i=0}^{h} {}_{|S|}C_i \approx \mathrm{O}(|S|^h) \tag{3.2}$$

3.4 クラスタリングを利用した枝刈り手法

同じく組み合わせを扱う問題として，Association Rule Mining が挙げられる [8]．こちらの問題は，トランザクション中に頻出するオブジェクトの組み合わせを求めるものである．単純には，naive method と同じく可能な全ての組み合わせを列挙して，それらの頻度を調べれば，必要なオブジェクトの組み合わせを求めることができる．この問題に対する高速化手法として最も有名なものは，Apriori アルゴリズムである．Apriori アルゴリズムでは，ある集合が頻出であるためには，その部分集合が頻出である必要がある，という規則をもとに，解の絞り込みを行う．

同様に，これまで出てきた組み合わせ検索の問題を見てみると，いずれも形式は違えど，直接組み合わせを求めるのではなく，何らかの条件を用いて絞り込みを行うことで，効率化に成功していることがわかる．先に述べた Apriori アルゴリズムであれば，ボトムアップに組み合わせを生成し，部分集合が頻出でないものを調べない点である．

絞り込みを行うために考えられる手法としては，索引付け，部分解を求める手法の 2 つが挙げられる．索引付けとは，目的のオブジェクトまで効率よく到達するために，特殊なデータ構造にオブジェクトを格納することを指す．この手法は，ディスクアクセスの効率化や，類似検索等幅広く応用されている．部分解を求める方法は，前章で述べた TASM が例として挙げられる [1]．TASM では，部分木全ての編集距離を調べるのではなく，処理の始めに，TASM-postorder と呼ばれるアルゴリズムで，クエリとの距離が小さくなる可能性のある部分木の抽出を行う．それらの抽出した部分木に対してクエリと距離計算を行うことで，効率化を達成した．Top-k Combinatorial Search での最も大きな問題は組み合わせを扱う点である．そこで，まず，類似するオブジェクトをクラスタリングし，代表点をつくる．そして，次にその代表点同士の組み合わせを考え，解の見当をつけることで計算の効率化を行う．

以上を踏まえて，提案手法は，前処理と，クエリ処理の 2 つの処理を行い，解を求める．前処理では，オブジェクトをクラスタに分割する．クエリ処理では，クラスタの組み合わせを求めて，解を絞り込んだのち，オブジェクトの組み合わせを求める．

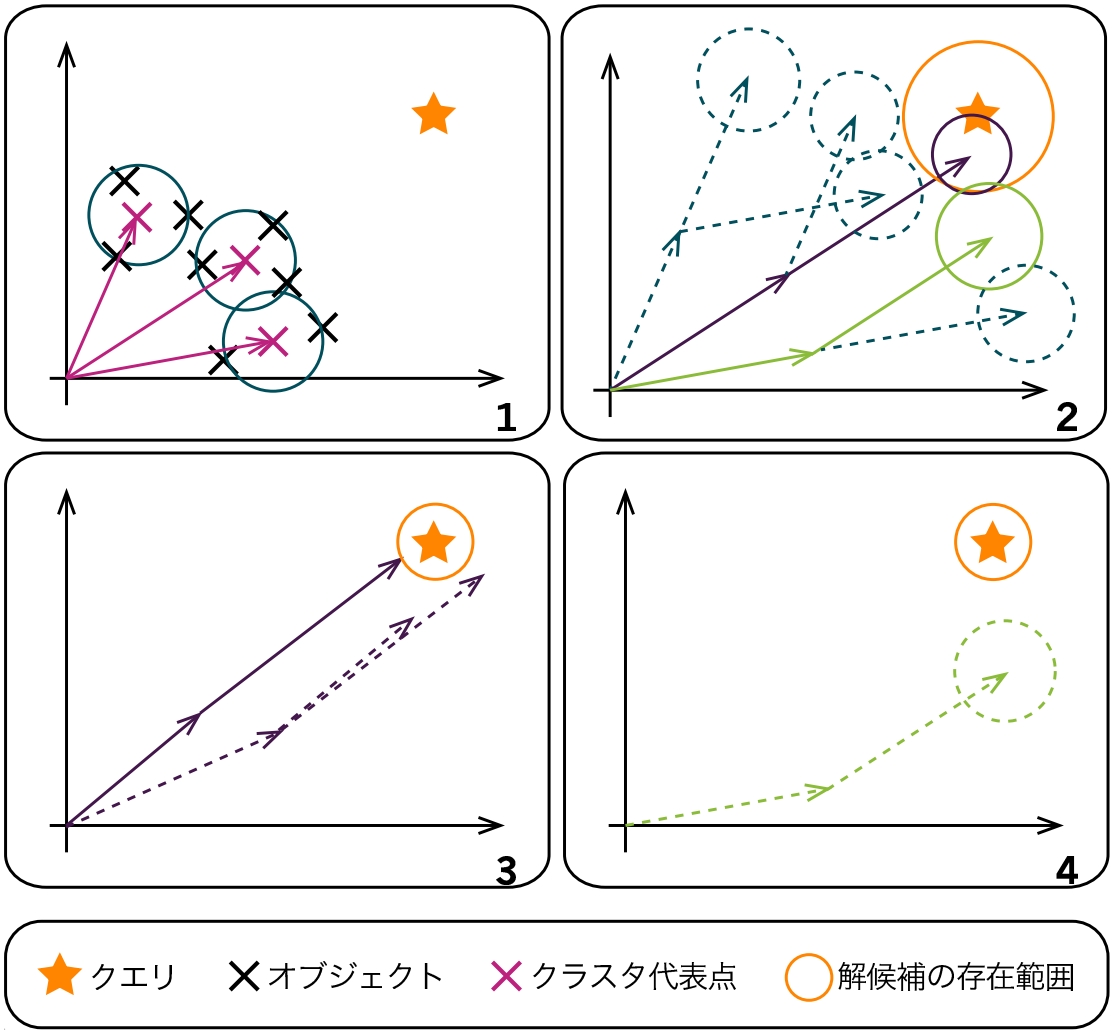

図 3.1: 処理の概要

3.4.1 提案手法の処理の概要

提案手法の流れは以下のとおりである (図 3.1).

1. オブジェクトをクラスタリングし，各クラスタにおいて代表点と半径を求める．図中の円はオブジェクトの存在する可能性のある範囲を示す．

2. クラスタ代表点の組み合わせを求め，解になり得ない組み合わせを除去する．図 3.1 の 2 コマ目中の点線で描かれている組み合わせは，解候補の存在範囲と，クラスタの組み合わせから作られるオブジェクトの組み合わせの存在範囲が重ならないため，解になり得ない．

3. 2. で求めたクラスタ代表点の組み合わせのうち，クエリとの距離が小さいも

のから順にオブジェクトの組み合わせを調べる．図 3 コマ目では，クエリとの距離が最も小さい図 2 コマ目の紫色で示されたクラスタの組み合わせから，オブジェクトの組み合わせを生成し，クエリとの距離を調べる．

4. 解候補が k 組集まったら，解候補中で最も距離の大きいものと，残っているクラスタ代表点の組み合わせを比較し，打ち切り判定を行う．図 4 コマ目では，図 3 コマ目で求めた解候補の存在範囲と，クラスタからできるオブジェクトの存在範囲が重ならないため，ここで処理を打ち切る．

3.4.2 クラスタリングを利用した前処理

データベース中のオブジェクト集合 S を，互いに共通要素を持たないクラスタ $\mathcal{C} =: \{C_1, C_2, \cdots, C_n\}$ に分割する．ここで行うクラスタリングに関して，どのようなクラスタリングアルゴリズムでも適用可能であるが，ここでは k-means 法を利用する*．

それぞれのクラスタ $C \in \mathcal{C}$ に対し，クラスタ重心 (式 (3.3)) 及びクラスタ半径 (式 (3.4)) を求める．また，クラスタ中の全てのオブジェクト $\mathbf{v}$ に対し，クラスタ重心からの相対ベクトル (式 (3.5))，とその長さ $\|\mathbf{r}_v\|$ を求めておく (図 3.2，図 3.3)．

$$\mathbf{m}_C := \frac{1}{|C|} \sum_{\mathbf{v} \in C} \mathbf{v} \tag{3.3}$$

$$\mathbf{m}_C := \frac{1}{|C|} \sum_{\mathbf{v} \in C} \mathbf{v} \tag{3.4}$$

$$\mathbf{r}_v := \mathbf{v} - \mathbf{m}_C \tag{3.5}$$

3.4.3 クエリ処理

クラスタ集合の部分集合 $\mathcal{D}$ を，クラスタの組み合わせと呼ぶ．クエリ処理では，クエリ $\mathbf{q}$ からの距離の小さい上位 k 組に入るようなオブジェクトの組み合わせを含む，クラスタの組み合わせを求める．オブジェクトの組み合わせとは異なり，1 つのクラスタから複数のオブジェクトが選ばれることもあり得るため，クラスタ

*k-means の “k” はクラスタ数を表す．クラスタ数を表す “k” と，求める解の数を表す “k” は無関係である．

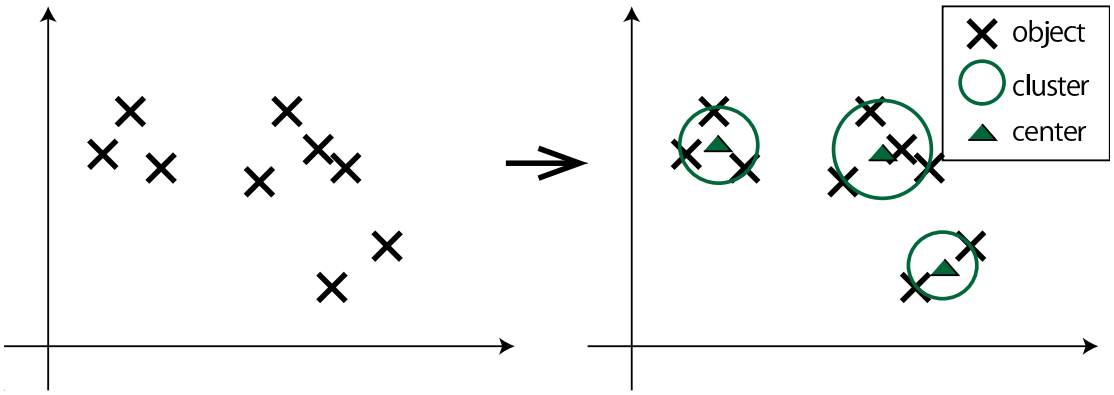

図 3.2: オブジェクトのクラスタリング

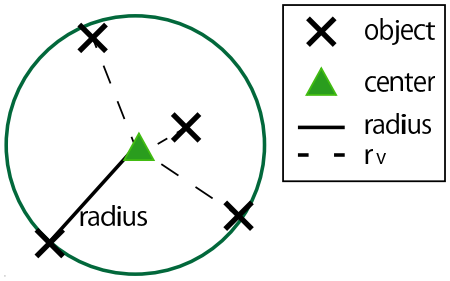

図 3.3: 相対ベクトル，クラスタ半径，クラスタ重心

の組み合わせは，最大要素数 h のクラスタ集合の多重集合†となる．例えば，クラスタの組み合わせ $\{C_1, C_1, C_2\}$ が与えられたとき，クラスタ C_1 から 2 個，クラスタ C_2 から 1 個オブジェクトを選び，オブジェクトの組み合わせを作成する．すなわち，一般的には，クラスタの組み合わせ $\mathcal{D} = \{C_1, \cdots, C_l\}$ から，オブジェクトの組み合わせ $\{\{\mathbf{v}_1, \cdots, \mathbf{v}_l\} \mid \mathbf{v}_1 \in C_1, \cdots, \mathbf{v}_l \in C_l\}$ が生成される．以後，クラスタの組み合わせ $\mathcal{D}$ に含まれるオブジェクトの組み合わせを $os(\mathcal{D})$ と表す．$\mathcal{D}$ から得られるオブジェクトの組み合わせの数は，クラスタ C_i を選んだ回数を $m(C_i)$ とすると，以下の式 (3.6) により与えられる．

$$\prod_{i=1}^{l} {}_{|C_i|}C_{m(C_i)} \tag{3.6}$$

要素数が最大 m のクラスタの組み合わせの集合を，$\mathcal{C}^m$ と表す．これは，例えば，前処理で，クラスタ集合 $\mathcal{C} = \{C_1, C_2, C_3\}$ が得られたとき，1 つの組み合わせ内の最大要素数が 2 の場合，$\mathcal{C}^2$ は．$\mathcal{C}^2 = \{\{C_1\}, \{C_2\}, \{C_3\}, \{C_1, C_1\}, \{C_1, C_2\}, \{C_1, C_3\},$

†多重集合とは，1 つの集合に同じ元が複数含まれるものを指す．

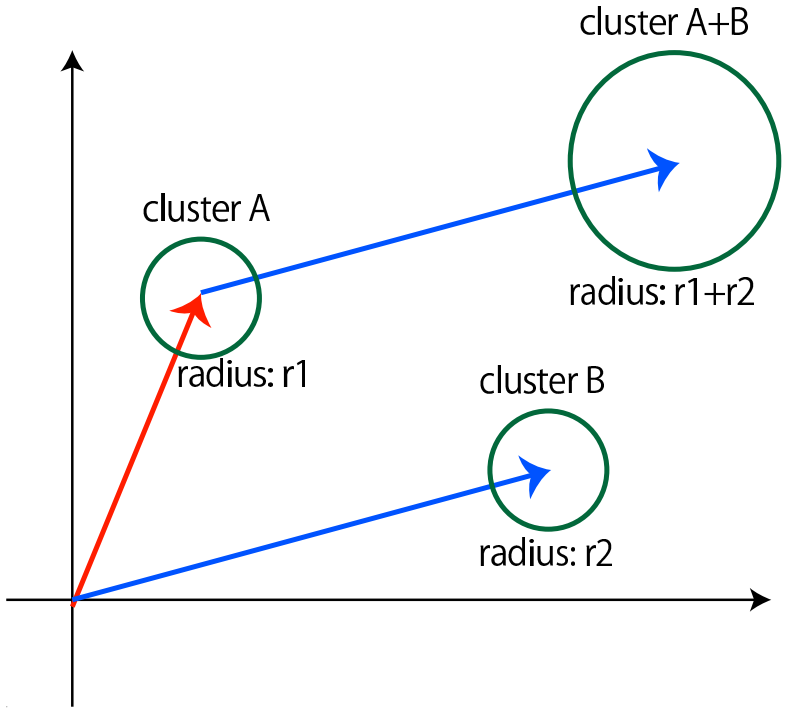

図 3.4: クラスタの組み合わせ

$\{C_2, C_2\}, \{C_2, C_3\}, \{C_3, C_3\}\}$ と，表される．ただし，クラスタに含まれるオブジェクト数以上に同じクラスタを選ぶことはできない．このように，クラスタの組み合わせから生成されるオブジェクトの組み合わせ $\cup_{\mathcal{D} \in C^m} os(\mathcal{D})$ は，オブジェクト集合から得られるオブジェクトの組み合わせと等価であることは明らかである．

ベクトル空間上では，クラスタの組み合わせは，クラスタ重心とクラスタ半径を足し合わせることで表される (図 3.4)．同じクラスタを選んだ場合，同じクラスタ重心と半径を足し合わせる．ただし，クラスタ内のオブジェクト数とクラスタを同数選んだ場合，同数選ばれたクラスタ半径を 0 として扱う．

クラスタの組み合わせ C^h に対して枝刈りを行うことを考える．クラスタの組み合わせ $\mathcal{D} = \{C_1, \cdots, C_l\}$ 中に含まれる，全てのオブジェクトの組み合わせ $O = \{\mathbf{v}_1, \cdots, \mathbf{v}_l\}$ ($\mathbf{v}_i$ はクラスタ C_i 中のオブジェクト) とクエリ $\mathbf{q}$ との距離は，以下の式

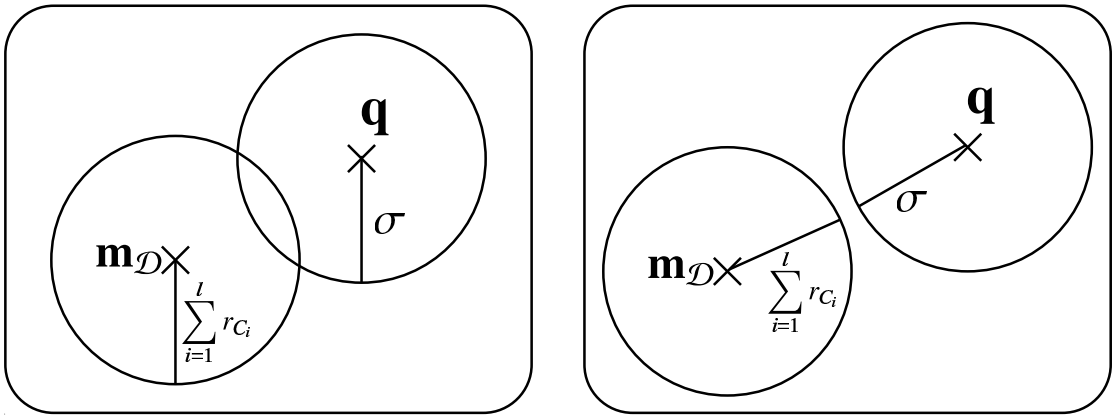

図 3.5: クラスタ重心について，最もクエリから離れている組み合わせとクエリとの距離 σ を利用して枝刈りを行う．図左のように重なる部分があれば σ よりも小さくなるオブジェクトの組み合わせを含む可能性があるため，詳細に調べるが，図右のように重なる部分がなければ，このクラスタの組み合わせに関しては以後調べない．

(3.7) により下限が与えられる．

$$\begin{aligned} d(\mathbf{q}, O) &= \|\mathbf{q} - \sum_{i=1}^{l} \mathbf{v}_i\| = \|\mathbf{q} - \sum_{i=1}^{l} \mathbf{m}_{C_i} - \sum_{i=1}^{l} \mathbf{r}_{v_i}\| \\ &\geq \|\mathbf{q} - \sum_{i=1}^{l} \mathbf{m}_{C_i}\| - \sum_{i=1}^{l} \|\mathbf{r}_{v_i}\| \\ &= d(\mathbf{q}, \mathbf{m}_{\mathcal{D}}) - \sum_{i=1}^{l} \|\mathbf{r}_{v_i}\| \\ &:= \underline{d}(\mathbf{q}, O) \end{aligned} \tag{3.7}$$

$\mathbf{m}_{C_i}$ と $\mathbf{r}_{v_i}$ はそれぞれクラスタ C_i の重心，クラスタ C_i の重心からオブジェクトへの相対ベクトルを表す，$\mathbf{m}_{\mathcal{D}}$ はクラスタ重心を表すベクトルの和 $\sum_i \mathbf{m}_{C_i}$ を表し，これを，$\mathcal{D}$ におけるクラスタ重心の組み合わせと呼ぶ．

オブジェクトの組み合わせに対する距離の下限値 (式 (3.7)) より，クラスタの組み合わせに対して与えられる距離の下限値は以下式 (3.8) のようになる．

$$\underline{d}(\mathbf{q}, \mathcal{D}) := \min_{O \in os(\mathcal{D})} \underline{d}(\mathbf{q}, O) = d(\mathbf{q}, \mathbf{m}_{\mathcal{D}}) - \sum_{i=1}^{l} r_{C_i} \ . \tag{3.8}$$

式 (3.8) により与えられるクラスタの組み合わせとクエリとの距離の下限値は，ク

ラスタ重心の組み合わせ $\mathbf{m}_{\mathcal{D}}$ とクエリ $\mathbf{q}$ との距離を 1 度計算すれば求めることができる．なぜなら，前処理の段階で，それぞれのクラスタについてクラスタ半径 r_{C_i} を求めているからである．

TA アルゴリズムにあるように，上位 k 番目のオブジェクトの組み合わせとクエリとの距離を上限値 σ とする [7]．これと，クラスタの組み合わせ $\mathcal{D}$ とクエリとの距離の下限値 $\underline{d}(\mathbf{q}, \mathcal{D})$ を比較し，σ を超えるものに関して枝刈りを行う (図 3.5)．アルゴリズム 3 に，クエリ処理の概要を示す．このアルゴリズムは，入力として前処理で生成したクラスタ集合 C，1 つの組み合わせ内における最大要素数 h, 求める解の数 k，クエリ $\mathbf{q}$ をとり，クエリ $\mathbf{q}$ に対して k 近傍のオブジェクトの組み合わせを返す．

Algorithm 3 Top-k combinatorial search

Input: a set of cluster C, maximum cardinality h, the number of answers k, query $\mathbf{q}$
Output: a set of object sets $A = (O_1, O_2, \ldots, O_k)$

1: sort the cluster combinations in C^h in the ascending order according to the lower bound given by Eq. (3.8)
2: initialize the upper bound σ and the candidate set A of object combinations
3: **for** $i := 1$ to $|C^h|$ **do**
4: **if** $\underline{d}(\mathbf{q}, \mathcal{D}_i) > \sigma$ **then**
5: return A
6: **end if**
7: calculate distances between $\mathbf{q}$ and object combination in $\mathcal{D}_i$
8: update the candidate set A to the k-closest object combinations in $A \cup os(\mathcal{D}_i)$
9: $\sigma := \max_{\mathbf{v} \in A} d(\mathbf{q}, \mathbf{v})$
10: **end for**

アルゴリズム 3 中の 2 行目では，上限値 σ と解候補のオブジェクトの組み合わせ A を，それぞれ無限大，空集合に初期化する．σ が小さいほど，効率よく枝刈りが可能である．ここで，σ が比較的容易な計算で，より小さな値が設定できれば，2 行目以降での計算量を減らすことが可能である．現在，上限値 σ の初期化とオブジェクトの組み合わせの解候補を以下の 3 つの処理により行なっている．

1. C^h 中のクラスタの組み合わせを，クラスタ重心の組み合わせとクエリ間の距離 $d(\mathbf{q}, \mathbf{m}_{\mathcal{D}})$ を用いてで昇順に並べ替える．

2. 以下の 2 つの条件を満たす，クエリとの距離が最も小さい n 個のクラスタの組み合わせを求める．

- $|os(\mathcal{D}_1) \cup \cdots \cup os(\mathcal{D}_{n-1})| < k$
- $|os(\mathcal{D}_1) \cup \cdots \cup os(\mathcal{D}_n)| \geq k$.

3. $os(\mathcal{D}_1) \cup \cdots \cup os(\mathcal{D}_n)$ 中の，クエリとの距離が最も近いオブジェクトの組み合わせの上位 k 組を解候補集合 A とし，その中で最大の距離を σ とする．

4 行目にて，条件 $\underline{d}(\mathbf{q}, \mathcal{D}_i) > \sigma$ が満たされた場合，処理を終了する．これは，1 行目において，クラスタの組み合わせは距離の下限値により整列されているためである．それゆえ，$\mathcal{D}_i, \mathcal{D}_{i+1}, \cdots, \mathcal{D}_{|C^h|}$ から得られるオブジェクトの組み合わせは，4 行目における条件が満たされていれば，必ず上限値 σ よりも大きくなることが保証できる．

7 行目において，クエリ $\mathbf{q}$ からの k 近傍のオブジェクトの組み合わせは，解候補の集合 A と，$os(\mathcal{D}_i)$ から得られる．解を得るためには，先ほど定めた σ よりも距離が小さくなるものだけに関して調べれば充分である (図 3.6).

$$\{\mathbf{v} \in os(\mathcal{D}_i) \mid d(\mathbf{v}, \mathbf{q}) \leq \sigma\} \tag{3.9}$$

しかし，式 (3.9) で与えられる条件を満たすようなオブジェクトを求めるためには，$\mathcal{D}_i$ 中の全てのオブジェクトの組み合わせについて，クエリ $\mathbf{q}$ との距離を調べなければならない．ここで，式 (3.7) で与えられる条件ではなく，以下の式 (3.10) を利用して組み合わせを求めることを考える．

$$\{\mathbf{v} \in os(\mathcal{D}_i) \mid \underline{d}(\mathbf{v}, \mathbf{q}) \leq \sigma\}\,. \tag{3.10}$$

式 (3.10) を満たすオブジェクトの組み合わせは，式 (3.7) で与えられる条件を満たすことは明らかである．ただし，式 (3.10) を利用した条件を満たすかどうかは，クラスタ重心の組み合わせ $\mathcal{D}_i$ とクエリ $\mathbf{q}$ との距離を利用して判定することができる．なぜなら，前処理の段階でクラスタ重心から，クラスタを構成するそれぞれのオブジェクトへ伸びる相対ベクトルとその長さを求めているからである．これを利用して，アルゴリズム 3 の 7 行目における距離計算回数を削減し，計算量を削減することが可能である．

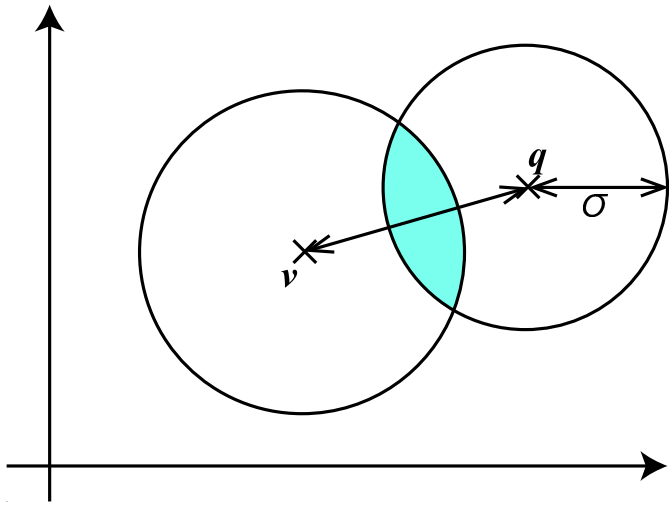

図 3.6: クラスタの組み合わせからできるオブジェクトの存在範囲と，距離の上限値 σ からできる解候補の存在範囲．重なっている部分に存在するオブジェクトのみを調べればよい．

3.4.4 主成分分析を利用した下限値の改善

クラスタの組み合わせ中のオブジェクトの組み合わせに関して，図 3.6 に示すとおり，クエリを中心として，距離の上限値 σ が作る円と，クラスタの組み合わせが作る円とが重なる部分に存在するオブジェクトの組み合わせのみを調べればよい．しかし，式 (3.7) で与えられる距離の下限値条件では，統合成立は，クラスタ内のオブジェクトが作る相対ベクトル $\mathbf{r}_{v_i}$ $(1 \leq i \leq l)$ が全て $\mathbf{q} - \mathbf{m}_{\mathcal{D}}$，すなわち，クラスタ重心の組み合わせとクエリが作るベクトルと同じ向きを向いている場合のみである．式 (3.7) で与えられる下限値では，クラスタ重心からの相対ベクトルの向きを考慮していない．よりよい下限値を求めるためには，相対ベクトルの向きを考慮する必要がある．しかし，各相対ベクトルに対して，クエリ $\mathbf{q}$ との向きを計算してしまうと，実質的には全解探索と変わらないほどの計算コストがかかってしまう．そこで，主成分分析を応用して，相対ベクトルの向きの近似を行う．事前にデータセットに対して主成分分析を行い，第 1 主成分へ射影を取ることで，クラスタ半径 r の代わりに，軸上での長さを枝刈りに利用することを考える (図 3.7).

$\mathbf{e}$ を第 1 主成分を表す単位ベクトルとする．主成分分析により，以下の式 (3.11)

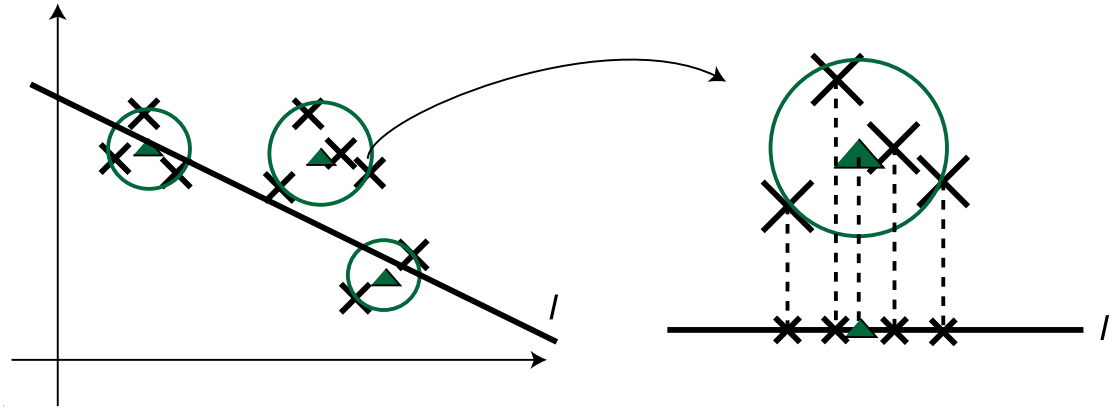

図 3.7: 第 1 主成分上へのオブジェクトの射影

のようにして下限値を得ることができる．

$$
\begin{aligned}
d(\mathbf{q}, O) &= \|\mathbf{q} - \sum_{i=1}^{l} \mathbf{v}_i\| = \|\mathbf{q} - \mathbf{m}_{\mathcal{D}} - \sum_{i=1}^{l} \mathbf{m}_{v_i}\| \\
&\geq |(\mathbf{q} - \sum_{i=1}^{l} \mathbf{m}_i - \sum_{i=1}^{l} \mathbf{v}'_i) \cdot \mathbf{e}| \\
&= |\mathbf{q} \cdot \mathbf{e} - \mathbf{m}_{\mathcal{D}} \cdot \mathbf{e} - \sum_{i=1}^{l} \mathbf{m}_{v_i} \cdot \mathbf{e}| \\
&:= \underline{d}_{PCA}(\mathbf{q}, O) \,. \qquad (3.11)
\end{aligned}
$$

第 3 項の $|\mathbf{m}_{v_i} \cdot \mathbf{e}|$ は，たいていの場合において $\|\mathbf{m}_{v_i}\|$ よりも小さくなる．そのため，下限値 $\underline{d}_{PCA}(\mathbf{q}, O)$ は，式 (3.7) により与えられる下限値 $\underline{d}(\mathbf{q}, O)$ よりも大きくなることが期待できる．

式 (3.11) 中の第 2 項，第 3 項は，前処理の段階で計算できる．第 1 項，すなわち，クエリと第 1 主成分を表す単位ベクトルとの内積を用いることで，オブジェクトの組み合わせとクエリ $\mathbf{q}$ との距離を計算せずとも，下限値を求めることができる．

クラスタの組み合わせにおいて，式 (3.11) から得られる下限値を適用すると，以下の式 (3.12) のように表すことができる．

$$
\underline{d}_{PCA}(\mathbf{q}, \mathcal{D}) := \min_{O \in os(\mathcal{D})} \underline{d}_{PCA}(\mathbf{q}, O) \,. \qquad (3.12)
$$

主成分分析を利用した下限値を利用するために，前処理に以下の 2 つの処理を加える．

- データベース中のオブジェクト集合 S に対して主成分分析を行い，第 1 主成分を表す単位ベクトル $\mathbf{e}$ を求める．
- それぞれのクラスタ重心 $\mathbf{m}$ と，オブジェクト $\mathbf{v}$ に対して，それぞれ $\mathbf{e}$ との内積 $\mathbf{m} \cdot \mathbf{e}$，$\mathbf{v} \cdot \mathbf{e}$ を求める．

主成分分析を利用した処理を加えるため，クエリ処理を以下のように変更する．

- アルゴリズム 3 の 1 行目において，式 (3.8) により与えられる下限値 $\underline{d}(\mathbf{q}, O)$ の代わりに，式 (3.12) により得られる下限値 $\underline{d}_{PCA}(\mathbf{q}, \mathcal{D})$ を利用する．
- アルゴリズム 3 の 4 行目と 7 行目において，式 (3.7) により与えられる下限値 $\underline{d}(\mathbf{q}, O)$ の代わりに，式 (3.12) により得られる下限値 $\underline{d}_{PCA}(\mathbf{q}, \mathcal{D})$ を利用する．

表 3.1: パラメータ設定

	N	k	h	# clusters
実験 1	300	1	3	30,60,90, $\cdots$, 300
実験 2(abalone)	100,200,$\cdots$,1500	1	3	$N \times 0.2$
実験 2(corel)	100,200,$\cdots$,1400	1	3	$N \times 0.7$
実験 3	300	1,10,100, $\cdots$, 100000	3	60
実験 4	100	1	2,3,4,5	20

3.5 評価

3.5.1 実験概要

提案手法の有効性を示すため，以下の 2 つのデータセットを用いて実験を行った．なお，データセットは，UCI Machine Learning Repository のものを使用した．

- abalone data set(以後，abalone とする): abalone は，4177 個の鮑から集めた生体データである．生体データは，大きさ，重さ等の 8 次元からなり，実験には，そのうち数値であるもの 7 次元を利用した．
- Corel Color Histogram(以後，Corel とする): Corel は，68040 個の画像データから得た色ヒストグラムであり，32 次元の特徴ベクトルからなる．

以下の 4 つのパラメータについて実験を行った．

- クラスタ数
- オブジェクト数 $|S|$
- 求める解の数 k
- 1 つの組み合わせ中の最大の要素数 h

表 3.1 に各パラメータの設定と実験の条件をまとめる．

以下の方法でクエリを生成した．(3) にて，平均ベクトルを h 倍するのは，計算時間が最もかかる設定であるのが，平均ベクトルの組み合わせの上限数倍がクエリとして与えられたときだからである．

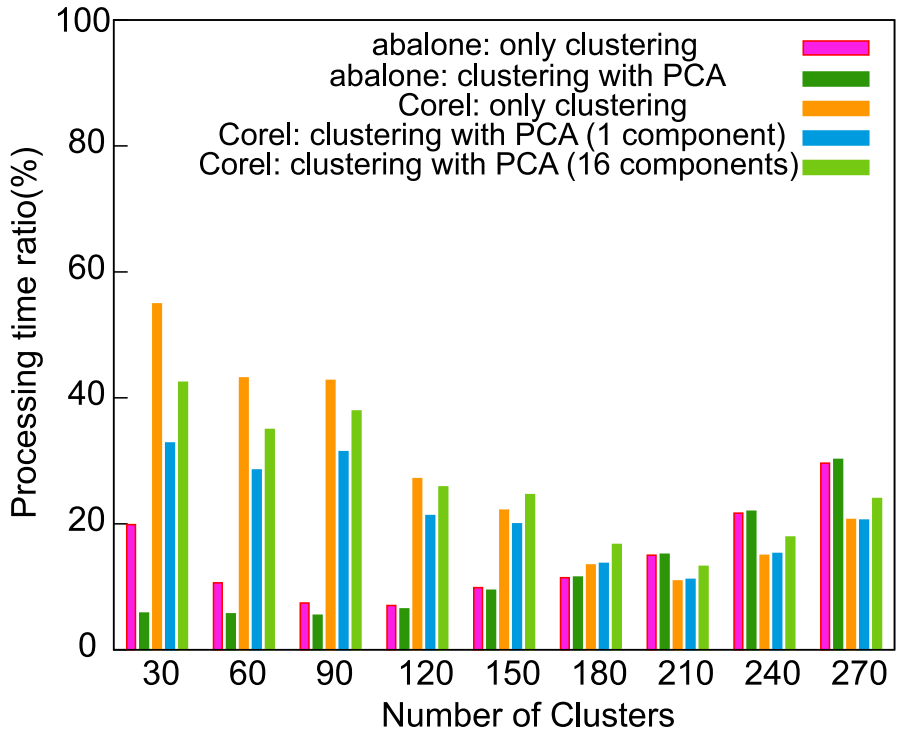

図 3.8: クラスタ数を変化させた場合の相対計算時間の変化

1. 100 個のオブジェクトをランダムに選ぶ
2. 選んだオブジェクトの平均をとる
3. 選んだオブジェクトの平均を，h 倍する

それぞれ異なるクエリ 50 個に対し，計算時間を計測し，その平均を結果とした．実験は，Intel(R) Xeon(R) X5492 3.2GHz プロセッサと 64GB の RAM を搭載した計算機上で行った．アルゴリズムの実装は C 言語で行い，GNU/Linux CentOS 上の gcc 4.4.4 でコンパイルを行った．

評価方法は，提案手法と，クラスタ数 3.3 章で述べた naive method との比較で行った．グラフに，提案手法と naive method の処理時間比をプロットした．グラフ中のラベル “only clustering” と “clustering with PCA” は，それぞれ，3.4 章のクラスタリングのみを利用した提案手法，3.4.4 章の主成分分析を利用してより効率的な距離の下限値を設定したものを示す．

3.5.2 結果と考察

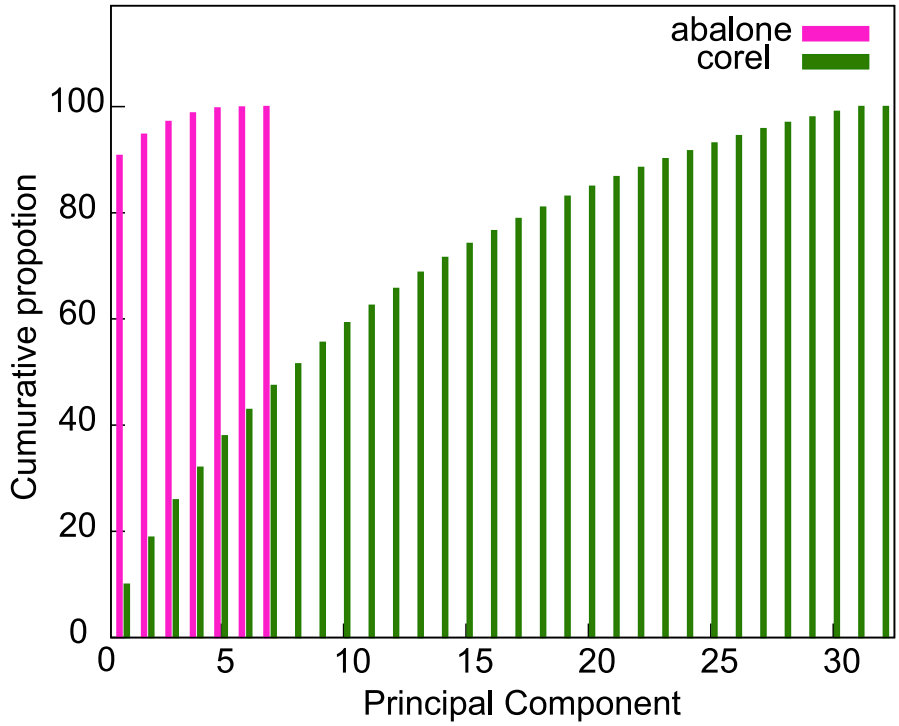

図 3.9: 主成分の累積寄与率

クラスタ数を変化させた場合　図 3.8 に，abalone と Corel においてクラスタ数を変化させたときの，naive method との相対処理時間を示す．縦軸が相対処理時間を示し，横軸がクラスタ数を表す．表 3.1 の 1 行目にクラスタ数以外のパラメータの設定をまとめる．このグラフから，以下の 3 つのことがわかる．

1. クラスタ数が性能に大きく影響するため，クラスタ数を適切に設定する必要がある

2. 主成分分析を利用した枝刈りは，特にクラスタ数が小さいとき，すなわち，クラスタ内のオブジェクトが多いときに有効である．

3. 第 1 主成分の寄与率が大きいほど，効率的に処理が可能である．

1 番目について述べる．naive method に対する処理時間は，最適なクラスタ数のときに，abalone で最高 1/20，Corel で最高 1/10 を達成した．この結果から，事前処理において，クラスタ数が適切であれば，クラスタリングを行うことは有効であることがわかる．また，クラスタ数が処理時間の改善に大きく影響することが図 3.9 からわかる．abalone では，クラスタ数が小さいときに 1/5 まで，クラスタ

数が大きい時に 1/3 まで性能が低下した．同様に Corel でも，クラスタ数が小さいときに最大 1/2，クラスタ数が大きいときには 1/5 まで性能が低下した．

クラスタリング後の計算量は，クラスタ数を N' とすることで，以下の式 (3.13) により与えられる．第 1 項は，クラスタを利用した絞り込みに必要な計算量で，第 2 項は，絞り込んだ後で解の決定に必要な計算量を表す．N' が小さければ小さいほど，調べなければならない組み合わせの数が小さくなるが，そのぶんクラスタ内の組み合わせ数が増えてしまうため，第 2 項が大きくなってしまう．そのため，クラスタ数は適切に設定する必要がある．

$$\sum_{i}^{h} {}_{N'}C_i + \alpha(k) \approx \mathrm{O}((N')^h) + \mathrm{O}(\alpha(k)) \tag{3.13}$$

2 番目について述べる．クラスタ数を増加させると，クラスタの組み合わせの数も増加することは明らかである．提案手法では，クラスタの組み合わせは全て生成しなければならないため，クラスタ数が増加すると式 (3.13) の第 1 項が大きくなる．しかし，クラスタ内に含まれるオブジェクトが少なくなるため，式 (3.13) の第 2 項が小さくなる．逆に，クラスタ数を少なくすると，式 (3.13) の第 1 項は小さくなるが，クラスタ内のオブジェクト数が大きくなるため，アルゴリズム 3 の 7 行目，すなわちクラスタの組み合わせから真の解を得るための計算量を示す第 2 項のが大きくなる．これらはいずれも提案手法の処理速度の律速となり得るが，クラスタ数が少ないときに主成分分析を応用した枝刈りを適用することで，律速成分を緩和することができた．これは，主成分分析を応用した条件 (式 (3.11)) を用いることで，ベクトルの向きを考慮することができるため，アルゴリズム 3 の 7 行目において，より効率的な枝刈りが行えるためである．

3 番目について述べる．第 1 主成分の寄与率が大きいほど，処理速度が向上する．これは，式 (3.11)，式 (3.12) により与えられる距離の下限値を，より適切な値に設定できるためである．図 3.9 に abalone および Corel における主成分の累積寄与率を示す．主成分分析を利用した高速化の効果は，第 1 主成分の寄与率が全体の 90%を占める abalone にて顕著に出ている．図 3.8 より，Corel では，第 1 主成分の寄与率の大きさが全体の 10%であるため，abalone ほどの効果が出ていないことがわかる．

主成分の寄与率の影響を詳細に調べるべく，第 1 から第 16 までの 16 個の主成分 (累積寄与率 75%) を利用した場合と，第 1 主成分のみを利用した場合とで比較を行った．16 個の主成分を利用する場合，アルゴリズム 3 のステップ 7 において，

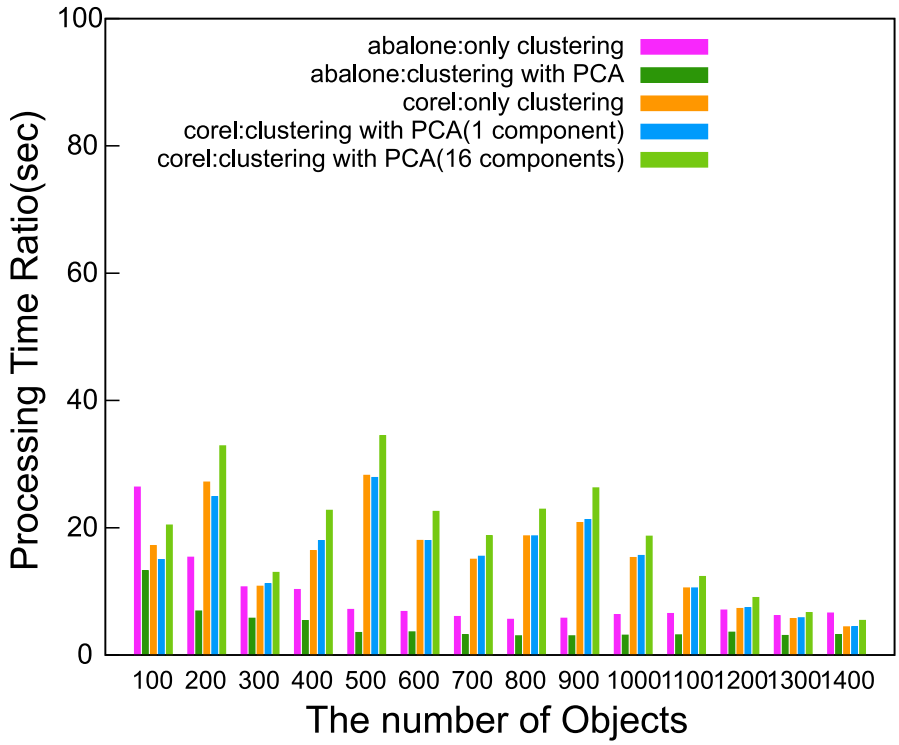

図 3.10: N が変化したときの相対計算時間

各主成分を利用してできる式 (3.12) の条件全てを満たすときのみ，オブジェクトの組み合わせを計算する．図 3.8 中の緑色の棒が第 1 主成分のみ 1 個を利用した場合，図中の青色の棒が，第 1 主成分から第 16 主成分を利用した結果を示す．第 1 主成分のみを用いたほうが，16 個の主成分を用いたものよりも処理時間が少ないことが分かる．原因として，次元数の増加が挙げられる．用いる主成分の数を増やすと，それに伴い下限値の計算回数が増加するため，主成分分析を応用することで得られる枝刈り効果を相殺してしまうことが原因であると考えられる．

オブジェクト数を変化させた場合の結果　図 3.10 に，オブジェクト数 $|S|$ を変化させた場合の相対処理時間を示す．パラメータ等の実験の設定は，表 3.1 の実験 3 の行にまとめる．図より，オブジェクト数の増加に対しても，abalone，corel それぞれに対して 1/20，1/6 を達成したことがわかる．しかし，オブジェクト数が小さいときは，あまり改善されていないことがわかる．そのため，クラスタを利用した枝刈りを行う際には，オブジェクト数がある程度多い場合でないと効果が少ないと言える．

図 3.11 にそれぞれ実際に処理にかかった時間を示す．縦軸が処理時間の対数，

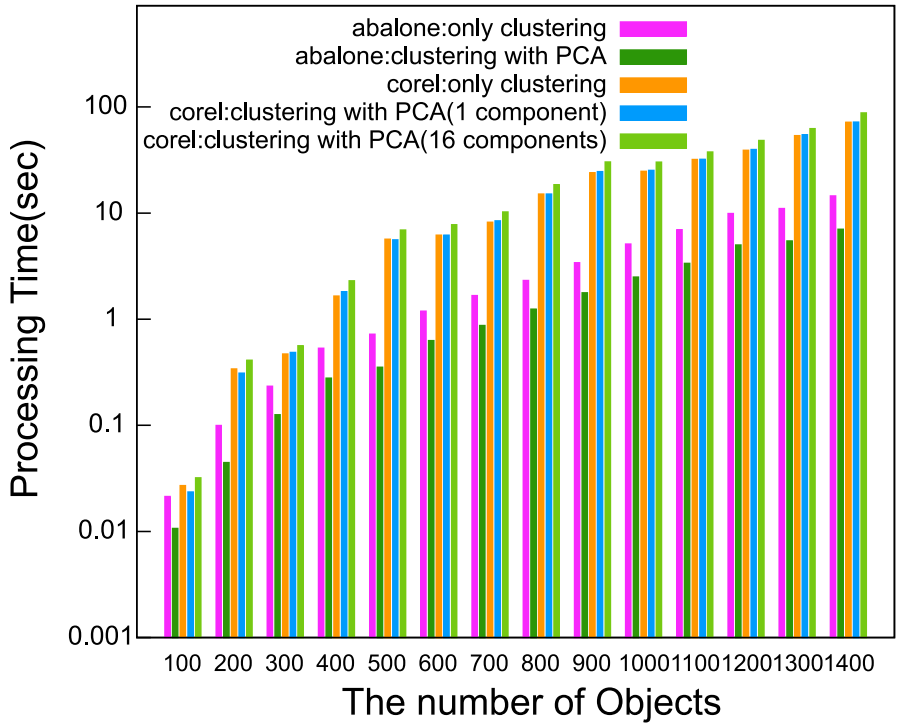

図 3.11: N が変化した時の実計算時間

横軸がオブジェクト数を表す．処理効率はよくなっているが，オブジェクト数の増加に伴って指数関数的に処理時間が増えているため，式 (3.13) に示す通り，提案手法は時間的計算量のオーダを変えられないことがわかった．そのため，まだスケーラビリティの面で課題が残っていると言える．

解の数 k を変化させた場合　図 3.12 に，求める解の数 k を変化させた場合の結果を示す．パラメータ等の実験条件は，表 3.1 の “実験 2” の行にまとめる．

図 3.12 から，k の増加に伴い計算効率が低下していることがわかる．これは，アルゴリズム 3 の 7 行目における上限値 σ が大きくなり，k に比例して緩くなってしまうため，効率が落ちてしまう．つまり，図 3.6 中のクエリを中心とした解候補の存在範囲が大きくなり，クラスタの組み合わせが解候補になりやすいことが原因である．

オブジェクトを組み合わせる数の変化　図 3.13 に，1 つの組み合わせが最大いくつのオブジェクトを含むことが出来るか，を表すパラメータ h を変化させた場合の相対処理時間を示す．パラメータ等の実験条件は，表 3.1 の “実験 3” にまとめ

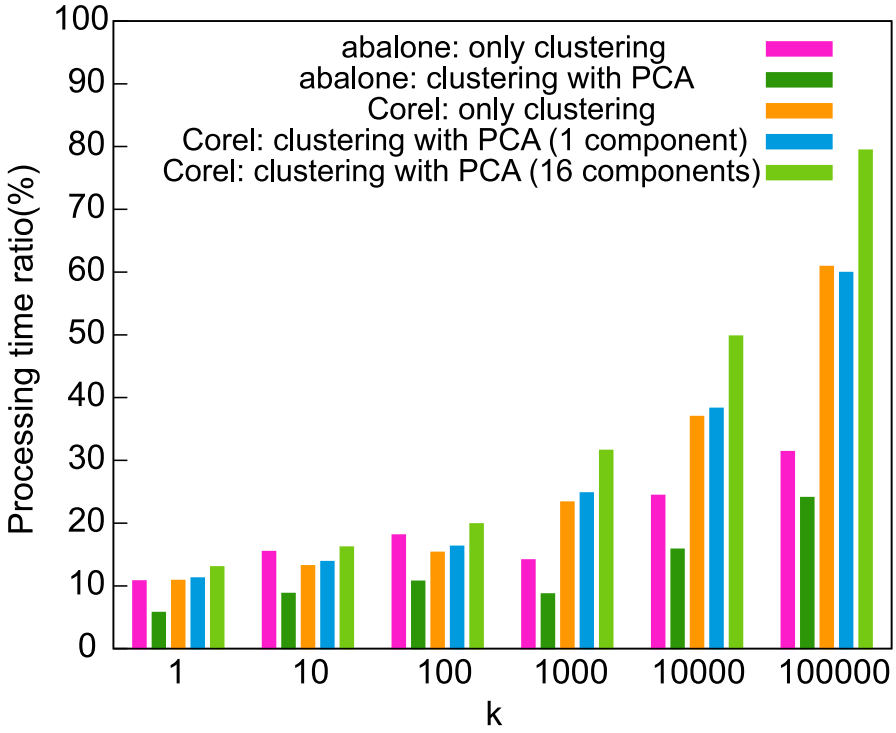

図 3.12: k を変化させた場合の相対計算時間の変化

た．図が示す通り，提案手法は 1 つの組み合わせ中の最大オブジェクト数を増やしても，効果が発揮できることがわかった．特に Corel では，$h = 2$ のときは 1/5 ほどの処理時間削減に留まったが，$h = 5$ のときは 1/50 まで削減できている．h の増加に対しても，提案手法は有利であるということができる．

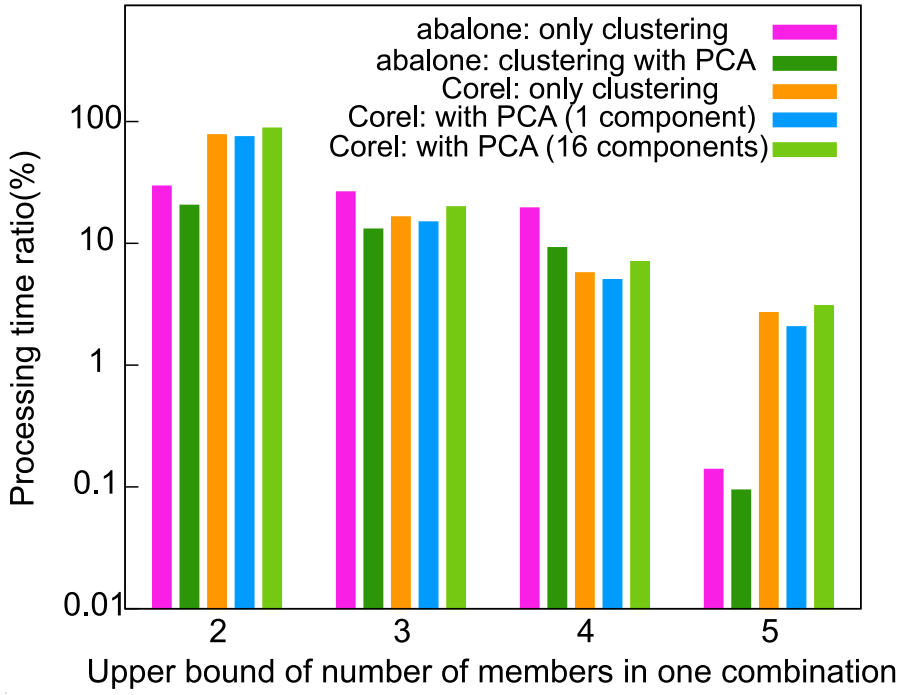

図 3.13: h を変化させた場合の相対計算時間の変化

3.6 本章のまとめ

本章では，ベクトル空間における距離尺度の Top-k 組み合わせ検索の提案を行った．探索方法の効率化手法として，クラスタリングを利用した枝刈り手法を提案した．また，さらなる効率化として，主成分分析を用いた下限値の決定法を提案した．これら提案手法により，nested loop を利用した naive method と比較して，最大で 20 倍ほどの性能向上を確認した．

今後の課題は以下の 2 つである．まず 1 つめとして，提案手法では，クラスタリングに加えて主成分分析を利用し，より適切な距離の下限値を設けることで処理効率をあげることに成功したが，第 1 主成分の寄与率が小さい場合や，次元数が大きい場合に性能向上に貢献できない．そのため，今後は，主成分分析以外の次元圧縮，次元削減手法の利用を検討する．2 つめは，クラスタ手法の検討である．今回はクラスタリング手法として k-means を採用したが，k-means では，適切なクラスタ数を探す必要があり，チューニングに大きなコストがかかってしまう．そこで，k-means 以外のクラスタ手法を試すことで，この問題に合ったクラスタ手法がどのようなものになるのかを検討する．

第 4 章
クラスタリングを利用した Top-k Join 高速化手法

4.1 概要

3 章では，クラスタリングを利用して，ベクトル空間における Top-k 組み合わせ検索の高速化を行った．本章では，3 章で有効性を確認した手法を，Complex Search Task における Top-k Join へ応用することを考える [4].

再度問題を確認する．Complex Search Task における Top-k Join とは，クエリ $Q = (q_1, q_2, \ldots, q_m)$ のサブクエリ q_i に対する回答リスト L_i が与えられた場合，各 L_i から 1 つずつオブジェクトを選んで組み合わせを作成し，その中で最も得点の高い上位 k 組を求めるというものであった．2 章で紹介した，Shalem らが提案した既存手法では，全てのサブクエリの回答列 $L_1, \ldots, L_m$ を同期的に処理する [16]. しかし，Wu らの提案によれば，各 L_i について非同期で処理を行うことで，処理効率を挙げられる可能性がある [22]．また，Shalem らの提案は非常にシンプルな方法であり，同時に 3 章で扱った問題の特殊形であるため，クラスタリングを行うことで処理速度の安定化を図ることが可能であると考えられる．

3 章との対応を確認する．各サブクエリ q_i を次元の 1 つとみなすと，q_i の回答リスト L_i 中のオブジェクトは，ベクトル空間上の座標 $(0, 0, \ldots, score(o_j^i), 0, \ldots, 0)$ にマッピング可能である．また，各オブジェクトの得点を単調変化関数，すなわち重み付きベクトルとの内積で計算され，得点の高いものから順位付けが行われる．ここでの最も高い得点とは，ベクトル空間上では最も長いベクトルであることと同義であり，これは同時にクエリを無限遠点においたときのオブジェクトの組み合わせとの距離と等しい．以上から，Top-k Join は，3 章の特殊な形だとみなすことができる．対応を表 4.1 にまとめる．

提案手法は，既存の VNL，HNL にクラスタリングを適用した CVNL，CHNL の 2 つである．

本章の構成を以下にまとめる．まずはじめに，クラスタリングを利用した Top-k Join 手法である CVNL，CHNL の説明を行い，実験によって有効性を確かめる．次

表 4.1: 3 章と本章の対応

	3 章	4 章
クエリ	ベクトル空間上の任意の点	無限遠点
オブジェクト	ベクトル空間上の任意の点	軸上
組み合わせ	任意のオブジェクト	各 L_i から 1 つずつ
組み合わせ数	上限値を与える	軸の数

に，システム全体の最適化に必要な処理コストの予測手法を提案し，実データおよび人工データを用いて提案手法の性質について議論する．

4.2 クラスタリングを利用した高速化手法

本章と3章の最も大きな違いは組み合わせ方である．3章では，1つの組み合わせ中のオブジェクト数に上限hが与えられ，任意のオブジェクトに対して組み合わせを行うのであるが，本章では，軸の数だけオブジェクトを組み合わせる．また，各軸上にのみオブジェクトが存在する特徴がある．これまでは，空間上に散らばったオブジェクトに対して，k-meansを使ってオブジェクトのクラスタリングを行い，ある程度まとまりをつけることで絞り込みを行なっていた．しかし，既に軸上に存在することがわかっており，また，各軸から1つずつオブジェクトを選ぶということが決まっているため，k-meansのようにオブジェクトに関するクラスタリングは有益ではない．したがって，各サブクエリの回答リストに対してクラスタリングを行う．3章では，前提として，検索対象データがあらかじめ与えられていたため，k-meansや主成分分析と言った事前処理を行うことができた．今回は，クエリが決まって初めて処理対象のデータ，すなわちサブクエリに対する回答のリストが得られるため，事前処理としてクラスタリングや主成分分析に時間を割くことはできない．以上をまとめると，Top-k Joinに対する処理の戦略としては，リスト同士のクラスタリングを行うこと，各クラスタごとに非同期に処理を進めること，の2つになる．

4.2.1 CVNL

クラスタリングを用いる手法において，クラスタ数は性能を決定する重要な要因の1つである．VNLでは，各クラスタにおいて全ての可能な組み合わせを生成するため，クラスタ数が少なくなり，各クラスタ内の回答リストが増えると，解に到達する前の処理の重みが増してしまう．この処理には解候補の得点と，オブジェクトの組み合わせが取り得る最高得点との比較による終了判断がないため，必ずかかってしまうコストとして計上される．求める解の数kは，1度にユーザが必要とする結果の数であるため，kは大きくならないことが考えられる．そのため，今回はクラスタ数を増やし，早い段階で解に到達することで，枝刈りの効果を狙う．したがって，クラスタとしてペアもしくは3つ組を生成し，クラスタ内の要素数を最小にする．つまり，リスト数が偶数のときは$m/2$個のリストのペアを作成し，リスト数が奇数のときは3つ組を1つとり，残りをリストのペアとする．クラスタ数の最大は，サブクエリの数をmとして，$m' = \lfloor m/2 \rfloor$となる．

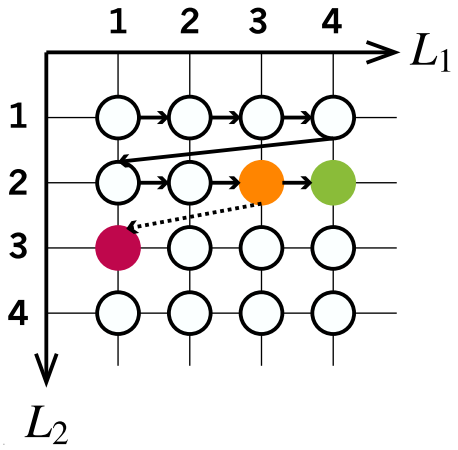

図 4.1: 2 つのリストの場合での例．横軸，縦軸がそれぞれサブクエリの回答リストを表し，格子点がオブジェクトの組み合わせを表す．現在注目している橙色の点が表す組み合わせが取り得る最大の得点と，解候補中の最も小さい得点と比較する．前者が大きい場合は次の L_2 のオブジェクト (図中緑色) を調べる．後者が大きい場合は，L_1 のオブジェクトを赤紫色に変更し，L_2 のオブジェクトを先頭から調べる．

部分解を，各クラスタにおいて生成される，要素数 m 個未満のオブジェクトの組み合わせであるとする．CVNL(Cluster VNL) では，まず始めに，サブクエリに対する回答リストのクラスタを作成した後，各クラスタについて，VNL を利用して可能な組み合わせを全て生成し，得点の降順に整列した部分解のリストを作成する．各部分解のリストが得られたら，それらについて再度 VNL を行うことで解を求める．VNL については，2 章のアルゴリズム 1 を参照されたい．

CVNL ではリストを図 4.1 のように走査していくため，必ずしも得点の高い順に組み合わせを生成することができない．そのため，解候補の得点と，現在注目しているオブジェクトの組み合わせが取り得る得点を利用した打ち切り判定を行うことが難しい．したがって，各クラスタにおいて全ての組み合わせを生成した後，再度 VNL を行い，解を求める．

VNL では，探索の効率化のために，現在注目しているオブジェクトの組み合わせに関して，先に進めるかどうかの判定を行う (図 4.1)．現在，i 番目のリストの j 番目のオブジェクト o^i_j に注目しているとし，これまでに成立したオブジェクトの組み合わせ g の得点を $score(g)$，$i+1$ 番目以降の回答リスト中のオブジェクト

が持つ最大得点を $\sum_{l=i+1}^{m} score(o^l)$ とすると，以下の式 (4.1) のようになる．

$$score(g_k) > score(g) + \sum_{l=i+1}^{m} score(o^l) \tag{4.1}$$

クラスタを用いる場合についても，式 (4.1) と同様に，オブジェクトを各クラスタからできた部分解とみればよい．各クラスタにおいて成立した部分解のなかで，最大の得点をそれぞれ $M'_1, \ldots, M'_{m'}$ とする．現在，i 番目の部分解リストの j 番目の部分解 a^i_j に注目しているとし，これまでに成立した部分解の組み合わせ g' の得点を $score(g')$，$i+1$ 番目以降の部分解リスト中の最大得点を $\sum_{l=i+1}^{m'} M'_i$ とすると，判定式は式 (4.2) となる．

$$score(g_k) > score(g') + \sum_{l=i+1}^{m'} M'_l \tag{4.2}$$

CVNL の利点は式 (4.1) で行う判定よりもよい判定ができる点である．これまでは，各回答リストにおける最大値を利用して判定を行なっていた．しかし，各回答リストの最大得点を持つオブジェクト同士が Join 可能であるとは限らない．CVNL では，各クラスタにおける部分解の最大値を利用して判定を行う．部分解として成り立つもののなかでの最大値であるため，オブジェクトの最大値以下になる．すなわち，式 (4.2) の第 2 項は，式 (4.1) の第 2 項で求められる値以下となるため，効率的な判定が可能である．

4.2.2 CHNL

CHNL(Cluster HNL) の概要を以下にまとめる (図 4.2).

1. サブクエリの回答リストのクラスタを作成する．
2. 各クラスタについて，1 組以上 Join が成立するまで HNL を行い，成立した組み合わせを得点の高い順に部分解リストに追加する．
3. 部分解リストに対して再度 HNL を行い，解候補となる組み合わせを生成する．
4. 式 (4.3) が成り立つまで 2. と 3. を続ける
5. 式 (4.3) が成り立った場合，クラスタからの生成を打ち切り，部分解リストについて HNL を行う．

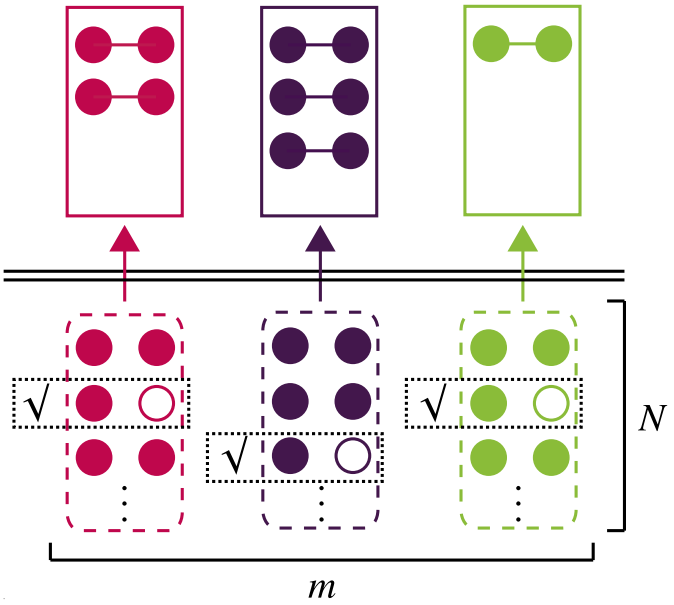

図 4.2: CHNL の計算モデル．回答リストのクラスタ (図下点線角丸枠) において 1 個以上組み合わせができるまで HNL を行い，できた組み合わせを集めて部分解リスト (図上実線枠) を作成する．部分解リストについて再度 HNL を行い，組み合わせを完成させる．各クラスタから作られる全ての組み合わせを作るのではなく，できたものから逐時処理をする．

6. 部分解リストについても式 (4.3) が成り立ったら，探索を終了する．

まずはじめに，サブクエリの回答リストのクラスタを作成する．CHNL においても，CVNL と同様の理由で，クラスタ内の回答リスト数を最小にするため，クラスタとして回答リストのペアか，もしくは 3 つ組を作成する．

次に，各クラスタにおいて HNL を行い，部分解を作成する．各クラスタで部分解を作る理由は，成り立たない組み合わせの早期除去である．HNL では，全ての回答リストを同時に利用し，組み合わせの作成を行う (図 4.3)．そのため，組み合わせの作成順によっては無駄な組み合わせを何度も調べてしまう欠点がある．例えば，図 4.4 のように，どのオブジェクトとも Join 不可であるが，オブジェクト自体の得点が高い場合，何度も成り立たない組み合わせを調べることになってしまう．しかし，クラスタ毎に部分解を作ることで，そのようなオブジェクトを除去できるので，処理を効率化できると考えられる． 打ち切り判定は解候補が k 個集まったときから開始する．2. において，各クラスタにおいて成立した組み合わせのうち最大の得点をそれぞれ $M'_1, \ldots, M'_{\lfloor m/2 \rfloor}$ とし，また，これまでに各クラスタで得られた最大得点を $M_1, \ldots, M_{\lfloor m/2 \rfloor}$ とする．ここで，各クラスタにおいて成立した

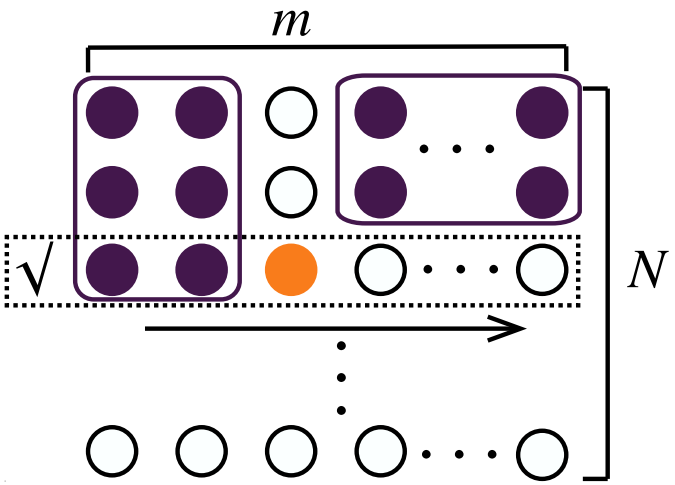

図 4.3: HNL の計算モデル．橙色円で表す現在注目しているオブジェクトと，それ以前に出現した紫色円で表すオブジェクトとの全ての組み合わせを考える．オブジェクトは得点順に整列されているので，得点の大きいものから順に組み合わせが生成される．

部分解のうち，最大の得点のものとしたのは，2. においては必ずしも得点の大きな順に 1 つずつ生成されるとは限らないためである．解候補の中で最も得点の小さい，k 番目の組み合わせ g_k の持つ得点を $score(g_k)$ としたときに，以下の式 (4.3) が成り立てば探索を打ち切ることができる．全てに対して以下の式 (4.3) が成り立たなくとも，一部成立したクラスタに関しては以後 2. で示される処理を行わない．

$$\forall 1 \leq i \leq \lfloor m/2 \rfloor, i \in \mathbf{N} : score(g_k) > \sum_{j \neq i} M_j + M'_i \tag{4.3}$$

CHNL の利点は 2 つある．1 つは，CVNL のときと同様に，処理の終了判定を行う条件がより効率良くなっている点である．式 (4.3) の第 2 項では，成立した部分解の得点の最大値を利用しているため，各回答リストにおける得点の最大値を利用するよりも小さくなるからである．もう 1 つは，クラスタを行うことで成り立たない組み合わせを早期に枝刈りできる点である．

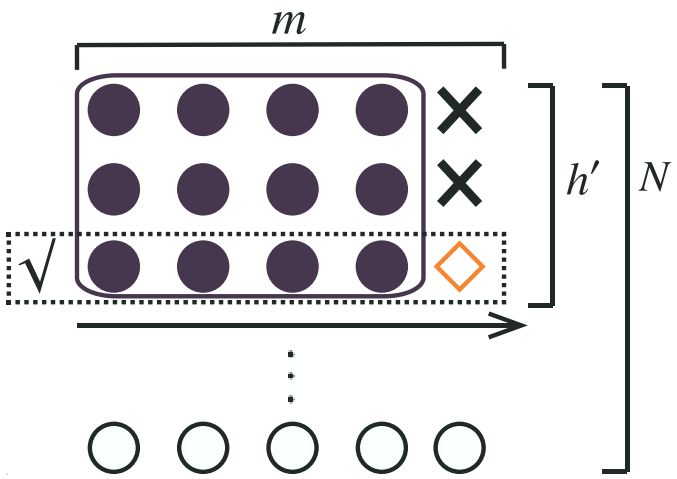

図 4.4: HNL がうまくいかない場合の一例．図中 × 印をどのオブジェクトとも Join 不可能なオブジェクトであるとし，図中橙色の ◇ 印をどのオブジェクトとも Join 可能なオブジェクトとする．HNL では，◇ にたどり着くまでに h'^m 個のオブジェクトを調べることとなり，無駄なコストがかかってしまう

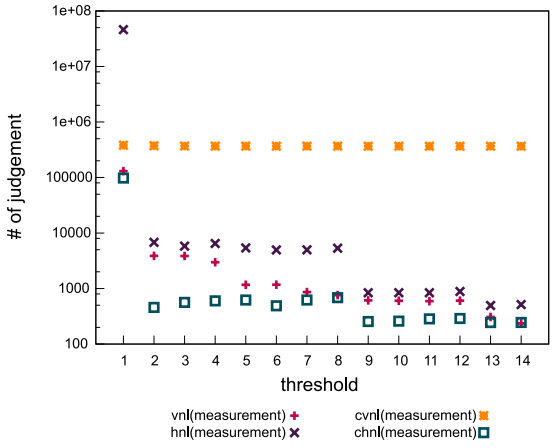

図 4.5: CMU Face Images(k=10)

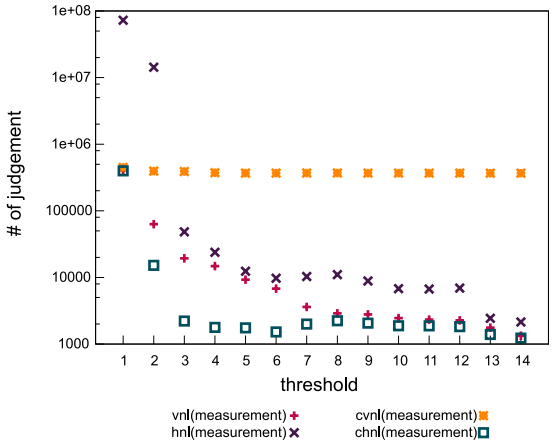

図 4.6: CMU Face Images(k=50)

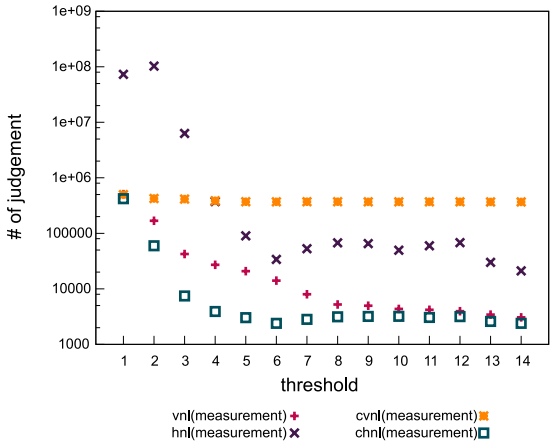

図 4.7: CMU Face Images(k=100)

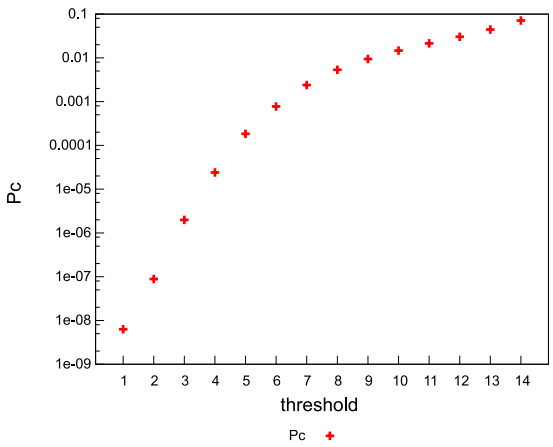

図 4.8: 横軸の閾値に対応する P_c

4.3 性能評価

実験は，Intel(R) Xeon(R) X5492 3.2GHz プロセッサと 64GB の RAM を搭載した計算機上で行った．アルゴリズムの実装は C 言語で行い，GNU/Linux Debian 6.0 上の gcc 4.4.5 でコンパイルを行った．

実験には，CMU Face Images[21] を用いて生成したデータを利用した．結果を図 4.5 から図 4.7 に，横軸の閾値に対応した P_c を図 4.8 に示す．P_c は，以下に再掲する式 (2.5) で求める．式 (2.5) 中の $PreAns$(Q) はサブクエリの回答リストの集合を表す．

$$P_c = \frac{Join\text{ が成功する組み合わせの数}}{PreAns(\mathrm{Q})\text{ から 1 つずつオブジェクトを選んでできる全組み合わせ数}} \quad ((2.5)：再掲)$$

実験シナリオとして，“複数の異なるカメラに写る人物のうち，どのカメラにおいても共通して挙動が特異な人物はどれか”，を探す検索を想定した．挙動が特異な人物を異常値とみなし，Lakhina らによって提案された手法を利用して異常値を求める [12]．そして，異常かどうかを判断する値を順位付けのための得点として用い，検索を行う．データ生成の概略を以下に示す．

1. 各画像について，GIST 特徴量を求める [15]．抽出には Matthijs らによる実装を利用した [6]．

2. 特徴量を抽出したら，主成分分析を行い，固有ベクトルを求める．今回の実験では，求めた固有ベクトルのうち，第 1 主成分から第 8 主成分を利用した．

3. 求めた固有ベクトルからできる行列 $\mathbf{U}$ から，residual subspace への写像 $\mathbf{P} = \mathbf{I} - \mathbf{U}\mathbf{U}^T$ を求める．$\mathbf{x}$ を 1. で求めた特徴ベクトルとし，先ほど求めた $\mathbf{P}$ を利用して，residual vector $\mathbf{z} = \mathbf{P}\mathbf{x}$ を求める．

4. 自乗予測誤差 $\mathrm{SPE} = \|\mathbf{z}\|^2$ を求め，順位付けの得点とする．ここで，$\|\mathbf{z}\|$ はベクトル $\mathbf{z}$ の 2 乗ノルムを表す．

2 つのオブジェクトが Join 可能であるかどうかは，画像間の類似度を利用した．類似度の判定には Euclid 距離を利用し，2 つの画像間の距離がある一定以下の場合，Join 可能であると判断する．図中の横軸が画像間の距離の閾値を表す．小さいほど類似度が高いもの同士でのみ Join 可能であるため，P_c が小さくなり，大きくなるほど類似度が小さくても Join 可能であると判断されるため，P_c が大きくなる．

提案手法のうち，CVNL は解を求める前の処理，すなわち各クラスタにおいて全てのペアを求める段階において大きくコストがかかっているため，他の 3 手法に大きく水を開けられている．また，図 4.8 からわかるとおり，P_c が非常に低いため，VNL が HNL よりも効率よく処理できる．しかし，このような状況においても，提案手法である CHNL は，最大で HNL の 1/100，VNL の 1/10 の判断演算回数を達成した．これは，図 4.2，図 4.3 に示すように，HNL では成り立たない組み合わせを毎回調べなければならない可能性があるが，CHNL では．組み合わせの生成に，各クラスタごとに成立した組み合わせのみを利用するためであると考えられる (図 4.4)．

しかし，条件によっては既存手法のほうが有効であることが図 4.5 から図 4.7 よりわかる．処理効率の最善を目指すためには，その時々の状況に合わせた手法の選択をしなければならないが，選択のためには判断基準が必要となる．そこで，次

節では，VNL，HNL，CVNL，CHNL の各手法について，いくつか仮定をおいた場合における性能予測を行う．

表 4.2: パラメータ設定

パラメータ	値
各リストのオブジェクト数	N
リスト数	m(ただし m は偶数)
バイナリ Join の成功確率	P_b
(CVNL，CHNL のみ) クラスタ数	$m/2$(1 つのクラスタに 2 個のリスト)

4.4 性能予測

本節では，最適な手法選択のための計算コスト予測のための判断基準導出を行う．前節において，どのような場合においても提案手法が有効なのではなく，条件によって手法選択の余地があることがわかった．そこで，手法選択の判断基準を作成すべく，いくつか仮定をおいた場合について計算コストの予測を試みる．ここでは，計算コストを，処理の最小単位である，判断演算回数として近似する．

4.4.1 処理における仮定

検討には，以下の 3 つの仮定を置く．各リストは同数のオブジェクトを持つこととし，バイナリ Join の成功確率は独立でかつ全て等しいとした．

仮定 (1) 各リストはそれぞれ N 個のオブジェクトを持つ．

仮定 (2) バイナリ Join の成功確率 P_b はどのペアをとっても等しく，かつ完全に独立であるとする．そのため，$P_c = (P_b)^{{}_mC_2}$ が成立する．

仮定 (3) バイナリ Join の成功確率 P_b はどのペアをとっても等しいため，Join を $L_1, L_2, \ldots, L_m$ の順で進めて一般性を失わない．

検討のためのパラメータを表 4.2 にまとめる．

4.4.2 VNL におけるコスト予測

VNL について，2 つのオブジェクトが Join 可能かどうかの判断演算回数の期待値 $E_{\mathrm{VNL}}(N, m, P_b)$ を求める．演算回数の期待値は，組み合わせが成立する数の期待値に対して，1 つの組み合わせあたりに必要な演算回数の期待値を乗算すればよい．

まずは，m 本のリストからできる組み合わせの数の期待値 $f(m)$ を求める．オブジェクトをそれぞれ N 個持った m 本のリストにおいて，生成された組み合わせの数を $f(m)$ とする．$f(m)$ に N 個のオブジェクトを持った，$m+1$ 本目のリストを追加した場合を考える．m 個の要素を持つ，1 つの組み合わせに対して，1 個のオブジェクトを追加するとき，追加するオブジェクトが，組み合わせ中のオブジェクト全てに対してバイナリ Join が成立する必要があるため，確率 P_b^m で組み合わせが成立する．リスト中にオブジェクトが N 個あるため，リストを 1 個追加した場合に生成される組み合わせの数は，$m \geq 2$ のとき，以下の式 (4.4) により表される．ただし，$m = 2$ のとき，$f(2) = N^2 P_b$ である．

$$f(m+1) = f(m) N P_b^m \tag{4.4}$$

続いて，1 つの組み合わせに関して演算を行う回数の期待値を求める．m 個の組み合わせの場合，判断演算回数の期待値 β は以下の式 (4.5) により表される．

$$\begin{aligned} \beta(m) &= \sum_{l=1}^{m-1} \left(P_b^{l-1} (1 - P_b) l \right) + m P_b^{m-1} \\ &= \sum_{l=1}^{m-1} P_b^{l-1} \end{aligned} \tag{4.5}$$

$m = 2$ のときは特別な場合で，2 本のリストからできる組み合わせを全て調べる必要があるため，N^2 となる．$m \geq 3$ のときの判断演算回数の期待値は，$f(m-1) N \beta(m-1)$ となる．

以上より，$E_{\mathrm{VNL}}(N, m, P_b)$ は，以下の式 (4.6) ように表される．

$$E_{\mathrm{VNL}}(N, m, P_b) = N^2 + \sum_{i=2}^{m-1} N f(i) \beta(i) \tag{4.6}$$

4.4.3 CVNL の場合

CVNL について，判断演算回数の期待値 $E_{\mathrm{CVNL}}(N, m, P_b)$ を求める．CVNL では，始めにサブクエリの回答リストをクラスタに分割し，各クラスタそれぞれにおいて VNL を利用して部分解を求める．そして，各クラスタから得られた部分解を再度 VNL により組み合わせて，最終的な結果とする手法である．表 4.2 に示すとおり，クラスタ数は $m/2$ としているので，各クラスタはそれぞれ 2 個のリストを要素として持つ．

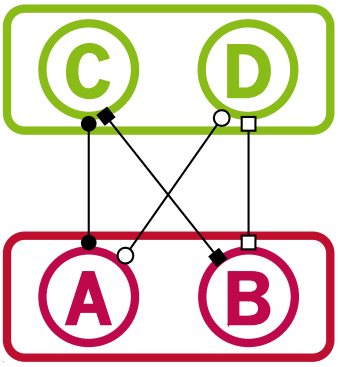

図 4.9: 2 つのペアを 1 回 Join する場合，判定演算は A-C，A-D，B-C，B-D の 4 回必要である．

まずは，各クラスタにおける判断演算回数を求める．それぞれのクラスタは，リストを 2 個要素に持つので，少なくとも $mN^2/2$ 回演算が必要である．各クラスタから部分解が N^2P_b 個得られるので，あとは，リスト数 $m/2$ 個，リスト中の各オブジェクト数 N^2P_b 個の VNL とみなせばよい．ただし，要素数 2 の部分解同士の組み合わせを考えるため，リスト数の 4 倍演算を行わなければならない (図 4.9)．以上より，m' 個のクラスタから得られる組み合わせの数の期待値 $f'(m')$ は，以下の式 (4.7) の関係を持つ．ただし，$f'(2) = N^4P_b^2$ である．同様にして，ペアの組み合わせ 1 組における演算回数の期待値 β' は，ペア数は ${}_{m'}C_2$ 個あるので，以下の式 (4.8) となる．

$$f'(m'+1) = f'(m')N^2P_bP_b^{4m'} \tag{4.7}$$

$$\beta'(m') = \sum_{l=1}^{{}_{4m'}C_2} P_b^{l-1} \tag{4.8}$$

以上より，$E_{\mathrm{CVNL}}(N, m, P_b)$ は，$m \geq 4$ かつ m が偶数のとき，以下の式 (4.9) のように表すことができる．

$$E_{\mathrm{CVNL}}(N, m, P_b) = \frac{mN^2}{2} + N^4P_b^2(1 + P_b + P_b^2 + P_b^3) + \sum_{i=2}^{m/2-1} N^2P_bf'(i)\beta'(i) \tag{4.9}$$

4.4.4 HNL の場合

HNL についてバイナリ Join が成立するかどうかの判定演算回数の期待値 $E_{\mathrm{HNL}}(N, m, P_b)$ を求める．HNL は図 4.3 のように，リストの先頭から順に組み合わせを生成する．詳しくは，2 章のアルゴリズム 2 を参照されたい．

HNL も VNL と同様，できる組み合わせの数に対して，1 組の組み合わせに必要な演算回数の期待値を乗算すればよい．N 行目までに得られる全ての組み合わせの数は N^m 個であるため，演算回数の期待値は，以下の式 (4.10) となる．

$$E_{\mathrm{HNL}}(N, m, P_b) = N^m \beta(m) \tag{4.10}$$

4.4.5 CHNL の場合

CHNL について，判定演算回数の期待値 $E_{\mathrm{CHNL}}(N, m, P_b)$ を求める．CHNL の動作をまとめると，以下のようになる．まずは各クラスタにおいて HNL を行い，クラスタから得られた組み合わせについて再度 HNL を行う．各クラスタにおいては，CVNL のようにクラスタからできる部分解を全て生成するのではなく，逐時部分解を収集し，解候補を生成する (図 4.2).

各クラスタにおいて，部分解を作成するのに必要な演算回数はそれぞれ N^2 回である．各クラスタからオブジェクトのペアは $N^2 P_b$ 個できるため，$N^2 P_b$ 個のオブジェクトを持ったリストが $m/2$ 個あると考えると，式 (4.10) より，CHNL の演算回数期待値は以下の式 (4.11) となる．

$$E_{\mathrm{CHNL}}(N, m, P_b) = \frac{mN^2}{2} + (N^2 P_b)^{m/2} \beta'\left(\frac{m}{2}\right) \tag{4.11}$$

4.5 コストの予測値と実測値の比較

4.5.1 実験設定

本節では，前節で導出したコスト予測式をもとに，実測値との比較を行う．比較は，人工データと先ほどの CMU Face Images を利用する．

人工データの特徴を以下の表 4.3 にまとめる．人工データでは，バイナリ Join がどのペアで成り立つかを，毎回乱数を用いて変化させて実験を行う．各 P_b についてそれぞれ 100 回計測を行い，平均回数を求めて結果とした．

コストの下限値の予測は，前節で求めた式 (4.6)，式 (4.9)，式 (4.10)，式 (4.11) 中の N に，以下に再掲する式 (2.6) で求められる h を代入して行う [16]．この h は，P_c が与えられた場合に，リストの先頭からいくつ調べれば，解を k 個得られることが期待できるかを表す指標である．今回，$k = 10, 50, 100$ を代入して得られる値を利用するため，最初に得られた組み合わせが，そのまま解になる場合についての期待値を算出することとなる．したがって，これらの式に h を代入して得られる期待値は，$k = 10, 50, 100$ のそれぞれの場合における計算コスト下限の期待値を表す．

$$h = \sqrt[m]{\frac{k}{P_c}} \tag{2.6:再掲}$$

4.5.2 理想的な場合における比較

図 4.10 から図 4.21 に人工データにおける判断演算回数の実測値平均，下限値の期待値を示す．図より，どの手法においても期待値と実測値の傾向は一致しているため，処理コストの傾向を定性的にではあるが見積もることが可能である．

表 4.3: 人工データのパラメータ

パラメータ	値
求める解の数 k	10,50,100
各リスト中のオブジェクト数 N	100
リスト数 m	4
P_b	$0.1, 0.2, 0.3, \ldots, 0.9$
オブジェクトの得点	1 番目は 100，2 番目は 99，…，100 番目は 1．

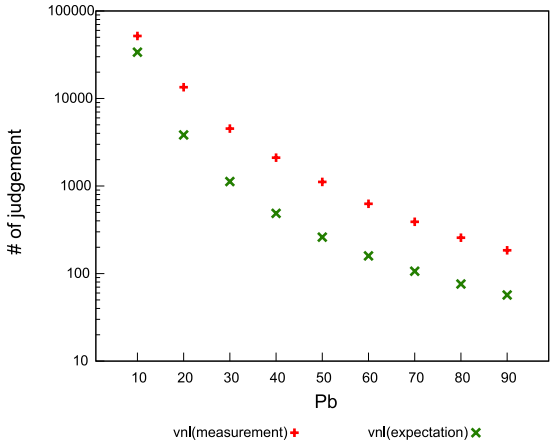

図 4.10: VNL での実測値と期待値の比較 (k=10)

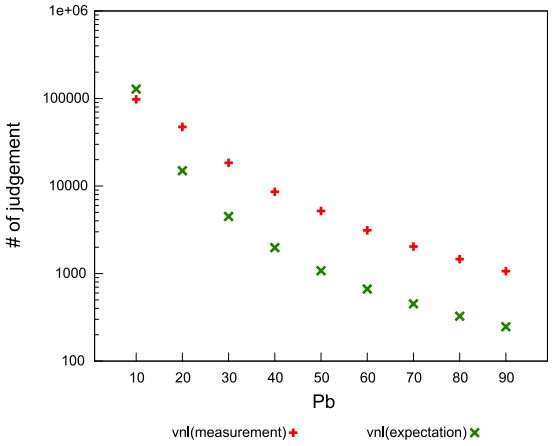

図 4.11: VNL での実測値と期待値の比較 (k=50)

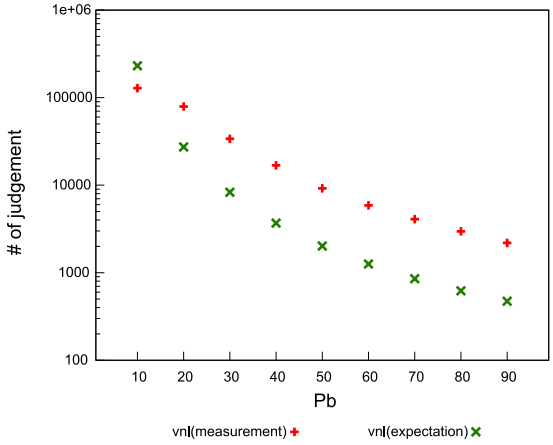

図 4.12: VNL での実測値と期待値の比較 (k=100)

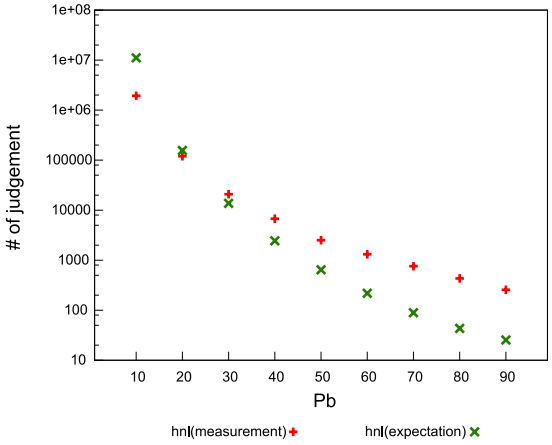

図 4.13: HNL での実測値と期待値の比較 (k=10)

バイナリ Join の成立する確率 P_b が低い場合，CVNL，HNL において，実測値と期待値の差が大きくなっている．この原因は，予測式自体が枝刈りの効果をうまく考慮できていない点，また，P_c を利用して求めた h を代入する近似にあると考えられる．予測式は，各回答リストに対して，N 個全てを利用して組み合わせを作る場合について，判断演算回数を求める．そのため，途中で打ち切り判定が成立した場合，ずれが生じてしまう．これは，P_c は式 (2.5) より，できうる組み合わせ全体に対して，成立する組み合わせの割合として求められるため，P_c の局所性，すなわちどこで解候補が成立しやすいか，を考慮できない点に問題がある．P_c の局所性は，必要な解の数 k からも影響を受ける．k が大きくなった場合，成立する組み合わせの数に対して，必要な解候補の数が増えるため，局所性が問題になり

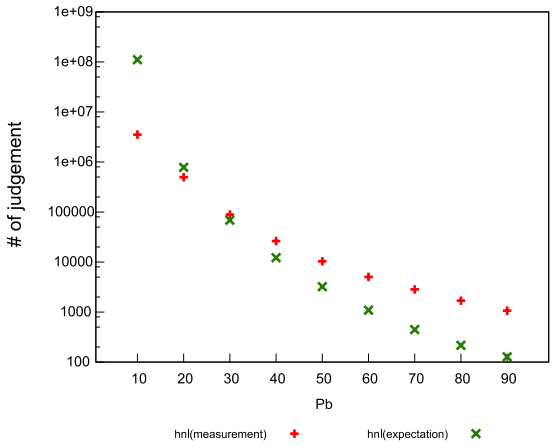

図 4.14: HNL での実測値と期待値の比較 (k=50)

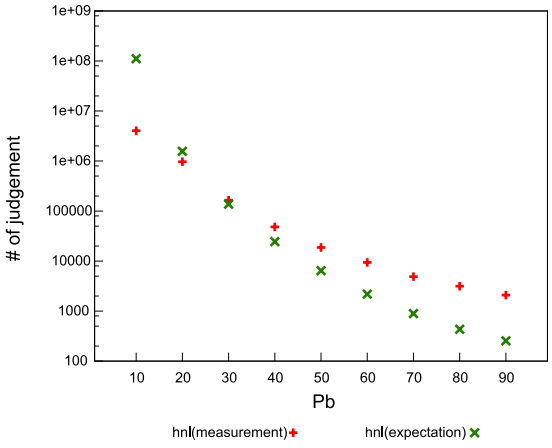

図 4.15: HNL での実測値と期待値の比較 (k=100)

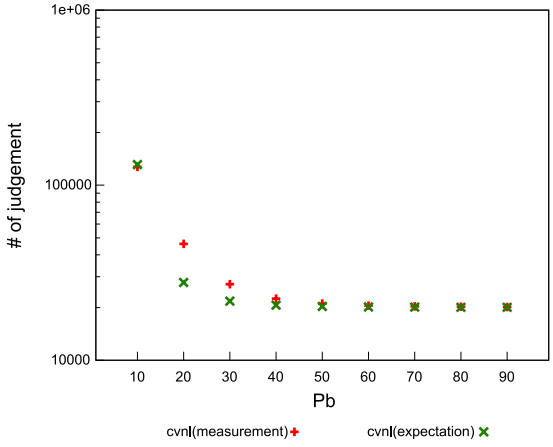

図 4.16: CVNL での実測値と期待値の比較 (k=10)

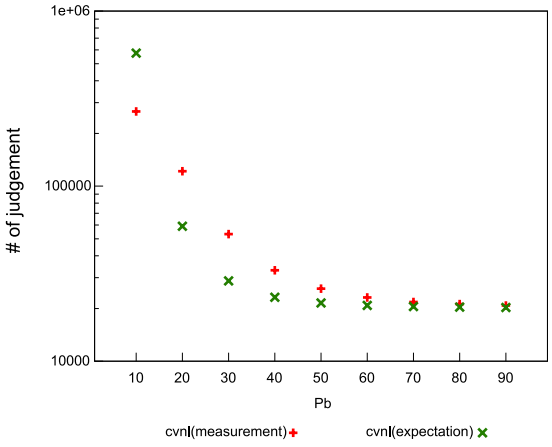

図 4.17: CVNL での実測値と期待値の比較 (k=50)

にくい．以上から，予測が正確になる条件は，P_c を定数ではなく，分布として表すこと，そして必要な解の数 k を大きくとることであると考えられる．

特に HNL について，P_b が増加するほど乖離が大きくなるのは，下限値の期待値の求め方によるものである．今回，下限値を見積もるために，k 個の組み合わせを得ることが期待できる h を求めた．先程も述べた通り，h を利用して求めた予測値は，あくまで下限値の期待値である．つまり，k 個得られた時点で上位 k 件が確定し，それ以降は処理の必要がないと判定される場合である．実際には，k 個以上解候補を求めているため，解候補が k 個得られた時点で探索を終了するとした予測値よりも実測値が大きくなった．

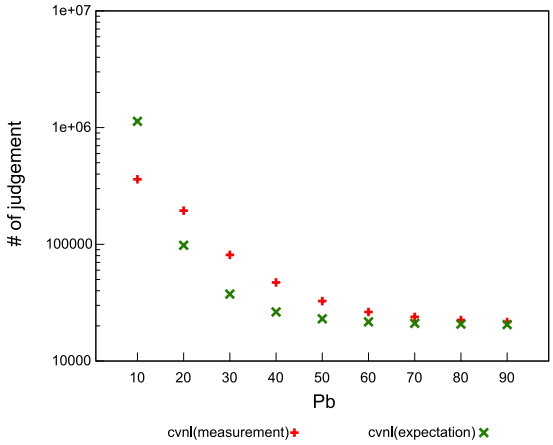

図 4.18: CVNL での実測値と期待値の比較 (k=100)

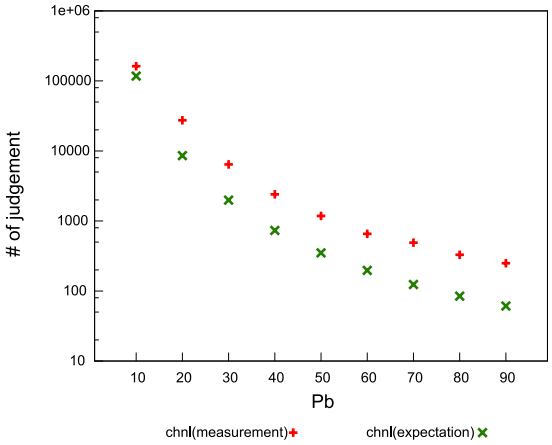

図 4.19: CHNL での実測値と期待値の比較 (k=10)

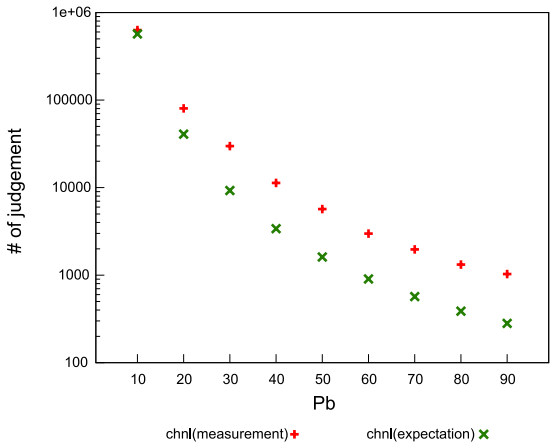

図 4.20: CHNL での実測値と期待値の比較 (k=50)

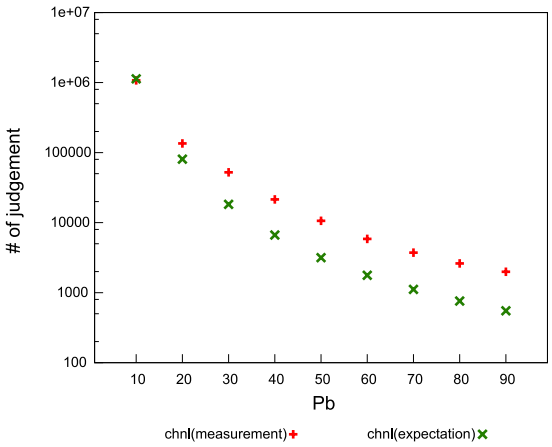

図 4.21: CHNL での実測値と期待値の比較 (k=100)

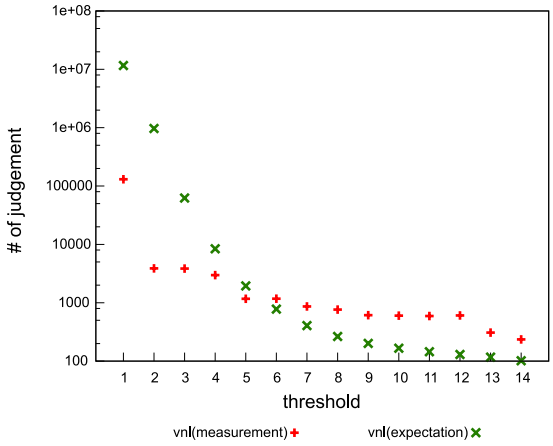

図 4.22: VNL での実測値と期待値の比較 (k=10)

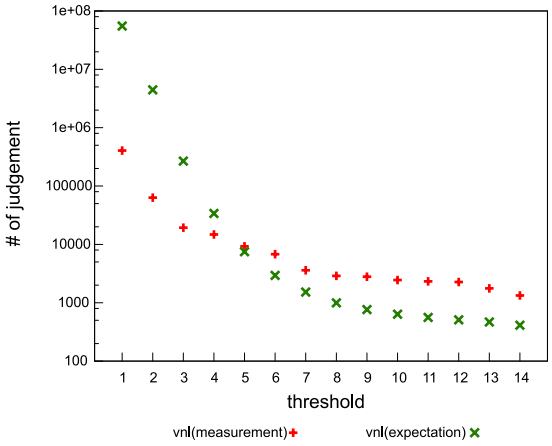

図 4.23: VNL での実測値と期待値の比較 (k=50)

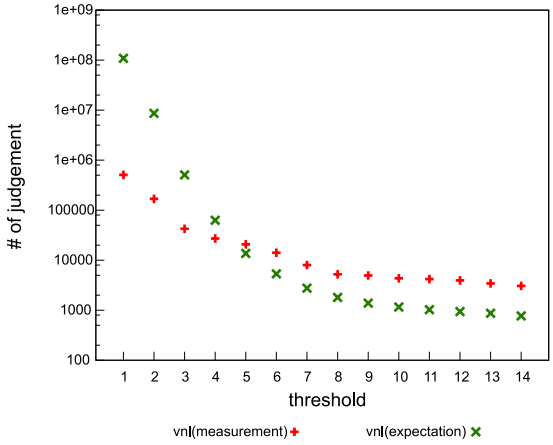

図 4.24: VNL での実測値と期待値の比較 (k=100)

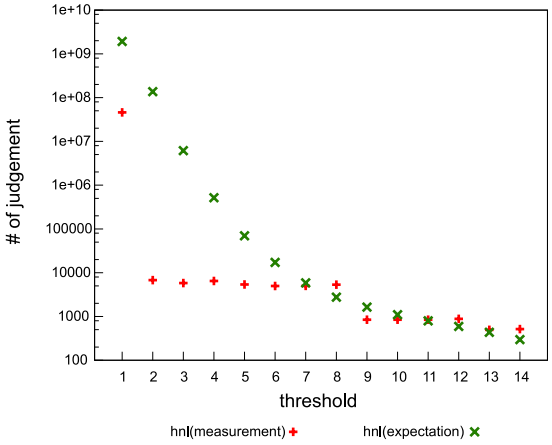

図 4.25: HNL での実測値と期待値の比較 (k=10)

4.5.3 実データにおける比較

図 4.22 から図 4.33 に CMU Face Images を利用した場合の判定演算回数の実測値平均，下限値の期待値を示す．図より，画像の同一判定を行う閾値の小さい部分，すなわち P_c が小さい部分では，予測と実測が大きく離れている．しかし，閾値が大きくなってくるにつれて，似た傾向を示すようになった．また，k の増加にともなって，予測値と似た傾向を示すことがわかる．よって，P_c と k の増加にしたがい，実測値との差が小さくなり，傾向を予測可能である．

しかし，P_c が小さいところでは，予測値と実測値の乖離が大きい．これは，人工データの場合と同じ理由であると考えられる．P_c が小さい領域では，h が大きく

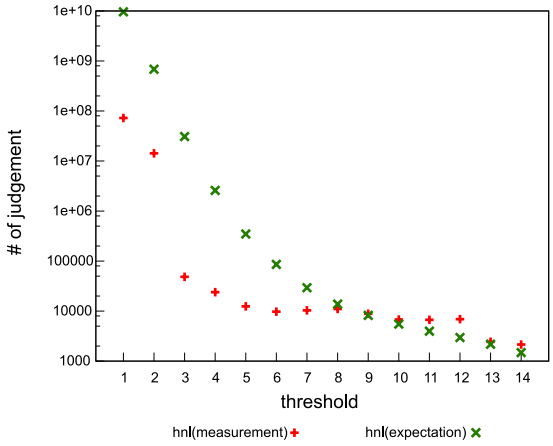

図 4.26: HNL での実測値と期待値の比較 (k=50)

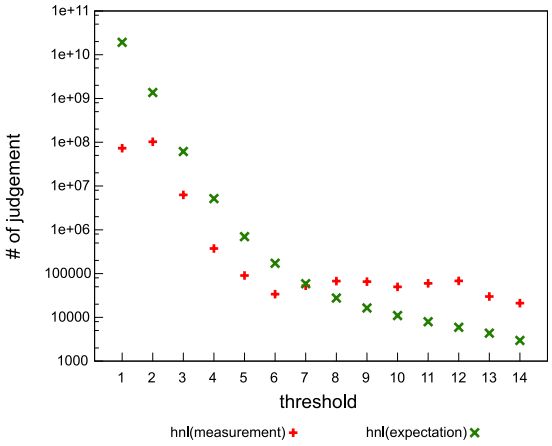

図 4.27: HNL での実測値と期待値の比較 (k=100)

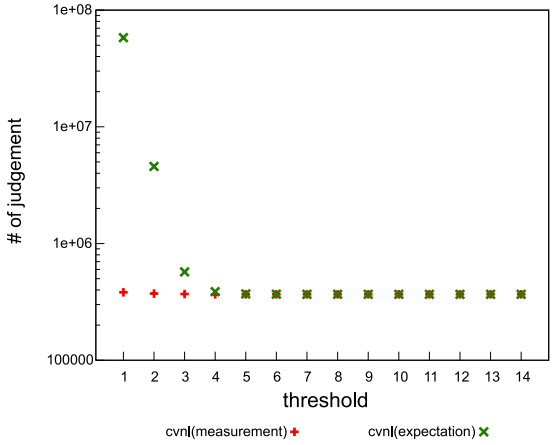

図 4.28: CVNL での実測値と期待値の比較 (k=10)

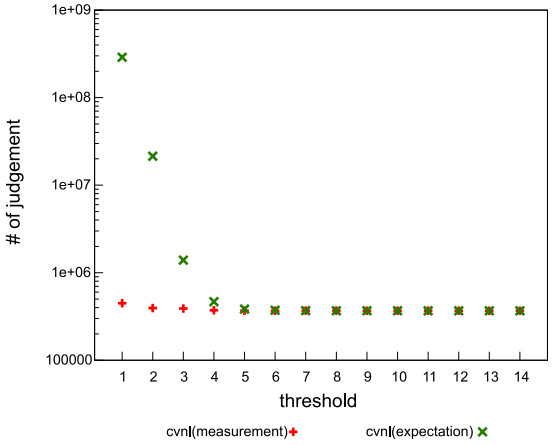

図 4.29: CVNL での実測値と期待値の比較 (k=50)

なり，生成される組み合わせの数が増えるため，どこで解候補が得られるか，と言った確率の局所性が重要になってくると考えられる．しかし，P_c が大きい領域では，h が小さくなり，生成される組み合わせの数が減るため，この局所性の問題は現れにくくなる．これは，k が増加したときも同様で，生成される組み合わせに対して，必要な解候補数が大きくなるため，予測値と実測値の差が小さくなったと考えられる．

予測値と実測値の誤差の要因は，P_c を一定とみなすことや，k だけでなく，P_b を独立かつ一定と仮定した点にあると考えられる．各リスト間の P_b は，表 4.4 に示すとおり偏りがあるため，バイナリ Join が成立する確率は一定とした仮定が成立しない．また，一般に，Join の成り立つもの同士は何らかの関係性を暗に持っ

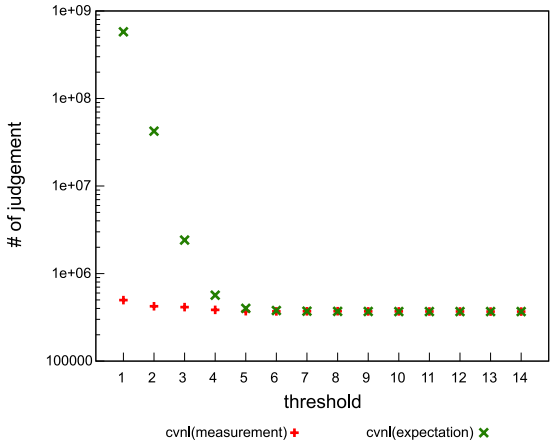

図 4.30: CVNL での実測値と期待値の比較 (k=100)

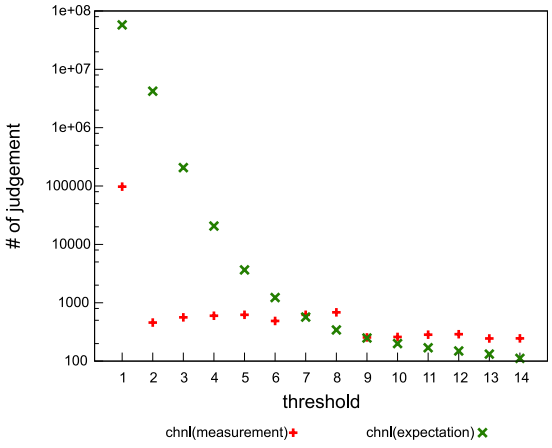

図 4.31: CHNL での実測値と期待値の比較 (k=10)

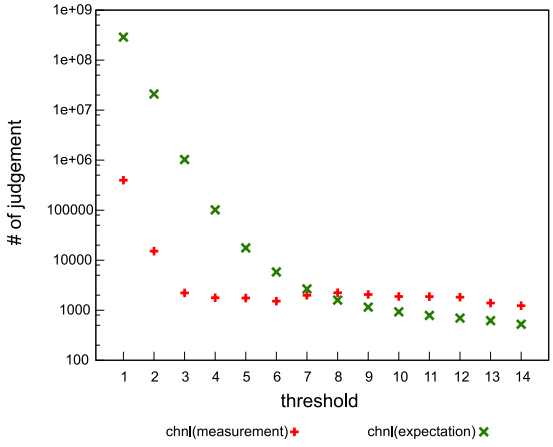

図 4.32: CHNL での実測値と期待値の比較 (k=50)

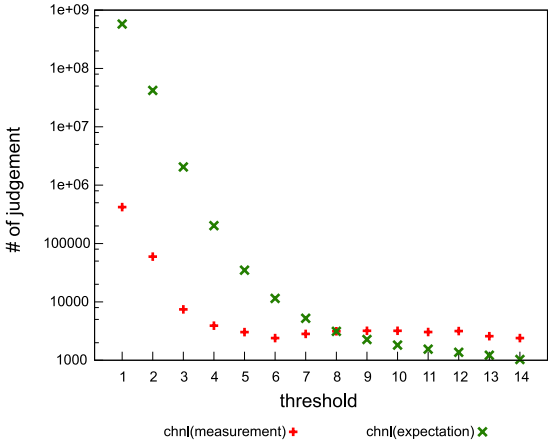

図 4.33: CHNL での実測値と期待値の比較 (k=100)

ていると考えられるので，確率的に完全に独立ではなく，従属であると考えられる．以上が原因となり，予測値と実測値にずれが生じた．

4.5.4 本節のまとめ

本節では，VNL，HNL，CVNL，CHNL の各手法において，計算モデルを仮定し，2 つのオブジェクトが Join 可能であるかどうかの判断演算回数の予測を行った．判断演算は最も多く行われる演算であるため，判断演算回数が予測できれば，計算コストの予測が可能である．

人工データと実データを用いた実験を行い，予測値と実測値の比較を行った．実

表 4.4: CMU Face Image における，各リスト同士の P_b(一部抜粋)

閾値 ＼ リストの組み合わせ	1-2	1-3	1-4	2-3	2-4	3-4
1	0.26	0.17	0.13	0.17	0.15	0.17
7	0.37	0.30	0.26	0.29	0.27	0.29
14	0.62	0.58	0.56	0.58	0.57	0.58

験結果より，予測と実測の傾向が合うのは，組み合わせが成立する確率 P_c が大きいとき，求める解の数 k が大きいときであることが明らかになった．

しかし，P_c や k が小さいとき，実測値と予測値は大きく乖離してしまう．これの原因の 1 つに，予測に用いたパラメータ h が挙げられる．パラメータ h は，各リストにおいて先頭からいくつオブジェクトを調べれば，解候補を k 個得ることが期待できるか，を表す値である．P_c が小さくなると，h が大きくなるため，調べる組み合わせの数が増加する．そのため，実際にどこで解候補が得られるのか，という局所性が重要になるが，P_c は組み合わせ全体から求める値であり，局所性を考慮できない点に問題がある．k が大きくなると，得られる解候補に対して，必要な解候補の割合が増えるため，局所性の問題が現れにくい．

予測式は，2 つのオブジェクトに関して Join が成立する確率 P_b を一定かつ独立としている．一般に，P_b は偏りがあり，また，Join が成立するオブジェクト同士で何らかの関係性があると考えられるため，確率的に独立ではなく従属である．

以上を踏まえて，より正確な予測をするためには，P_c を定数ではなく分布で扱うこと，また，確率的に従属である場合を考慮したモデルを立てることが必要である．

4.6 本章のまとめ

本章では，Top-k Join について，クラスタリングを利用した高速化手法を提案した．これまでの Top-k Join のように，全ての組み合わせが成立するのではなく，異なる複数のデータベースに対する検索結果を組み合わせることを目的としているので，Join が成立する場合としない場合がある．したがって，これまでの Top-k Join の汎用アルゴリズムをそのまま適用することは難しい．そこで，本手法では，Shalem らが提案した手法のように，各データベースの検索結果のリスト全てに対して同時に処理を行うのではなく，各リストをクラスタに分割し，クラスタごとに処理することで高速化を行った．CVNL では，各クラスタで成立する部分解全てを求め，それらを組み合わせることで解を求める．CHNL では，各クラスタについて部分解を 1 つ以上生成するまで HNL を行い，生成したら逐時 Join を行い，解を求める．実データを用いた実験から，CVNL では各クラスタにおける処理のコストが大きく，他の手法に大きく差をつけられてしまうが，CHNL では，既存手法と比較して 1/10 ほどコストを減らすことができた．

提案手法は，条件によっては既存手法のほうが高速に処理できる場合もあるため，システムの高速化を目指すのであれば状況に合わせて適切な手法を選択する必要がある．選択を行うには計算コストの見積もりが必須であるので，コスト予測式の提案した．この予測式では，求める解の数 k，組み合わせが成立する確率 P_c が大きい場合，うまく傾向を説明できる．しかし，P_c や k が小さい場合，また，2 つのオブジェクトが Join 可能である確率 P_b が全体で一定でない場合，実測値とのずれが大きくなってしまう．そのため，より精度よく計算コストを予測するためには，P_c が全体で一定ではないこと，P_b が全体で一定かつ独立ではないことを考慮したモデルが必要である．

第 5 章
結論

5.1 本論文のまとめ

本論文では，検索の質の高度化を目指し，異なる 2 つの種類の Top-k 組み合わせ検索の処理速度改善について検討を行った．まずはじめに，クエリとの類似度評価に Euclid 距離を用いた，ベクトル空間における Top-k 組み合わせ検索アルゴリズムを提案した．提案手法は，事前にデータに対してクラスタリングを適用する．クエリが与えられたら，まずはクラスタの組み合わせを求めることで解を絞り込み，高速化を行う．2 種類の異なるデータに対して実験を行い，最大で 20 倍程度の性能を向上させることを示した，次に，クラスタリングを利用する高速化手法を，全てのオブジェクトが Join 可能でない場合の Top-k Join に適用し，有効性を示した．また，システムの最適化に必要な処理コストの予測式の提案を行い，人工データ，実データの両方において大域的な挙動を説明できることを示した．

最後に，本論文の貢献を以下にまとめる．

1. クラスタリングを利用した Top-k 組み合わせ検索のアルゴリズムを提案し，手法の有効性を示した．

2. クラスタリングを利用した高速化手法を Top-k Join へ応用し，手法の有効性を示した．

3. 検索システムの最適化に必要な，処理コスト予測式を提案し，実データと人工データにおいて大域的な挙動を説明できることを示した．

5.2 今後の課題

今後の課題としては，以下の 3 つが挙げられる．

1. アプリケーションに即したデータで実験を行い，状況に応じた手法の修正と改善を行う．
2. コスト予測式の精度を高める．
3. 導出した式を利用してコスト予測を行い，クエリの最適化を行う．

本論文では，基礎的な検討に重点をおいているため，実データに対する評価が乏しい．したがって，今後はアプリケーションに特化した手法を検討し，より現実に近い高速化手法および計算コスト予測を行うことが必要である．具体的には，データの偏りを利用した枝刈り手法や，パラメタを自動で設定するクラスタリング手法，より精度の高い処理コスト予測式の提案である．

謝辞

本研究は，多くの方々の暖かいご支援により，進めることが出来ました．ここに，篤く御礼申し上げます．

指導教官である安達淳教授には，日々暖かい指導をいただきました．枚挙に暇がありませんが，何より，のびのびと自由に研究活動をやらせていただいたことに深く感謝しています．また，シンポジウムや学会参加といった貴重な機会を惜しみなく与えていただき，興味と知識の幅を拡げることが出来ました．ありがとうございました．

高須淳宏教授には，全体ミーティングだけでなく，個別ミーティングにいたるまで，何度も丁寧にご指導いただきました．高須教授の励ましのおかげで，ここまでめげずに続けられました．海外出張の際には，先生のご引率のおかげで，安心して発表することができました．深く感謝いたします．

浅見徹教授，川原圭博講師には，研究室を変わってからも大変お世話になりました．研究だけでなく，楽しいイベントに誘っていただいたり，就職活動でアドバイスをいただくなど，本当に感謝しています．また，浅見・川原研究室の皆様にも大変お世話になりました．ありがとうございました．

安達研究室 OB の倉沢央さんには研究室生活において大変お世話になりました．データ処理の基礎知識の指導や研究に関する相談ももちろんのこと，昼食のおいしいお店に連れていっていただいたり，おかげで楽しく研究室生活を送ることができました．ありがとうございました．

安達研究室の博士課程 1 年の木村光樹さんには研究活動において大変お世話になりました．研究へのアドバイスや指導だけでなく，暖かい励ましの言葉をいただいたり，飲み会に連れて行っていただいたおかげで何とか続けてくることができました．深く感謝いたします．

安達研究室 OB の渡辺健太郎さん，安達研究室のメンバーである博士課程 3 年の Chu Yimin さん，修士課程 1 年の山本敬介君，赤塚裕人君，同期の那小川君，お世話になりました．皆様のおかげで，研究室の雰囲気がよくなり，落ち着いて研究をすすめることができました．ありがとうございました．

事務の久芳さん，上野さんには，書籍や備品などの必要な資材の購入から，出張の細かいサポートまで大変お世話になりました．感謝いたします．

また，同じ NII 所属の山本研究室の楠戸健一郎さん，桝本尚之さん，同期の高田健太君，松川智哉君，浅野研究室の同期の山田淳二さんのおかげで，本郷から遠く離れた NII においても，飲み会や食事会等で楽しい時間を過ごせました．ありがとうございました．

高校からの同期で数学科の野村亮介君には，何度も研究の相談に乗っていただきました．研究を進める上で，野村君との議論は必要不可欠でした．深く感謝いたします．

最後になりましたが，学生生活をあらゆる面で支えていただいた両親，家族，友人，そして 4 年間を共にした電気系の同期への感謝の意を表して，謝辞とさせていただきたいと思います．皆様，本当にありがとうございました．

鈴木 貴敦

参考文献

[1] Nikolaus Augsten, Denilson Barbosa, Michael H. Böhlen, and Themis Palpanas. Tasm: Top-k approximate subtree matching. In *2010 IEEE 26th International Conference on Data Engineering*, pp. 353–364. IEEE, 2010.

[2] Stephan Börzsönyi, Donald Kossmann, and Konrad Stocker. The skyline operator. In *Proceedings of the 17th International Conference on Data Engineering*, pp. 421–430, Washington, DC, USA, 2001. IEEE Computer Society.

[3] Ceek.jp. `http://ceek.jp/`.

[4] Stefano Ceri and Marco Brambilla. *Search Computing Challenges and Directions*, Vol. 5950. Springer LNCS, 2010.

[5] comScore, Inc. comscore reports global search market growth of 46 percent in 2009. `http://www.comscore.com/Press_Events/Press_Releases/2010/1/Global_Search_Market_Grows_46_Percent_in_2009`.

[6] Matthijs Douze, Hervé Jégou, Harsimrat Sandhawalia, Laurent Amsaleg, and Cordelia Schmid. Evaluation of gist descriptors for web-scale image search. In *International Conference on Image and Video Retrieval*. ACM, july 2009.

[7] R. Fagin. Optimal aggregation algorithms for middleware. *Journal of Computer and System Sciences*, Vol. 66, No. 4, pp. 614–656, June 2003.

[8] Jiawei Han. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[9] J.W. Kim and K.S. Candan. Skip-and-prune: Cosine-based top-k query processing for efficient context-sensitive document retrieval. In *Proceedings of the 35th SIGMOD international conference on Management of data*, pp. 115–126. ACM, 2009.

[10] M. Kitsuregawa and T. Nishida. Special issue on information explosion. *New Generation Compution*, Vol. 28, pp. 207–215, 2010.

[11] Kosmix. `http://kosmix.com/`.

[12] Anukool Lakhina, Mark Crovella, and Christophe Diot. Diagnosing network-wide traffic anomalies. In *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '04, pp. 219–230, New York, NY, USA, 2004. ACM.

[13] Chengkai Li, Kevin Chen-Chuan Chang, Ihab F. Ilyas, and Sumin Song. Ranksql: query algebra and optimization for relational top-k queries. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pp. 131–142, New York, NY, USA, 2005. ACM.

[14] Yi Luo, Xuemin Lin, Wei Wang, and Xiaofang Zhou. Spark: top-k keyword query in relational databases. In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pp. 115–126, New York, NY, USA, 2007. ACM.

[15] Aude Oliva and Antonio Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *International Journal of Computer Vision*, Vol. 42, pp. 145–175, May 2001.

[16] Mirit Shalem and Yaron Kanza. Computing the top-k maximal answers in a join of ranked lists. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, CIKM '10, pp. 1381–1384, New York, NY, USA, 2010. ACM.

[17] I-Fang Su, Yu-Chi Chung, and Chiang Lee. Top-k combinatorial skyline queries. In *Database Systems for Advanced Applications, 15th International Conference, DASFAA 2010*, pp. 79–93, 2010.

[18] The DBLP Computer Science Bibliography. `http://www.informatik.uni-trier.de/~ley/db`.

[19] Martin Theobald, Gerhard Weikum, and Ralf Schenkel. Top-k query evaluation with probabilistic guarantees. In *VLDB '04: Proceedings of the Thirtieth international conference on Very large data bases*, pp. 648–659. VLDB Endowment, 2004.

[20] Panayiotis Tsaparas, Themistoklis Palpanas, Yannis Kotidis, Nick Koudas, and Divesh Srivastava. Ranked join indices. *Data Engineering, International Conference on*, Vol. 0, p. 277, 2003.

[21] UCI Machine Learning Repository. http://archive.ics.uci.edu/ml/.

[22] Chao Wu, Guang-zhong Sun, and Guo-liang Chen. Efficient parallel top-k computation algorithm using symmetry breaking. In *Proceedings of the International*

Symposium on Parallel and Distributed Processing with Applications, ISPA '10, pp. 231–235, Washington, DC, USA, 2010. IEEE Computer Society.

[23] Chuan Xiao, Wei Wang, Xuemin Lin, and Haichuan Shang. Top-k Set Similarity Joins. *2009 IEEE 25th International Conference on Data Engineering*, pp. 916–927, March 2009.

[24] Dong Xin, Jiawei Han, and Kevin C. Chang. Progressive and selective merge: computing top-k with ad-hoc ranking functions. In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pp. 103–114, New York, NY, USA, 2007. ACM.

[25] Dong Xin, Jiawei Han, Hong Cheng, and Xiaolei Li. Answering top-k queries with multi-dimensional selections: the ranking cube approach. In *VLDB '06: Proceedings of the 32nd international conference on Very large data bases*, pp. 463–474. VLDB Endowment, 2006.

[26] XMark – An XML Benchmark Project. `http://www.xml-benchmark.org/`.

[27] Inc. Yahoo. Yahoo Pipes. `http://pipes.yahoo.com/pipes/`.

[28] Pavel Zezula, Giuseppe Amato, Vlastislav Dohnal, and Michal Batko. *Similarity Search: The Metric Space Approach*, Vol. 32 of *Advances in Database Systems*. Springer, 2006.

[29] Lei Zou, Lei Chen, and Yansheng Lu. Top-k subgraph matching query in a large graph. In *PIKM '07: Proceedings of the ACM first Ph.D. workshop in CIKM*, pp. 139–146, New York, NY, USA, 2007. ACM.

発表文献

[1] 鈴木貴敦，山田和広，川原圭博，浅見徹，“LCX を利用した通信システムの TCP 通信性能特性，” 電子情報通信学会 ソサイエティ大会，B-15-22, September. 2009.

[2] 山田和弘， 坂井洋介， 鈴木貴敦，川原圭博， 浅見徹，“東海道新幹線における車内インターネット接続の ICMP による通信特性の測定，” 電子情報通信学会 ソサイエティ大会，B-15-21， September. 2009.

[3] 鈴木貴敦， 山田和弘， 川原圭博， 浅見徹，“Wi-Fi を利用した通信システムの通信性能評価，” 電子情報通信学会 総合大会，B-15-33， March 2010.

[4] 山田和弘， 坂井洋介， 鈴木貴敦， 川原圭博， 浅見徹， “高速移動環境における Wi-Fi を利用した通信システム，” 電子情報通信学会 総合大会， B-15-32, March 2010

[5] Kazuhiro Yamada, Yosuke Sakai, Takanobu Suzuki, Yoshihiro Kawahara, Tohru Asami, and Hitoshi Aida, “A Communication System with a Fast Handover under a High Speed Mobile Environment,” VTC 2010-Fall, pp.1–5 , Ottawa, Canada, September. 2010.

[6] 鈴木貴敦， 高須淳宏， 安達淳， “距離尺度の組み合わせによる Top-*k* 検索の提案，” 第 73 回情報処理学会全国大会，March 2011.

[7] 鈴木貴敦， 高須淳宏， 安達淳， “クラスタリングを利用した距離尺度の組み合わせによる Top-*k* 検索，” 第 10 回科学技術フォーラム，September 2011.

[8] Takanobu Suzuki, Atsuhiro Takasu, Jun Adachi, “Top-*k* Query Processing for Combinatorial Objects Using Euclidean Distance,” In Proceedings of the 15th International Database Engineering & Applications Symposium (IDEAS2011), September 2011.

[9] 鈴木貴敦， 高須淳宏， 安達淳， “クラスタリングを用いた Top-*k* Join 処理，” 第 74 回情報処理学会全国大会，March 2012.（発表予定）