情報処理学会論文誌　データベース　Vol.10　No.4　1–5 (Dec. 2017)

テクニカルノート

メニーコアプロセッサを用いた構造的類似度に基づくグラフクラスタリングの高速化

高橋 知克[1,a)]　塩川 浩昭[2,b)]　北川 博之[2,c)]

受付日 2017年6月7日, 採録日 2017年8月3日

概要：構造的類似度に基づくグラフクラスタリング手法 SCAN は，グラフ中からクラスタやハブ，外れ値を高精度で検出できるため幅広く利用されている．しかしながら，SCAN はすべてのエッジの構造的類似度を計算するため，大規模グラフの分析には膨大な計算時間を必要とする．そこで本稿では，メニーコアプロセッサ Intel Xeon Phi を用いた SCAN の高速化手法 SCAN-XP を提案する．提案手法は，並列グラフ処理のボトルネックを解消するとともに，アルゴリズムを Intel Xeon Phi の持つ 512 ビット SIMD 演算に最適化する．本稿では実データを用いた評価実験を行い提案手法の有効性を示す．

キーワード：グラフクラスタリング，SIMD 演算，Intel Xeon Phi

Efficient and Exact Parallel Structural Graph Clustering Algorithm on Intel Xeon Phi Processor

Tomokatsu Takahashi[1,a)]　Hiroaki Shiokawa[2,b)]　Hiroyuki Kitagawa[2,c)]

Received: June 7, 2017, Accepted: August 3, 2017

Abstract: The structural graph clustering method SCAN is successfully used in many applications because it can extract clusters, hubs and outliers from a graph at the same time. However, it is difficult for SCAN to compute large-scale graphs since SCAN needs to evaluate the density for all edges in graphs. To address the above problem, we present SCAN-XP that performs over Intel Xeon Phi. SCAN-XP avoids the bottlenecks that arise from parallel graph computations and it effectively exploits 512 bit SIMD instructions. Our evaluations on real-world graphs demonstrate that SCAN-XP outperforms existing approaches.

Keywords: graph clustering, SIMD instruction, Intel Xeon Phi

1. はじめに

SCAN [1] は密度ベースのクラスタリング DBSCAN [2] を基に考案された構造的類似度に基づくグラフクラスタリング手法である．**図 1** のように SCAN はグラフ中から密に接続されたノードの集合であるクラスタ，クラスタ間を橋渡しするノードであるハブ，およびノイズである外れ値を同時に検出できることから幅広く利用されている．しかしながら，SCAN はクラスタを抽出するためにグラフ $G = \{V, E\}$ に含まれるすべてのエッジに対しての構造的類似度と呼ばれるノード間の接続の強さを示す評価指標を計算する必要がある．そのため，近年増加している大規模グラフに対して，SCAN は膨大な計算時間を必要とする．そのため，大規模グラフに SCAN を適用することは難しい．

SCAN を大規模グラフに適用するために，2015 年以降，いくつかの高速化手法が提案されている．Shiokawa らによる SCAN++ [3] や Chang らによる pSCAN [4]，Mai らによる anySCAN [5] は，実世界のグラフに内包された構造特性の利用や計算結果が明示であるエッジへの計算回数の削

[1] 筑波大学大学院システム情報工学研究科
Graduate School of Systems and Information Engineering, University of Tsukuba, Tsukuba, Ibaraki 305–8577, Japan
[2] 筑波大学計算科学研究センター
Center for Computational Sciences, University of Tsukuba, Tsukuba, Ibaraki 305–8577, Japan
[a)] shihakata@kde.cs.tsukuba.ac.jp
[b)] shiokawa@cs.tsukuba.ac.jp
[c)] kitagawa@cs.tsukuba.ac.jp

© 2017 Information Processing Society of Japan

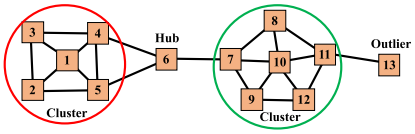

図 1　SCAN によるグラフクラスタリングの例
Fig. 1　Example of SCAN.

減により SCAN の高速化を実現した．しかしながら，1 億以上のエッジを持つ大規模なグラフに対しては，依然として大きな計算時間を必要とする．したがって，SCAN のさらなる処理性能の向上は現在も重要な課題となっている．

そこで本稿ではメニーコアプロセッサ Intel Xeon Phi [6] を用いた高速な構造的類似度に基づくクラスタリング手法 SCAN-XP（Structural Clustering Algorithm for Network over Xeon Phi）を提案する．SCAN において計算時間の大部分を占める構造的類似度の計算はエッジごとに独立している．ゆえに，Intel Xeon Phi の高い並列処理性能を活用することで大幅な高速化が期待できる．SCAN-XP では，Intel Xeon Phi 上で効率的な並列化を行うため，SCAN がかかえる並列化におけるボトルネックを解消するとともに，Intel Xeon Phi が持つ 512 ビット SIMD 演算を効率的に活用した構造的類似度計算手法を提案する．Intel Xeon Phi における並列計算性能を向上させることで，高速かつスケーラブルなグラフクラスタリングを実現する．

2.　従来手法 SCAN

SCAN は core 検出処理，クラスタ検出処理，ハブ・外れ値検出処理の 3 つから構成されている．

core 検出処理：まず，グラフ中のすべてのエッジに対して定義 2.1 に示した構造的類似度を計算し，クラスタの核となるノード core を検出する．

定義 2.1（構造的類似度）　$v, w \in V$ に対する構造的類似度は $\sigma(v,w) = |\Gamma(v) \cap \Gamma(w)|/\sqrt{|\Gamma(v)||\Gamma(w)|}$．ただし，$\Gamma(v) = \{w \in V | \{v,w\} \in E\} \cup \{v\}$ とする．

ノード v とその隣接ノード w の構造的類似度が閾値 ϵ 以上を示した場合，v と w は構造的に類似である．ノード v の構造的に類似なノードの集合を $N_\epsilon(v)$ としたとき，定義 2.2 から $|N_\epsilon(v)|$ が閾値 μ 以上のとき v は core である．

定義 2.2（Core）　$v \in V$ と $\epsilon \in \mathbb{R}$，$\mu \in \mathbb{N}$ が与えられたとき，$|N_\epsilon(v)| \geq \mu$ ならば v は core である．

クラスタ検出処理：次に，グラフ内のクラスタの検出を行う．core v を選択し，v と $N_\epsilon(v)$ に含まれるノードを同じクラスタとする．ここで，同じクラスタとしたノード $u \in N_\epsilon(v)$ が core だった場合はさらに $N_\epsilon(u)$ に含まれるノードを v と同一のクラスタとする．SCAN はすべての core がクラスタに含まれるまで，一連の処理を繰り返し行う．

ハブ・外れ値検出処理：最後に，クラスタに所属していないノードを定義 2.3 に基づきハブまたは外れ値に分類する．

定義 2.3（ハブと外れ値）　グラフ中のいずれのクラスタにも属していないノード u が与えられたとき，その隣接ノードが 2 つ以上のクラスタに属していた場合 u はハブである．そうでない場合 u は外れ値である．

Algorithm 1 SCAN-XP

Input: $G = \{V, E\}$, $\epsilon \in \mathbb{R}$ and $\mu \in \mathbb{N}$
Output: C(Set of clusters), H(Set of hubs), and O(Set of outliers)
1: $\forall v \in V$ are labeled as *unclassified*;
2:
3: // Step 1: Parallel core detection in Section 3.1
4: **for** each edge $(v, w) \in E$ **do in parallel**
5:　run Algorithm 2;
6: **end for**
7:
8: // Step 2: Parallel cluster construction in Section 3.2
9: //$C(v)$:Set of nodes that are belong to the same cluster as node v
10: **for** each core node $v \in V$ **do in parallel**
11:　**for** each $w \in N_\epsilon(v)$ **do**
12:　　**if** $find(v) \neq find(w)$ **then**
13:　　get $C(v) \cup C(w)$ by using $union(v, w)$ and CAS instruction;
14:　　node v and node w are labeled as *cluster-member*;
15:　　**end if**
16:　**end for**
17: **end for**
18:
19: // Step 3: Parallel hubs/outliers detection
20: **for** each node v that is not included in any clusters of C **do in parallel**
21:　**if** $\exists u, w \in \Gamma(v)$ *s.t.* $find(u) \neq find(w)$ **then**
22:　　label node v as hub, and $H = H \cup \{v\}$;
23:　**else**
24:　　label node v as outlier, and $O = O \cup \{v\}$;
25:　**end if**
26: **end for**

3.　提案手法 SCAN-XP

SCAN はすべてのエッジに対して構造的類似度を計算するため，膨大な計算時間がかかる．そこで本稿では，Intel Xeon Phi の高い並列演算性能を用いた高速化手法 SCAN-XP を提案する．SCAN-XP は 2 章で示した 3 つの処理をそれぞれ Intel Xeon Phi により並列化する．SCAN-XP のアルゴリズムを Algorithm 1 に示す．

3.1　core 検出処理の並列化

本節では，core 検出処理のボトルネックである構造的類似度計算のスレッド並列化とデータ並列化について述べる．

3.1.1　構造的類似度計算のスレッド並列化

構造的類似度計算はグラフの各エッジに対して独立であることに着目してスレッド並列化する．優れたキャッシュ効率からグラフ処理において一般的なデータレイアウト CRS（Compressed Row Storage）[7] を用いる．CRS は**図 2** のように，ノード 0 の隣接ノードから順にすべてのノードの隣接ノードが連続して格納されている to 配列，to 配列のどこにどのノードの隣接ノードが存在するかを示すポインタである ptr 配列により構成される．CRS を用いてスレッド並列化する場合，to 配列からは要素がどのノードの隣接ノードなのか分からないため，ptr 配列の要素単位，すなわちノード単位での並列化を行う必要がある．しかし，実世界のグラフはべき乗則に従う次数分布を

© 2017 Information Processing Society of Japan

持ち [8]，各ノードの持つ隣接ノード集合の大きさに偏りが生じるため，スレッド間のタスク割当量が偏り並列化効率が低下する．

この問題を解決するために，図 2 に示すように to 配列の要素と 1 対 1 で対応した逆ポインタで from 配列を導入する．これにより，CRS の持つ空間効率やキャッシュ効率を維持しつつ，エッジ単位でのスレッド並列化を可能とする．

3.1.2　共通隣接ノード検出処理のデータ並列化

構造的類似度計算の隣接ノード集合の積集合計算がボトルネックであるため，本稿では Inoue らによって提案された高速な積集合計算手法 [9] に基いて，SIMD を用い実装する．

Algorithm 2 を用い処理を説明する．2 行目から 4 行目にて，積集合を計算する 2 つの隣接ノード配列の先頭と最後を示すポインタ，そして，各配列からレジスタにロードされるノード数 α，β を決定する．この α，β は配列のサイズに違いがある場合，大きな配列からより多くのノードをレジスタにロードするよう決定する．次に 6 行目から 17 行目では，sort merge join による配列の比較が行われている．まず，各配列からそれぞれ α，β 個のノードが選択され，SIMD レジスタ上にロードされる．そして，compare 命令を用いてロードされたノードの全比較を行い，一致数が検出される．最後に，各配列からロードされたノードのうち，一番大きなノードどうしを比較し，小さい方のポインタが α もしくは β 分だけ進められる．この処理をどちらかのポインタが配列をすべて参照するまで繰り返し行う．

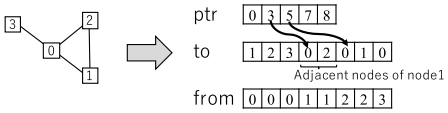

図 2　グラフを格納するデータ形式の例
Fig. 2　Example of graph data representation.

Algorithm 2 SIMD による構造的類似度計算

Input: $v, w \in V$,
Output: $\sigma(v, w)$
1: // Initialization
2: select α and β according to difference between two adjacent array size
3: get head pointers vp and wp from $\Gamma(v)$ and $\Gamma(w)$, respectively;
4: get tail pointers v_end and w_end from $\Gamma(v)$ and $\Gamma(w)$, respectively;
5:
6: **while** $vp < v_end$ && $wp < w_end$ **do**
7:　load α and β nodes into SIMD register reg_v and reg_w from $\Gamma(v)$ and $\Gamma(w)$, respectively;
8:　get the number of common nodes c between reg_v and reg_w by using SIMD instructions;
9:　$vw_common = vw_common + c$;
10:　**if** $vp+\alpha == wp+\beta$ **then**
11:　$vp = vp + \alpha$, $wp = wp + \beta$;
12:　**else if** $vp + \alpha > wp + \beta$ **then**
13:　$wp = wp + \beta$;
14:　**else**
15:　$vp = vp + \alpha$;
16:　**end if**
17: **end while**
18: $\sigma(v, w) = (vw_common + 2)/\sqrt{|\Gamma(v)||\Gamma(w)|}$;

3.2　クラスタ検出処理の並列化

SCAN のクラスタ検出処理は core からどこまでのノードがクラスタに含まれるのか探索的に求める必要がある．ゆえに，どのノードをどのスレッドで処理すればよいのか決定できず，容易にスレッド並列化できない．そこで本稿では，ノードが属する素集合の特定と素集合どうしの統合を高速に行える Union-Find 木と CAS 命令を用いて効率的かつ容易に並列処理できるアルゴリズムを提案する．

具体的な処理を Algorithm 1 を用いて説明する．12 行目の $find(v)$ は v の属するクラスタを検出する命令であり，13 行目の $union(v, w)$ は，v の所属するクラスタと w の属するクラスタを統合する命令である．10 行目から 17 行目において，各スレッドは，core とその N_ϵ で構成されるクラスタを検出していく．このとき，各クラスタ間で格納されたノードが重複した場合，それらのクラスタは union 命令により統合されるこの処理はすべての core が選択されるまで繰り返し実行する．また，union 命令による統合処理は，スレッド間の書き換えの衝突を検知できる CAS 命令を使用して行われ，スレッド間のデータの整合性を保つ．

4.　評価実験

本章では，実データを用いた測定実験により，提案手法の有効性を検証する．実験環境の詳細を**表 1** に，測定に使用した実データセットを**表 2** に示す．

また，webbase2001 は非常に大きなデータセットであり，KNL 以外ではメモリ不足により実行できなかった．

SCAN におけるユーザ定義の閾値 ϵ および μ について，SCAN と SCAN-XP はともに処理時間に対し影響を受けることはほぼないため，本稿においては $\epsilon = 0.3$，$\mu = 2$ と定める．また，SCAN-XP における α，β は予備実験により，良い性能を示した値を設定した．

表 1　実験環境
Table 1　Experimental environment.

CPU	Xeon E5 1620 (Xeon)	Xeon Phi 3120A (KNC)	Xeon Phi 7250 (KNL)
Main memory	16 GB	6 GB	MCDRAM 16 GB DDR4 96 GB
Clock rate	3.5 GHz	1.10 GHz	1.4 GHz
# of Core	4	57	68
# of Thread	8	228	272
OS	CentOS 7	Linux 3.10	Cent OS 7
SIMD	AVX2	IMCI	AVX-512

表 2　データセットの詳細
Table 2　Real-world datasets.

Dataset	Nodes	Edges	Data source
youtube	1,134,890	2,987,624	com-youtube [10]
BerkStan	685,230	6,649,470	web-BerkStan [10]
Pokec	1,632,803	22,301,964	soc-Pokec [10]
com-LJ	3,997,962	34,681,189	com-LiveJournal [10]
soc-LJ	4,846,609	42,851,237	soc-LiveJournal1 [10]
Orkut	3,072,441	117,185,083	com-Orkut [10]
webbase	11,5554,441	854,809,761	webbase2001 [11]

© 2017 Information Processing Society of Japan

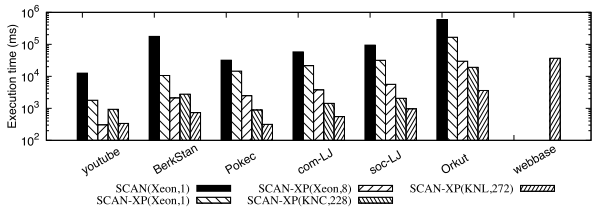

図 3　実データに対する実行時間

Fig. 3　Runtimes for real-world datasets.

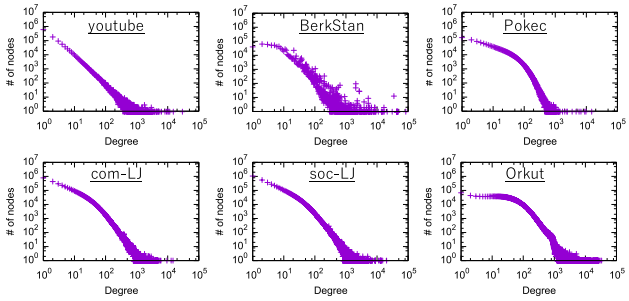

図 4　データセットの次数分布

Fig. 4　Degree distribution of real-world datasets.

実行時間の比較

Xeon 上で逐次実行された SCAN と各計算機で実行された SCAN-XP の実行時間比較を図 **3** に示す．いずれの実行環境でも SCAN-XP が SCAN より高速であることが分かる．特に KNL 上で 272 スレッド実行された SCAN-XP はすべてのデータセットにおいて SCAN より 100 倍以上高速であり，8 億 5 千万のエッジを持つ大規模グラフ webbase を 36 秒で処理した．しかし，小さなデータセット youtube および BerkStan では，Xeon Phi の並列処理性能を使いきれず，他のデータセットほどの高速化はされなかった．

また，SCAN および SCAN-XP ともにエッジ数が少ない BerkStan に対する実行時間が Pokec や com-LJ 等のエッジ数が多いグラフに対する実行時間を上回っている．これは図 **4** に示したように，BerkStan は他のデータセットと比較して高い次数を持つノードが多く存在し，構造的類似度の積集合計算に時間を要することが原因だと考えられる．

スケーラビリティ

スレッド数の増加にともなう SCAN-XP の性能向上について図 **5** に示す．図から，youtube および BerkStan は，エッジ数が少ないため Intel Xeon Phi の並列処理性能を使いきれていないことが分かる．一方で，他のデータセットはスレッド数が増えるとともに性能が向上した．

SIMD 演算の性能

SIMD によるデータ並列化の性能を図 **6** に示す．図は，KNL 上で実行された SCAN（SIMD なし）に対しての SCAN-XP（SIMD あり）の性能比を示している．youtube では約 8 倍，BerkStan では約 20 倍高速化され，他のデータセットでも．約 3〜4 倍の高速化を達成した．

youtube と BerkStan で著しい高速化が達成された理由

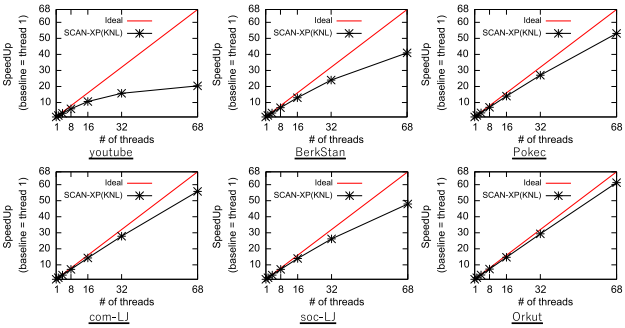

図 5　スレッド数増加にともなう性能向上

Fig. 5　Scalability.

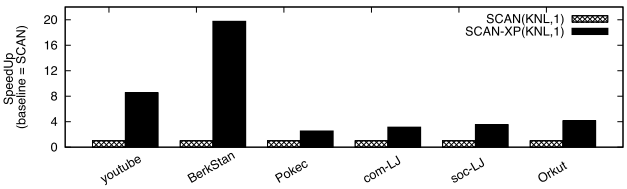

図 6　SIMD 演算による性能向上

Fig. 6　Performance improvement by SIMD instruction.

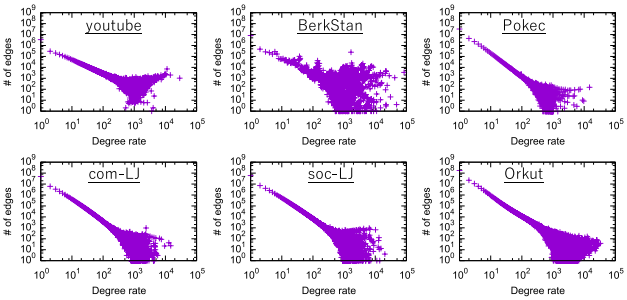

図 7　隣接ノード配列のサイズ比分布

Fig. 7　Size ratio distribution of adjacent node arrays.

は，図 **7** に示した積集合計算をする隣接ノードのサイズ比分布から推察できる．この図は横軸が積集合計算をする配列のサイズの比を表し，縦軸がそのサイズ比の組合せの頻度を示す．図 7 に示したように，youtube と BerkStan はサイズ比が大きく異なる組合せが他のデータセットと比べて多い．配列のサイズ比が大きく異なる場合，SCAN-XP は 3.1.2 項で示したように大きい方の配列から多くのノードを SIMD レジスタにロードすることで高速に積集合計算を行うため SIMD 演算の性能が向上したと考えられる．

5.　まとめと今後の課題

Intel Xeon Phi を用いて従来手法 SCAN より 100 倍以上高速にクラスタリングを行う SCAN-XP を提案した．今後の課題として，SIMD 演算のさらなる最適化や複数の Intel Xeon Phi を用いた並列化手法について検討する．

謝辞　本研究は，JSPS 科研費 JP26280037 JP16H06650，平成 28 年度筑波大学研究基盤支援プログラムならびに筑波大学計算科学研究センター学際共同利用プロジェクト

© 2017 Information Processing Society of Japan

（COMA）の助成を受けたものである．

参考文献

[1] Xu, X., Yuruk, N., Feng, Z. and Schweiger, T.A.J.: SCAN: A Structural Clustering Algorithm for Networks, *KDD*, New York, NY, USA, ACM, pp.824–833 (2007).

[2] Ester, M., Kriegel, H., Sander, J. and Xu, X.: A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise, *KDD*, pp.226–231 (1996).

[3] Shiokawa, H., Fujiwara, Y. and Onizuka, M.: SCAN++: Efficient Algorithm for Finding Clusters, Hubs and Outliers on Large-scale Graphs, *PVLDB*, Vol.8, No.11, pp.1178–1189 (2015).

[4] Chang, L., Li, W., Lin, X., Qin, L. and Zhang, W.: pSCAN: Fast and Exact Structural Graph Clustering, *ICDE*, pp.253–264 (2016).

[5] Mai, T.S., Dieu, S.M., Assent, I., Jacobsen, J., Kristensen, J. and Birk, M.: Scalable and Interactive Graph Clustering Algorithm on Multicore CPU, *ICDE*, San Diego, CA, USA, IEEE, pp.349–360 (2017).

[6] Jeffers, J. and Reinders, J.: *Intel Xeon Phi Coprocessor High Performance Programming*, *1st edition*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2013).

[7] Demmel, J., Dongarra, J., Ruhe, A. and van der Vorst, H.: *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA (2000).

[8] Faloutsos, M., Faloutsos, P. and Faloutsos, C.: On Power-Law Relationships of the Internet Topology, *SIGCOMM*, NewYork, NY, USA, ACM, pp.251–262 (1999).

[9] Inoue, H., Ohara, M. and Taura, K.: Faster Set Intersection with SIMD instructions by Reducing Branch Mispredictions, *PVLDB*, Vol.8, No.3, pp.293–304 (2015).

[10] Leskovec, J. and Krevl, A.: SNAP Datasets: Stanford Large Network Dataset Collection, available from ⟨http://snap.stanford.edu/data⟩ (2014).

[11] Boldi, P. and Vigna, S.: The WebGraph Framework I: Compression Techniques, *Proc. of the Thirteenth International World Wide Web Conference* (*WWW 2004*), Manhattan, USA, pp.595–601, ACM Press (2004).

高橋 知克

2016 年木更津工業高等専門学校制御・情報システム工学専攻卒業．現在，筑波大学大学院システム情報工学研究科に在学中．大規模グラフに対するクラスタリングの高速化に関する研究に従事．日本データベース学会学生会員．

塩川 浩昭　（正会員）

2009 年筑波大学第三学群情報学類卒業．2011 年同大学院システム情報工学研究科博士前期課程修了．2011 年 4 月から日本電信電話株式会社勤務の後，2015 年筑波大学大学院博士後期課程修了，博士（工学）．2015 年より筑波大学計算科学研究センター助教．データベース，データマイニング，特に大規模グラフ分析アルゴリズムの高速化に関する研究に従事．日本データベース学会，ACM，AAAI 各会員．

北川 博之　（正会員）

1978 年東京大学理学部物理学科卒業．1980 年同大学院理学系研究科修士課程修了．日本電気（株）勤務の後，筑波大学電子・情報工学系講師，同助教授を経て，現在，筑波大学計算科学研究センター教授．理学博士（東京大学）．データベース，情報統合，データマイニング，情報検索等の研究に従事．日本データベース学会前会長，情報処理学会フェロー，電子情報通信学会フェロー，ACM，IEEE，日本ソフトウェア科学会各会員．本会フェロー．

（担当編集委員　藤原 真二）

© 2017 Information Processing Society of Japan