検索対象データの事前インデックスを必要としない Top-k 検索アルゴリズムの提案と評価

佐々木 夢† 欅 惇志†† 宮崎 純††

† 東京工業大学大学院情報理工学研究科計算工学専攻 〒 152–8552 東京都目黒区大岡山 2–12–1
†† 東京工業大学大学院情報理工学院情報工学系 〒 152–8552 東京都目黒区大岡山 2–12–1
E-mail: †{sasaki,keyaki}@lsc.cs.titech.ac.jp, ††miyazaki@cs.titech.ac.jp

あらまし 本稿では検索対象データに関するインデックスを事前に作成できない状況において，任意のスコア関数に適用可能な Top-k 検索アルゴリズムを提案する．情報検索における既存の Top-k 検索では，事前にデータが索引付けされていることを想定しているが，スコア統合を行う検索システムにおいては，スコア統合の直前までデータの分布が不明であるため，データに対する索引付けを行うことができない．そこで本稿では，検索対象データの事前インデックスを用いない Top-k 検索アルゴリズムであるメッシュ分割法を提案する．メッシュ分割法はスコア関数をメッシュに分割し，スコアの高い文書をメッシュ情報から効率的に取得する．実験では様々な条件に対して提案手法を適用し，その性質を調査した．

キーワード 情報検索，Top-k 検索，スコア統合，Copula

1. 序 論

検索システムはユーザが求める情報を膨大な情報の中から正確かつ高速に検索できることが求められる．一般的に，検索システムによる検索は次のように行われる [13]: (1) ユーザは自身の要求を検索システムにクエリとして与える．(2) 検索システムは与えられたクエリから，検索対象データの適合度を計算する．(3) データを適合度の降順に整列し，ユーザに返却する．

ユーザが情報検索を行うとき，ユーザは一般的に適合度が上位のデータに興味がある．そのため，適合度が上位となるデータを効率的に検索できる検索システムは有用である．適合度が上位 k 件のデータを取得する検索は Top-k 検索と呼ばれる [10]．

検索システムの正確さには主に適合度の尺度が関係する．データとクエリから適合度を得る関数をスコア関数と呼ぶ．これまでに検索の高精度化を目指した数多くのスコア関数が提案されている [15], [17]．検索の高精度化のためには，どのクエリに対しても最も高精度となるスコア関数を選択することが望ましいが，ユーザの情報要求は多様で複雑であるため，そのようなスコア関数を一意に選択することは難しい．そのため，複数のスコア関数から計算された適合度を統合し，検索精度を向上させる研究がなされている [5], [7], [12], [18]．複数の適合度を統合して一つのスカラー値を得る関数をスコア統合関数と呼ぶ．

検索システムの速度には主に検索対象データの探索方法が関係する．Top-k 検索を実現する単純な方法は，全データの適合度を求め，全データを適合度の降順に整列する方法である．しかし，この方法は全データの整列にかかる計算コストが大きく，莫大なデータ量を扱う場合には不適当である．そのため，Top-k 検索を高速に処理する多数のアルゴリズムが提案されている [3], [6], [9], [19]．

Top-k 検索はアプリケーションの制約やスコア関数の性質など，その組み合わせによって様々なシナリオを想定することができる．アプリケーションの制約の例として，Fagin ら [6] は一般的な場合のアルゴリズムのほかに，ランダムアクセスが可能でない場合のアルゴリズムも提案している．スコア関数の性質の例として，Heo ら [9] は単調なスコア関数を想定しているのに対し，Zhang ら [19] はスコア関数に関する制約を課していない．

本稿では，検索対象データに関するインデックスを事前に作成できない状況において，任意のスコア関数に適用可能な Top-k 検索を高速に処理するアルゴリズムを提案する．情報検索における既存の Top-k 検索では，事前にデータが索引付けされていることを想定しているが，スコア統合を行う検索システムにおいては，スコア統合の直前までデータの分布が不明であるため，データに対する索引付けを行うことができない．また，Web 検索においては，文書集合の更新が多いため最新のインデックスを維持できず，検索に利用できない場合がある．図 1 はスコア統合を行う検索システムにおける Top-k 検索の処理の流れを示している．図中の D' と S' はそれぞれ検索システムの保持している全データとスコア関数の集合を表す．S'_i は S' の要素でありスコア関数を表す．$\mathcal{Q}(\mathcal{D}, \mathcal{S}, k)$ はデータ集合 $\mathcal{D}$ とスコア関数 $\mathcal{S}$，取得数 k を入力とし，データ集合 $\mathcal{D}$ のデータをスコア関数 $\mathcal{S}$ で評価したときに適合度が最も高くなる k 件のデータとその適合度を要求するクエリである．D は $\mathcal{Q}(D', S'_i, k')$ の結果を外部結合し，値のないセルを計算して得られるデータの表である．S はスコア統合を行うスコア関数を表す．

まず初めに，この検索システムは D' に対して各スコア関数 S'_i で検索を行い，それらの結果を統合して各スコア関数で上位となるデータと各スコア関数の適合度の表 D を得る．その後，その表を用いてスコア統合を行うスコア統合関数 S で Top-k 検索を行う．

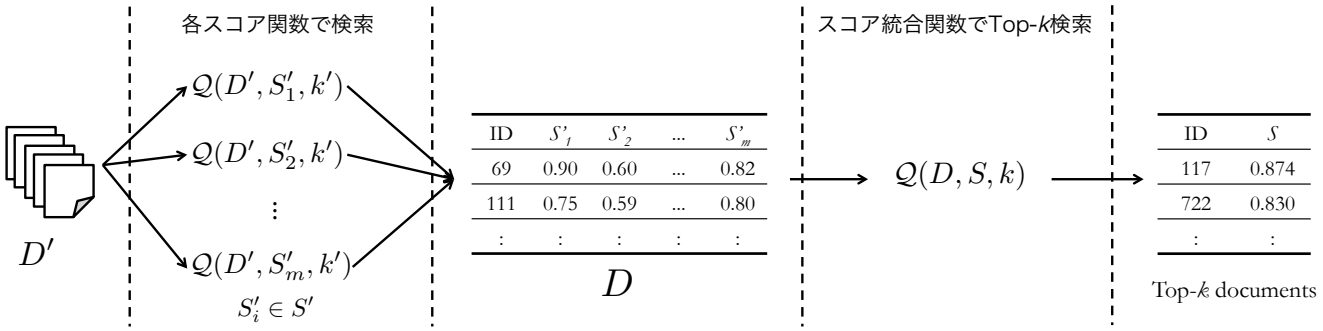


図 1　複数の適合度を統合する検索システムにおける Top-k 検索の流れ

S' に含まれる各スコア関数に対する Top-k 検索時は，検索対象データと索引付けする情報が既知であり，検索対象データに関するインデックスをクエリが与えられる前に用意できるため，それを検索時に利用できる．例えば，適合度の一つである BM25 [17] の Top-k 検索では，データ中の各タームの出現頻度のインデックスを作成しておくことがある [3]．

一方，スコア統合を行うスコア関数に対する Top-k 検索時は，検索対象データと索引付けする情報がすべてのスコア関数の Top-k 検索が完了するまで未知であり，検索対象データに関するインデックスをクエリが与えられる前に用意できず，インデックスを検索時に利用できない．すなわち，スコア統合を行う検索システムでは，事前に用意したインデックスを用いて Top-k 検索を行う既存の手法 [9], [19] を直接は適用できない．

本稿では，検索対象データの事前インデックスを必要としない Top-k 検索アルゴリズムであるメッシュ分割法を提案する．すなわち，図 1 中における D に関するインデックスを用いない Top-k 検索アルゴリズムを提案する．メッシュ分割法は次の性質をもつ: (1) 検索対象データに関するインデックスを必要としない．(2) 任意のスコア関数に対して適用可能なアルゴリズムである．実験では提案手法を様々なデータセットに適用し，既存手法を上回る性能があることを示した．

本稿は以下のように構成される．2. 節では関連研究として Top-k 検索とスコア統合手法について述べる．3. 節で提案手法を説明した後，4. 節で実験の内容とその結果について述べる．最後に 5. 節で本稿の結論と今後の課題を述べる．

2. 関 連 研 究

この節では，Top-k 検索とスコア統合に関する研究を紹介する．2.1 節では，代表的な Top-k 検索手法である Threshold Algorithm (TA) [6] と，任意のスコア関数で適用可能な Top-k 検索手法である OPT* [19] について，それらの特徴と本稿での仮定における問題点を述べる．続く 2.2 節では，スコア統合手法の紹介として，線形結合を用いた手法と Copula [14] を用いた手法について述べる．Copula を用いたスコア統合手法は線形結合を用いた手法より精度が高いため [11]，4. 節の実験では，スコア関数に Copula を用いた手法を採用した．そのため 2.3 節では，Copula に関する基本事項と Copula を用いたスコア関数について説明する．

2.1 Top-k 検索

適合度が上位となるデータを効率的に検索するため，多くの Top-k 検索アルゴリズムが提案されている [6], [9], [10], [19]．

Fagin らは TA を提案した [6]．TA はスコア関数が単調であるときに適用できるアルゴリズムである．TA は各属性のソートされたリストを必要とする．TA の処理は各リストを並行にシーケンシャルにアクセスし，未アクセスのデータの上限スコアを推定しながら行う．推定した上限スコアを超えるデータの数が要求されている数以上になったとき，それ以上のシーケンシャルアクセスを中止する．このようにして，TA はすべてのデータにアクセスすることなく Top-k 検索を完了する．

TA の問題点として，TA が扱えるスコア関数は単調なものに限られることが挙げられる．非単調なスコア関数に対して適用した場合，未アクセスのデータの上限スコアを正しく推定できない．また，スコア関数が単調であっても，TA は各属性のソートされたリストを要求するため，そのようなリストが作成されていない状況では TA をそのまま適用することができない．ソートされたリストの作成コストは高いことに加え，本稿の仮定ではクエリが与えられるたびにソートが必要となるため，TA の適用は非効率的である．

Zhang らは任意のスコア関数に適用可能な Top-k 検索アルゴリズムである OPT*を提案した [19]．OPT*は事前に検索対象データに対して各属性の B^+ 木を作成し，それらを複合して特別なインデックスを作成する．検索の際はインデックスのキーの範囲を利用してスコアの上限を計算し，それを評価関数とした A*探索を実行する．

OPT*の問題点として，検索対象データに関する特別なインデックスが必要であることが挙げられる．また，各属性の B^+ 木の作成の際にソートが発生することと，B^+ 木を元にした特別なインデックスの作成にかかる計算コストが非常に高いことから，本稿の仮定における OPT*の適用は非効率的である．

2.2 スコア統合

最も高精度となるスコア関数を一意に選択することは困難であるため，複数のスコア関数から得られた適合度を統合し，検索精度を向上させる研究がなされている．

Vogt らによって情報検索に導入された線形結合 [18] は，適合度のスコア統合手法としてその有用性の高さを示している [1], [2]．一方，Gerani らは非線形変換を行なった上で線形結合を行なった [8]．Gerani らの手法の精度が線形結合より高

かったことにより，線形従属性では捉えることのできないより複雑な従属性を捉える必要性が示唆された．

Eickhoff らは Copula をスコア統合のために情報検索に導入した [5]．Copula は同時分布とその周辺分布の関係を表す関数であり，分布間の従属性を Copula によって捉えることが可能である [14]．Eickoff らは Copula を用いた統合手法が線形結合よりも高くなることを示した [4]．小松田らは複数の単峰な Copula を混合させることで，Eickhoff らの手法では捉えることのできない複雑な分布を捉え，精度の向上を達成した [11]．

2.3 Copula

Copula は同時分布とその周辺分布の関係を表す関数である．任意の m 次元同時分布関数に対して，$F(x_1,\ldots,x_m) = C\left(F_1(x_1),\ldots,F_m(x_m)\right)$ となる Copula C が存在することが知られている．ここで，$F_i(x_i)$ は F の i 番目の 1 次元周辺分布関数である．この等式は Copula が同時分布とその 1 次元周辺分布関数をつなぐ役割をもっていることを示す．つまり，同時分布をモデル化するときに周辺分布関数と周辺分布間の従属構造を表す Copula を別々に推定することができる．F が連続である場合には C は一意的に定まり，$U = (u_1,\ldots,u_m) = (F_1(x_1),\ldots,F_m(x_m))$ として，$C(u_1,\ldots,u_m) = F\left(F_1^{-1}(u_1),\ldots,F_m^{-1}(u_m)\right)$ と与えられる．

応用上よく用いられている代表的な Copula であるアルキメデス型 Copula を紹介する．区間 $(0,1]$ 上で定義され，正の実数値をとる単調減少凸関数 ϕ が $\phi(1)=0$ を満たすとする．このとき，

$$C_\phi(U) = \phi^{-1}\left(\phi(u_1)+\ldots+\phi(u_m)\right) \tag{1}$$

をアルキメデス型 Copula と呼び，ϕ は C_ϕ の生成素という．$\phi(t) = \frac{t^{-\theta}-1}{\theta}$ のときに Clayton Copula，$\phi(t) = (-\log t)^\theta$ のときに Gumbel Copula，$\phi(t) = -\log\frac{e^{\theta t}-1}{e^\theta-1}$ のときに Frank Copula と呼ぶ．

小松田らは以下の手順で混合 Copula を用いたスコア関数を構成した [11]: (1) 訓練データを複数のクラスタに分ける．(2) クラスタ毎に周辺分布を推定する．(3) クラスタ毎に推定した周辺分布を用いて Copula のパラメータを推定する．(4) 各クラスタに重みを付与して混合し，式 (2)，(3) のスコア関数を得る．

$$C_{mix}(U) = \sum_{i=1}^{c} w_i C_i(U) \tag{2}$$

$$C_{mixprod}(U) = C_{mix}(U)\prod_{i=1}^{m}\sum_{j=1}^{c} w_j F_i^j(u_i) \tag{3}$$

ただし，c，w_i，C_i，F_i^j をそれぞれクラスタ数，i 番目のクラスタに付与した重み，i 番目のクラスタの Copula，j 番目のクラスタの i 番目の周辺分布とする．

3. 提 案 手 法

この節では，提案手法であるメッシュ分割法について説明する．この提案手法は次の二つの特徴を持つ Top-k 検索アルゴリズムである: (1) 検索対象データに関するインデックスを必要としない．(2) 任意のスコア関数に対して適用可能である．

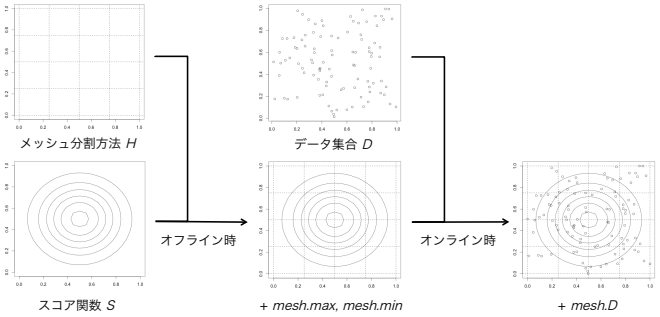


図 2　メッシュ分割法におけるスコア関数の空間情報の生成

3.1 節では提案手法のアルゴリズムを説明する．本手法はメッシュ分割方法を変更することで様々なバリエーションを持つ．そのため，3.2 節でメッシュ分割方法の 1 つである等分割法について述べる．

3.1 アルゴリズム

本手法は事前にスコア関数 S が値を取り得る空間をメッシュに分割し，各メッシュで S の空間情報を作成する．クエリ実行時には，データ集合 D に含まれるデータをメッシュ分割方法 H に従ってメッシュに配置し，S の空間情報を利用して効率的に探索することで Top-k 検索を実行する．

アルゴリズムの詳細を述べる前に，用語の定義を行う．データ集合 D はデータ d の集合であり，データ d は m 個の属性を持つ．データ d の i 番目の属性の値の定義域を A_i とするとき，データ集合 D の定義域 $\mathrm{Dom}(D)$ は $\prod_{i=1}^{m} A_i$ となる．スコア関数 S は $\mathrm{Dom}(D)$ から実数の集合 $\mathbb{R}$ への写像である．メッシュ分割方法 H は $\mathrm{Dom}(D)$ からメッシュ集合 M への写像である．メッシュ集合 M はメッシュ $mesh$ の集合であり，メッシュ $mesh$ はメッシュデータ集合 $mesh.D$ とメッシュの最大値 $mesh.max$，最小値 $mesh.min$ を持つ．メッシュデータ集合は $mesh.D = \{d \mid H(d \in D) = mesh\}$ と定義される．すなわち，データ集合 D のうちメッシュ分割方法 H によってそのメッシュに割り当てられるデータの集合である．メッシュの最大値と最小値は，それぞれそのメッシュが取り得る最大と最小のスコアである．それぞれ式 (4)，(5) で定義される．

$$mesh.max = \max\{S(d) \mid H(d \in \mathrm{Dom}(D)) = mesh\} \tag{4}$$

$$mesh.min = \min\{S(d) \mid H(d \in \mathrm{Dom}(D)) = mesh\} \tag{5}$$

図 2 は各メッシュでスコア関数の空間情報が生成されるまでの流れを示している．オフラインは検索対象データが与えられていない状態を表し，オンラインは検索対象データが与えられた状態を表す．まず，メッシュ分割方法とスコア関数を入力として，各メッシュの最大値と最小値が求められる．この計算にはデータ集合が必要ないため，クエリが与えられる前に計算可能である．その後，Top-k クエリ処理時にデータ集合が与えられ，各メッシュのメッシュデータ集合が得られる．

Algorithm 1 メッシュ分割法 (オフライン時)

Input: S, H
Output: M
  $M \leftarrow \mathrm{Im}(H)$
  for $mesh \in M$ **do**
    //メッシュの最大値と最小値の計算
    calc_mesh_score($mesh, S, H$)
  end for
  return M

メッシュの最大値と最小値を求める方法は大きく分けて 3 種類ある．1 つ目はスコア関数の勾配を計算し，それを利用する解析的な方法 [16] であり，2 つ目は山登り法や共役勾配法のような探索型の方法 [16] である．3 つ目はスコア関数やメッシュ分割方法が特定の性質を持つ場合にのみ適用可能な特殊な方法である．例えば，メッシュ分割方法が図 2 中の H でありスコア関数が単調増加の場合，あるメッシュの最大値はそのメッシュ内で点 $(1, 1)$ に最も近い点 p をスコア関数に適用することで計算できる．

本手法の効率的なデータの探索には次の事実を利用する．Top-k 検索 $\mathcal{Q}(\mathcal{D}, S, k)$ において，$S(d \in \mathcal{D}) \geqq \tau$ となるデータの数が k 以上ならば，データ集合 $\mathcal{A} = \{d \mid S(d \in \mathcal{D}) < \tau\}$ の要素は $\mathcal{Q}$ の解答でない．この事実を用いることで，メッシュの最大値が τ 未満のメッシュに含まれるデータは $\mathcal{Q}$ の解答でないことが保証される．

閾値 τ は以下の手順で求める．メッシュ集合にメッシュの最小値の降順でアクセスし，各メッシュのメッシュデータ集合の大きさを加算する．その値が k 以上になったときにアクセスしているメッシュの最小メッシュスコアを τ とする．メッシュへはメッシュの最小値の降順でアクセスしているため，スコアが τ 以上のデータが k 個以上あることが保証される．

候補データの抽出時は，メッシュ集合にメッシュの最大値の降順でアクセスする．メッシュの最大値が τ 以上ならば，そのメッシュのメッシュデータ集合のデータをスコアリングしヒープに挿入する．メッシュの最大値が τ 未満ならば，メッシュへのアクセスを終了し，ヒープのソートを行う．

アルゴリズム 1，2 はそれぞれ提案手法がオフライン時とオンライン時に実行する処理の疑似コードである．なお，アルゴリズム 1 中の $\mathrm{Im}(H)$ は H の値域を表す．

3.2 等分割法

この節ではメッシュ分割方法の 1 つである等分割法を紹介する．等分割法は図 3 のように，データの各属性値の定義域を $2^h (h \geqq 1)$ 個に等分する分割方法である．

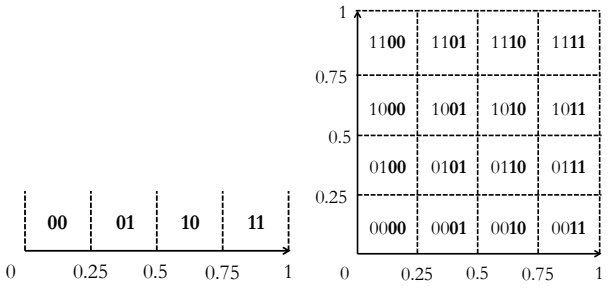
メッシュに割り当てる番号 (メッシュ番号) について説明する．データの属性数が 1 のとき，メッシュ番号は図 3(a) のように左から順番に与えられる．属性数が 2 のとき，メッシュ番号は，各属性に対して番号を属性数が 1 のときのように与え，片方の属性の番号に 2^h をかけた番号 (すなわち h 回左シフトした番号) をもう一方に加えることで与えられる (図 3(b))．一般に属性数が m のときのメッシュ番号は，i 番目の属性の番号を p_i としたとき式 (6) で与えられる．

$$\sum_{i=1}^{m} p_i \times 2^{h(i-1)} \tag{6}$$

データ $d = \langle a_1, \ldots, a_m \rangle$ が属すメッシュ番号は各属性の番号 p_i を計算し，式 (6) で求められる．p_i は a_i を 2^h で割った商を用いることで計算できる．データ d が属すメッシュ番号を計算するのにかかるコストは $\mathcal{O}(m)$ となる．

Algorithm 2 メッシュ分割法 (オンライン時)

Input: D, S, k, M, H
Output: Top-k results
  // データのメッシュ配置 (Assigning)
  for $d \in D$ **do**
    $H(d)$.data.push(d)
  end for
  // 閾値計算 (Threshold)
  $a \leftarrow 0$
  for $mesh \in M$ by descending order of minimum mesh score **do**
    $a \leftarrow a + |mesh.data|$
    if $a \geqq k$ **then**
      $\tau \leftarrow mesh.min$
      break
    end if
  end for
  // 候補データ抽出 (Scoring)
  heap $\leftarrow S(d)$ を優先度とする優先度付きキュー
  for $mesh \in M$ by descending order of maximum mesh score **do**
    if $mesh.max < \tau$ **then**
      break
    else
      for $d \in mesh.D$ **do**
        heap.push(d)
      end for
    end if
  end for
  // 候補データのソート (Sorting)
  for $i = 1$ to k **do**
    result.push(heap.top())
    heap.pop()
  end for
  return result



(a) 属性数 1 ($h = 2$)　(b) 属性数 2 ($h = 2$)

図 3　等分割法

表 1　実験データのパラメータ

パラメータ	値
データサイズ n	5000, 10000, 500000, 1000000, 1500000, 2000000, 2500000
属性数 m	2, 3, 4, 5
データ分布 $dist$	**unif**, **corr**, **anti**

例: メッシュ分割方法を $h = 2$ の等分割法とし，$A_i = [0, 1]$ とする．データ $d = \langle 0.1, 0.9, 0.4 \rangle$ が割り当てられるメッシュ番号は，2 進数で $p_1 = 00$，$p_2 = 11$，そして $p_3 = 01$ となるため，式 (6) より 011100 である．すなわち，$H(d) = mesh_{011100}$ である．

4. 実　　験

この節では，提案手法であるメッシュ分割法を評価する．評価項目は Top-k 検索の実行時間とスコアリングされたデータ数，メモリ消費量とした．4.1 節では実験環境について述べ，4.2 節では実験結果を述べる．

4.1 実 験 環 境

実験は CPU Intel(R) Core(TM) i7-2600K 3.40GHz とメインメモリ 8GB で行った．オペレーティングシステムは Ubuntu 16.04 である．すべてのアルゴリズムは C++で実装した．次の提案手法と比較手法を実装した．(1) **mesh:** 3. 節で説明したメッシュ分割法であり，提案手法である．メッシュ分割方法は 3.2 節で述べた等分割法である．スコア関数が作る空間にインデックスを作成する手法である．(2) **naive:** 全データをスコアリングした後にソートを行い，上位 k 件を得る手法である．インデックスを必要としない手法である．(3) **TA:** 2. 節で紹介した Fagin ら [6] が提案したアルゴリズムである．検索対象データに関するインデックスを作成する手法である．(4) **OPT*:** 2. 節で紹介した Zhang ら [19] が提案したアルゴリズムである．検索対象データ、ならびにスコア関数が作る空間の両方にインデックスを作成する手法である．

TA と OPT*を実行するには，各属性のソートされたリストと B^+ 木を複合したインデックスがそれぞれ必要である．本稿の前提では，検索対象データに関する事前インデックスをクエリが与えられる前に用意できない．そのため，この実験では，TA と OPT*は検索対象データが与えられた後に事前インデックスを構築する．したがって，TA と OPT*の実行時間とメモリ消費量は，本来の Top-k 検索のための処理に加えて，事前インデックス構築に費やした時間とメモリも測定される．また，OPT*の B^+ 木の次数は先行研究 [19] と同じく 200 とした．

スコア関数は単調と非単調の 2 種類のスコア関数を用意した．単調なスコア関数は 2.3 節で述べた混合 Copula を用いたスコア関数 C_{mix} である (式 (2))．Copula には Clayton Copula を用いた．非単調なスコア関数は多変量正規分布の同時密度関数 $f_{gaussian}$ である．$f_{gaussian}$ の平均は $(0.5, \ldots, 0.5)$，分散共分散行列は単位行列 I とした．

実験データはデータサイズ n と属性数 m，データ分布 $dist$ をパラメータとして生成した．パラメータの値は表 1 に示した．

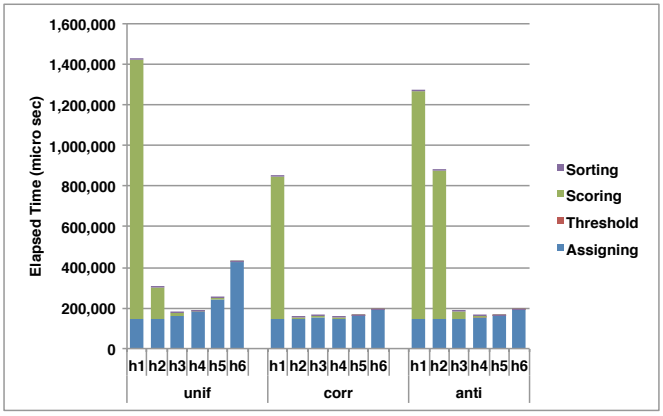


図 4　分割回数 h の変化 (メッシュ分割法)

表 2　分割回数 h の変化 (メッシュ分割法)

	$h=1$	$h=2$	$h=3$	$h=4$	$h=5$	$h=6$
scored_data	1875377	272873	34121	4191	848	300
メモリ (MB)	656	633	625	624	625	639

すべての属性値の定義域は $[0, 1]$ とした．データ分布 $dist$ について説明する．**unif** は一様分布を表す．**corr** と **anti** はそれぞれ $N((0.5, \ldots, 0.5)^T, \Sigma_{corr})$ と $N((0.5, \ldots, 0.5)^T, \Sigma_{anti})$ に従う多変量正規乱数を n 個生成した後，範囲 $[0, 1]$ で正規化した分布である．**corr** は任意の 2 属性の相関が ρ である分布を表し，**anti** は 1 つか全ての属性との相関が $-\rho$ である分布を表す．なお，Σ_{corr} と Σ_{anti} はそれぞれ式 (7) と (8) で表される．本実験において，ρ は 0.8 とした．

$$\Sigma_{corr} = \begin{pmatrix} 1 & \rho & \cdots & \rho \\ \rho & 1 & \cdots & \rho \\ \vdots & \vdots & \ddots & \vdots \\ \rho & \rho & \cdots & 1 \end{pmatrix} \tag{7}$$

$$\Sigma_{anti} = \begin{pmatrix} 1 & -\rho & \cdots & -\rho \\ -\rho & 1 & \cdots & \rho \\ \vdots & \vdots & \ddots & \vdots \\ -\rho & \rho & \cdots & 1 \end{pmatrix} \tag{8}$$

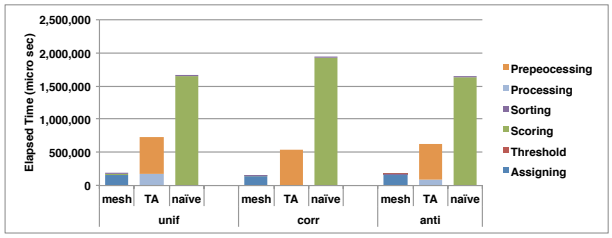
4.2 実 験 結 果

等分割法は分割回数 h をパラメータに持つため，他の手法との比較を行う前に，4.2.1 節でメッシュ分割法の性質を評価する実験の結果を述べる．その後 4.2.2 節で単調なスコア関数における比較実験の結果を述べ，4.2.3 節で非単調なスコア関数における比較実験の結果を述べる．

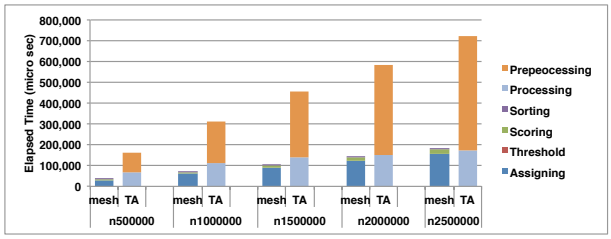
4.2.1 メッシュ分割法の性質

図 4 は $n = 2500000, m = 3, k = 100$, スコア関数 C_{mix} の条件のもと，h と $dist$ を変化させたときのメッシュ分割法の実行時間を示している．図中の実行時間の凡例はアルゴリズム 2 のコメントの処理名と対応している．また，分布が unif のときのスコアリングされたデータ数 (scored_data) とメモリ使用量を表 2 に示した．

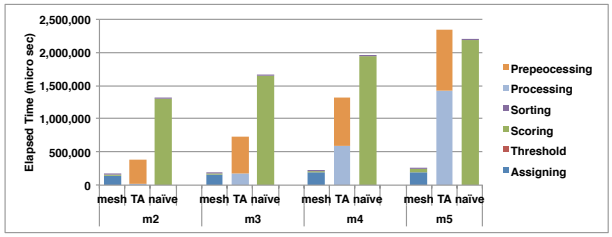
表 2 より，メッシュ分割法では分割回数の増加とともにスコアリングされるデータ数が減少していることが分かる．実行時間は，図 4 が示すように，基本的には分割回数の増加とともに減少している．これは C_{mix} の計算コストが高いため，メッ



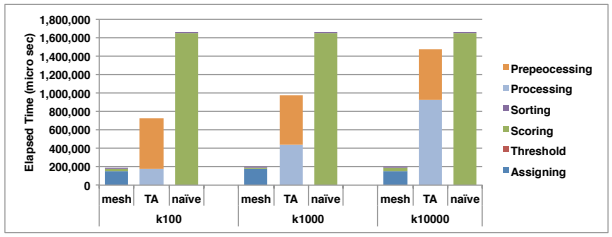
(a) データ分布 $dist$ の変化



(b) データ数 n の変化



(c) 属性数 m の変化



(d) 取得数 k の変化

図 5　比較 (単調なスコア関数)

表 3　データ数 n の変化 (単調なスコア関数)

	手法	$n = 500000$	$n = 1000000$	$n = 1500000$	$n = 2000000$	$n = 2500000$
scored_data	mesh	6767	13800	20393	27083	34121
	TA	76731	123660	158731	170901	192578
メモリ (MB)	mesh	114	213	350	413	625
	TA	149	285	491	557	841
	naive	117	220	332	395	588

シュ分割によるスコアリングされるデータ数の減少が実行時間の短縮に大きく寄与していると考えられる．一方で，データのメッシュ配置 (Assigning) にかかる時間は分割回数の増加とともに増加している．これは大量に生成されたメッシュへの空間的なアクセスコストが大きくなったためと考えられる．unif はデータが一様に散らばっており，空間的なアクセスコストの影響を最も大きく受けるため，corr と anti と比較してメッシュ配置の実行時間が長くなっていると考えられる．

4.2.2　単調なスコア関数における比較

図 5 にスコア関数を C_{mix} としたときの実験結果を示す．図中の凡例の Preprocessing と Processing は TA で行われる処理の凡例であり，それぞれ前処理であるソートと Top-k 処理を表す．OPT*は用意したデータセットでは実行が完了しなかったため，実験結果に載せていない．パラメータは特に指定がない限り $n = 2500000, m = 3, k = 100, dist$ =unif とする．また，mesh の分割回数 h は $hm \leqq 20$ を満たす h のうち，最も実行時間が短かったものを選択した．また，図 5(b) では，視認性のため naive の結果は載せていないが，(b) の naive の結果はすべてのデータ数において mesh の実行時間の約 9.7 倍だった．mesh はすべての条件において Top-k 検索を他の処理より早く完了した．また，属性数が 2 以外の条件においては，仮に TA の Preprocessing が行われていたとしても，mesh は TA と同程度かそれ以上に優れた性能を示した．以降は各実験結果の詳細を述べる．

図 5(a) はデータ分布を変化させたときの比較を表している．mesh の実行時間は分布の変化に対してロバストである一方，TA は分布の変化に敏感である．TA は分布が corr のとき Processing の時間が短くなるが，unif のときに長くなる．TA が Top-k 検索の早期終了に用いる閾値の決定方法から，TA の性能が分布に依存することが分かり，この結果はそれを支持する．

図 5(b) はデータ数を変化させたときの比較を表している．Assigning が mesh の実行時間の大半を占めており，その実行時間がデータ数に線形に増加するため，全体の実行時間も線形に増加している．TA は Preprocessing において，属性ごとに整列処理を行うため，データ数の増加に伴う Preprocessing の実行時間の増加が目立つ．

図 5(c) は属性数を変化させたときの比較を表している．mesh の Assigning は属性数に線形に増加している．これは 3.2 節で述べたデータのメッシュ番号の計算にかかるコストと一致する．TA は属性数の増加に伴う実行時間の増加量が mesh と比べて大きい．これはデータ分布が unif であるため，TA の設定した閾値を超えるデータがリストの前方に出現しにくく，多くのデータを評価してしまうためである．属性数が増えるほどこの影響が大きくなる．

図 5(d) は取得数を変化させたときの比較を表している．取得数の変化は，mesh の実行時間において主に Scoring と Sorting に影響を与える．取得数の増加とともに，スコアリングされるデータ数とソートされるデータ数が増えるためである．しかしながら一定以上の分割回数の mesh の場合，Scoring と Sorting の実行時間が全体に占める割合は極めて小さく，取得数の増加の影響は小さい．その一方で，TA は取得数の増加の影響を大

きく受けている．この原因も (c) の考察と同様に，データ分布が unif であるため，閾値を超えるデータがリストの前方に出現しにくいためである．取得数が増加したとき，さらに多くのデータを評価することになる．

表 3 にデータ数を変化させたときの各手法のスコアリングされたデータ数とメモリ使用量を示す．mesh のスコアリングされたデータ数は TA のものと比較して遥かに少なく，効率的に Top-k 検索を処理できていることが分かる．mesh のメモリ使用量は naive と同程度である．TA のメモリ使用量が大きくなっているのは，TA はデータ集合に加えて，Preprocessing で生成した各属性のデータの並びを保持するためである．

4.2.3 非単調なスコア関数における比較

表 4 と図 6 にスコア関数を $f_{gaussian}$ としたときの実験結果を示す．表 4 はデータ数の少ないデータセット D_{small} での実験結果であり，図 6 は 4.2.2 節で用いたデータセット D_{large} での実験結果である．OPT*の前処理の計算コストが非常に高く，D_{large} では実験が完了しなかったため，データ数の少ない D_{small} を用意した．そのため，D_{large} では OPT*との比較は行っていない．また，TA は非単調なスコア関数に対しては適用できないため，比較対象からは除外する．パラメータは特に指定がない限り，D_{small} では $n = 10000, m = 3, k = 100, dist$ =unif とし，D_{large} では $n = 2500000, m = 3, k = 100, dist$ =unif とする．

初めに D_{small} の結果について述べる．表 4 より，mesh は OPT*と naive より早く Top-k 検索を処理できていることが分かる．OPT*は前処理に多大な時間が必要であり，属性数とデータ数にスケールしない．また，OPT*のメモリ使用量は属性数やデータ数の増加とともに爆発的に増加する．そのため，条件によっては OPT*の Processing だけの実行時間が naive の総実行時間より長くなる．また，属性数が 2 以外の条件においては，仮に OPT*の Preprocessing が行われていたとしても，mesh は OPT*より優れた性能を示した．

続いて D_{large} の結果について述べる．図 6 に様々な条件での実験結果を示しているが，mesh が示す傾向は 4.2.2 節と変わらない．スコア関数が C_{mix} より単純な $f_{gaussian}$ に変更されている分，Scoring にかかる時間が短くなり，naive の実行時間が短縮されている．

表 5 にデータ数を変化させたときの各手法のスコアリングされたデータ数とメモリ使用量を示す．mesh でスコアリングされたデータ数は，スコア関数が単調なときの結果と同等であり，非単調なスコア関数においても mesh が有効であることが分かる．mesh のメモリ使用量は naive と同程度であった．

5. 結　　論

本稿では検索対象データに関するインデックスを事前に作成できない状況において，任意のスコア関数に適用可能な Top-k 検索を高速に処理するアルゴリズムであるメッシュ分割法を提案した．実験では，データの属性数やデータ数，データの分布，取得数，スコア関数の種類の組み合わせを変え，様々な条件に提案手法を適用し評価した．メッシュ分割法の実行時間はすべての条件において比較手法より良い結果を示した．

今後の課題は 2 点ある．1 つ目はデータのメッシュ配置の並列化である．本稿の実験結果によると，メッシュ配置の時間はメッシュの分割回数の増加とともに増加するが，このステップは並列化が容易であるため，実行時間を短縮できる．2 つ目の課題はより良いメッシュ分割方法の提案である．本稿ではメッシュ分割方法として等分割法を提案したが，データの分布によっては性能が落ちる．例えば，あるメッシュに全データが入る分布を考えたとき，これは全探索と同様の処理になり，効率が悪い．そのため，データの分布を考慮したメッシュ分割方法を考える必要がある．

謝　　辞

本研究の一部は，JSPS 科研費 JP15H02701，JP26280115，JP16H02908，JP15K20990，JP26540042 の助成を受けたものである．ここに記して謝意を表す．

文　　献

[1] Gloria Bordogna and Gabriella Pasi. A model for a Soft Fusion of Information Accesses on the web. *Fuzzy Sets and Systems*, 148(1):105–118, 2004.

[2] Célia Da Costa Pereira, Mauro Dragoni, and Gabriella Pasi. Multidimensional relevance: A new aggregation criterion. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 5478 LNCS, pages 264–275, 2009.

[3] Shuai Ding and Torsten Suel. Faster top-k document retrieval using block-max indexes. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information - SIGIR '11*, page 993, 2011.

[4] Carsten Eickhoff and Arjen P. de Vries. Modelling Complex Relevance Spaces with Copulas. *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management - CIKM '14*, pages 1831–1834, 2014.

[5] Carsten Eickhoff, Arjen P. de Vries, and Kevyn Collins-Thompson. Copulas for information retrieval. In *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval - SIGIR '13*, page 663, 2013.

[6] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. In *Journal of Computer and System Sciences*, volume 66, pages 614–656, 2003.

[7] Edward A Fox and Joseph A Shaw. Combination of Multiple Searches. *The 2nd Text Retrieval Conference TREC2 NIST SP 500215*, 500-215:243–252, 1994.

[8] Shima Gerani, Chengxiang Zhai, and Fabio Crestani. Score transformation in linear combination for multi-criteria relevance ranking. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 7224 LNCS, pages 256–267, 2012.

[9] Jun Seok Heo, Kyu Young Whang, Min Soo Kim, Yi Reun Kim, and Il Yeol Song. The partitioned-layer index: Answering monotone top-k queries using the convex skyline and partitioning-merging technique. *Information Sciences*, 179(19):3286–3308, 2009.

[10] Ihab Ilyas, George Beskales, and Mohamed Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Computing Surveys*, 40(4):1–58, 2008.

[11] Takuya Komatsuda, Atsushi Keyaki, and Jun Miyazaki. A Score Fusion Method Using a Mixture Copula. *27th Inter-*

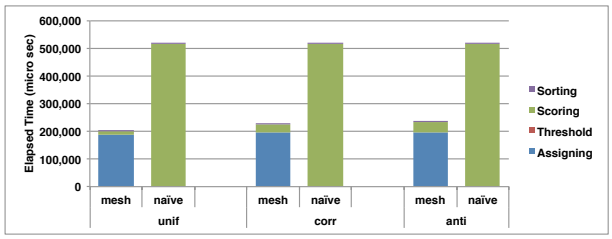
表 4　D_{small} での比較 (非単調なスコア関数)

(a) 属性数 m の変化

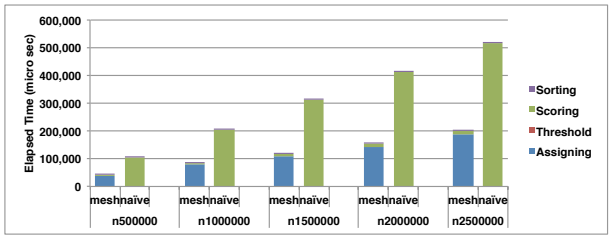
	手法	総実行時間 (μs)	Assigning	Threshold	Scoring	Sorting	Preprocessing	Processing	scored_data	メモリ (MB)
	mesh	756	657	0	91	8	-	-	491	16
m=2	OPT*	122721	-	-	-	-	122625	96	115	20
	naive	1670	-	-	1651	19	-	-	10000	16
	mesh	951	846	2	89	14	-	-	252	18
m=3	OPT*	6516128	-	-	-	-	6512560	3568	105	368
	naive	2347	-	-	2327	20	-	-	10000	16

(b) データ数 n の変化

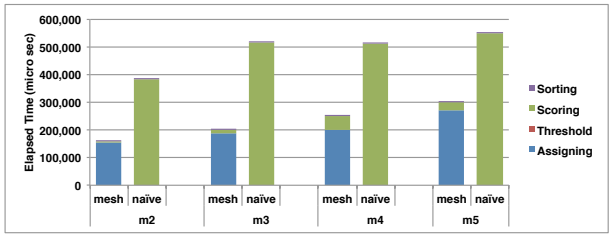
	手法	総実行時間 (μs)	Assigning	Threshold	Scoring	Sorting	Preprocessing	Processing	scored_data	メモリ (MB)
	mesh	531	450	1	74	6	-	-	179	17
n=5000	OPT*	750478	-	-	-	-	749708	770	108	59
	naive	1088	-	-	1072	16	-	-	5000	15
	mesh	951	846	2	89	14	-	-	252	18
n=10000	OPT*	6516128	-	-	-	-	6512560	3568	105	368
	naive	2347	-	-	2327	20	-	-	10000	16



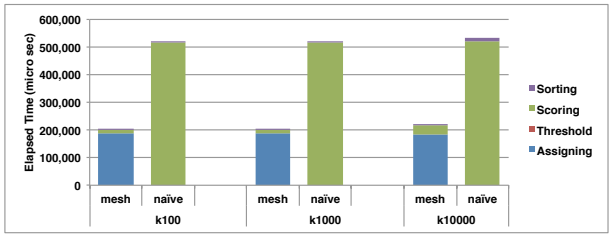
(a) データ分布 *dist* の変化



(b) データ数 n の変化



(c) 属性数 m の変化



(d) 取得数 k の変化

図 6　D_{large} での比較 (非単調なスコア関数)

表 5　データ数 n の変化 (非単調なスコア関数)

	手法	$n = 500000$	$n = 1000000$	$n = 1500000$	$n = 2000000$	$n = 2500000$
scored_data	mesh	6763	13589	20410	27290	34113
メモリ (MB)	mesh	115	216	351	417	625
	naive	117	220	332	395	588

national Conference on Database and Expert Systems Applications, pages 216–232, 2016.

[12] Tie-Yan Y Liu. Learning to Rank for Information Retrieval. *Found. Trends Inf. Retr.*, 3(3):225–331, 2009.

[13] Christopher D Manning, Prabhakar Raghavan, and Hinrich Schütze. Introduction to Information Retrieval. *2008*, 1(c):496, 2008.

[14] Roger B Nelsen. *An Introduction to Copulas*, volume 53. 2013.

[15] Jay M Ponte and W Bruce Croft. A language modeling approach to information retrieval. *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, 98(3):275–281, 1998.

[16] William H Press, Saul a Teukolsky, William T Vetterling, and Brian P Flannery. *Numerical recipes in C (2nd ed.): the art of scientific computing*, volume 29. 1992.

[17] S.E. Robertson, S. Walker, S. Jones, M.M. Hancock-Beaulieu, and M. Gatford. Okapi at TREC-3. *Proceedings of 3rd Text REtrieval Conference*, pages 109–126, 1994.

[18] Christopher C Vogt and Garrison W Cottrell. Fusion Via a Linear Combination of Scores. *Information Retrieval*, 1(3):151–173, 1999.

[19] Zhen Zhang, Seung-won Hwang, Kevin Chang, Min Wang, Christian Lang, and Yuan-Chi Chang. Boolean + ranking: querying a database by k-constrained optimization. *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 359–370, 2006.