情報論的学習理論テクニカルレポート 2009
Technical Report on Information-Based Induction Sciences 2009 (IBIS2009)

近傍ハッシュを用いた高速なグラフカーネル
A Fast Graph Kernel Using Neighborhood Hash

比戸将平* 鹿島久嗣†
Shohei Hido Hisashi Kashima

Abstract: We propose a novel graph kernel based on the structural characteristics of labeled graphs. The idea is to convert node labels to binary arrays and to compare the nodes by logical operations on the set of the adjacent node labels. The proposed kernel can be computed in linear time with the number of nodes times the average degree of the nodes. Experimental results show that our graph kernel is efficient and it performs better than a state-of-the-art graph kernel for benchmark data sets.
Keywords: graph kernel, hash, logical operation

1 はじめに

グラフ構造を持つデータからの学習やデータマイニングは重要な研究分野である．グラフは文字列や木などに比べて複雑な構造や関係を表現できる．例えば，計算機化学において化学物質のグラフ構造解析は必要不可欠である [10]．化学物質の性質はその構成原子の構造に強く依存するため，計算機上で分子構造をモデル化することで，大量の化学物質に対して高コストな手作業による実験を行わずに毒性などの性質を調べることができる．同様に，グラフはバイオインフォマティクスにおいて遺伝子ネットワークやタンパク質間相互作用ネットワーク，RNA の 2 次構造などとして現れる．さらに，ソーシャルネットワーク解析や Web マイニングなどの問題においても，ある種のグラフデータが主要な役割を果たしている．

グラフは自由度と表現力が高いため，一般に固定長のベクトルへ変換することは難しい．ところが，従来の機械学習アルゴリズムにおいてはデータが固定長のベクトルで表現されていると仮定されている．一方，サポートベクターマシン（SVM）に代表されるカーネル法においては，グラフ同士の特徴空間での内積の計算手法を与えることにより，グラフをベクトルへ変換することなく学習問題を解くことが可能である．よって Haussler が R-convolution カーネルを発表して以来 10 年，グラフカーネルを用いたグラフからの学習は大きな研究課題である [1, 5]．多くのグラフカーネルが提案されてきたが，それらは常にスケーラビリティと性能とのトレードオフに直面してきた．最も成功したグラフカーネルであるランダムウォークカーネルは，n をグラフ中のノード数として $O(n^3)$ の計算時間が必要である [4, 8, 11]．実際のところ，3 乗の計算量は依然として大きな負荷であり，ランダムウォークカーネルが適用可能なグラフのサイズは数百以下である．それゆえ，既存のグラフカーネルの応用範囲はこれまで主に小さな化学構造などのグラフデータに限られていた．

我々は本論文でラベル付きグラフに対し，近傍ハッシュカーネル（Neighborhood Hash Kernel）と呼ぶ新しいグラフカーネルを提案する．その鍵となるのは，ビット列として表現されたノードラベルと，それらの論理演算である．そのためにまず，変換関数によって各離散ノードラベルを固定長ビット列に変換する．例えば，ノードラベル A を 4 ビットのビットラベル #1000 と表す．重要なポイントは，ビットラベル集合は，適用順序に依存しない論理演算で扱えることである．あるノードに近接するノード集合のビットラベルに対する排他的論理和演算（XOR）は，そのノードに新しいラベルとしての固有のハッシュ値を与える．ノードラベルをそれら XOR 演算の値で更新することで，近接ノード集合の構造の情報を新しいラベルに集約することができる．更新ラベルを接続されたノード間で交換するために，この操作はグラフ中の全てのノードに対して数回適用され，それによって高次構造の特徴がグラフ中に伝播していく．そして最後に，更新ラベル集合同士が一致する割合によって，グラフ間のカーネル関数の値を計算する．近傍ハッシュカー

*IBM 東京基礎研究所, 〒 242-8501 神奈川県大和市下鶴間 1623-14, e-mail hido@jp.ibm.com,
IBM Research - Tokyo, 1623-14 Shimo-tsuruma, Yamato-shi, Kanagawa, 242-8501 Japan
†東京大学大学院情報理工学系研究科数理情報学専攻, 〒 113-8656 東京都文京区本郷 7-3-1, e-mail kashima@mist.i.u-tokyo.ac.jp,
Department of Mathemacical Informatics, the University of Tokyo, 7-3-1 Hongo, Bunkyo-ku, Tokyo, 113-8656 Japan

ネルの大きな優位性は計算がグラフサイズに対して線形時間と非常に高速に行えるスケーラビリティを備えると同時に，ノードの順列問題を回避しながら大きなグラフの高度な構造を扱えることである．

本論文は以下のように構成されている．最初に 2 節でラベル付きグラフを定義する．次に 3 節でグラフカーネルとその性質を説明する．そして 4 節で近傍ハッシュカーネルを提案する．まず 4.1 節でビットラベルと，それに対する論理演算を導入する．4.2 節では近傍ハッシュの計算方法を詳しく述べる．カーネル行列を求めるアルゴリズムは 4.3 節で説明する．5 節では計算量について考察する．最後に 6 節において既存カーネルとの比較によってスケーラビリティと性能の評価実験を行う．

2　ラベル付きグラフ

本論文で，我々はノードが離散ラベルで表現される属性を持ち，枝が無向であるグラフに重点を置く．ラベル付き無向グラフは，$G = \{V, E, \ell(\cdot)\}$ のように形式的に定義される．ここで，V はノード集合，$E = V \times V$ は枝集合を表す．各要素 $v \in \{v_1, \ldots, v_n\} = V$ はグラフ中の 1 つのノードを表す．ノードの数 $n = |V|$ をもって，グラフのサイズと定義する（$|G| = n$），ラベリング関数 $\ell(\cdot)$ はノードから離散ラベルの全集合であるアルファベット Σ へのマッピングを表す．各ノード v_i（$1 \leq i \leq n$）はラベル $\ell(v_i) \in \Sigma$ を持つ．各枝 e_{ij} は 2 つのノード v_i と v_j の間の無向枝である．ここで，枝が無向であるため，e_{ji} も必ず E に属する．枝集合 E は大きさ $n \times n$ の隣接行列 A としても表現できる．隣接行列の要素は，$e_{ij} \in E$ ならば $A_{ij} = A_{ji} = 1$，そうでなければ 0 である．$\mathrm{V}^{adj}(v_i)$ は，v_i に近接する（枝を共有する）ノードの集合を表し，そのサイズは $|\mathrm{V}^{adj}(v_i)| = d_i$ とする．同様に，r-近接ノード集合 $\mathrm{V}^{r}_{adj}(v_i)$ は最大 r 個の枝からなる経路で v_i に繋がるノードの集合とする．現実のグラフデータの多くが離散ラベル付き無向グラフで表現できるため，ほとんどの既存のグラフカーネルと同様，我々も離散ラベル付き無向グラフを扱う新たなカーネルを提案する．

3　グラフカーネル

本節で，カーネル法について簡単に紹介すると共に，どのようなグラフカーネルが実世界の応用において新たな可能性を見出せるかについて議論する．

サポートベクターマシン（SVM）が教師付き学習における強力なアプローチとして現れて以来，カーネル法に基づくアルゴリズムは深く研究されており，機械学習とデータマイニングにおけるおよそ全ての問題に対して適用され，成功を収めてきた．多くの学習問題は，カーネル法によって，最適解が反復法で効率的に求められる 2 次計画問題として解ける．グラフデータを扱う学習問題も同様にカーネル法の恩恵を受けられるはずだが，多くのカーネルは個々のデータが固定長の実数値ベクトルで表されていることを前提としている．実際のところ，グラフをその構造的な情報の欠損無しに固定長ベクトルへ変換することは一般に困難である．しかしながら，グラフカーネルの登場によって，グラフを高次元ベクトルとして陽に表現せずとも，SVM を含むカーネル法ベースの学習アルゴリズムをグラフデータに適用する道が開けた．グラフを含む構造を持つデータに対しては，Haussler が最初に全ての部分構造ペアを比較する R-convolution カーネルを考案した [5]．同じフレームワークを使って，特定の種類の部分構造，例えば経路や木，もしくは部分グラフを用いるグラフカーネルがいくつも提案されている [1].

次に，カーネルに関して，カーネル関数とカーネル行列，及びいくつかの重要な性質を説明する．2 つのグラフ G と G' の間のカーネル関数を $k(G, G')$ とする．G と G' が同一のグラフであることは，$k(G, G') = 1$ を満たす必要十分条件である．そうでなければ，カーネル関数の値は 0 以上 1 未満となる．グラフ集合 $\{G_1, \ldots, G_h\}$ とカーネル関数 $k(\cdot, \cdot)$ に対し，カーネル行列 K の各要素は以下のように計算される．

$$K_{ij} = K_{ji} = k(G_i, G_j) \ \ (1 \leq i, j \leq h).$$

半正定値性を満たすカーネルは Mercer カーネルと呼ばれ、SVM を含むカーネル法の凸最適化問題を解くための鍵となる．よって，カーネル行列は任意の複素数列 $c_i \in \mathbb{C} (1 \leq i \leq h)$ に対して以下の不等式を一般に満たす．

$$\sum_{i,j=1}^{h} c_i \bar{c_j} K_{ij} \geq 0.$$

カーネル行列の半正定値性は固有値が全て非負であるか否かで確かめることができる．また，カーネル関数は加法性を持つので 2 つのカーネル $k(\cdot, \cdot)$ と $k'(\cdot, \cdot)$ に対して，その線形和

$$k''(\cdot, \cdot) = \alpha k(\cdot, \cdot) + (1 - \alpha) k'(\cdot, \cdot) \tag{1}$$

もまたカーネルとなる．ここで，α は 0 以上 1 以下の実数である．この性質により，複数のカーネルの組み合わせによって，別のカーネルを定義できる．

次に，グラフカーネルの望ましい性質について説明する．

スケーラビリティ. 最も重要なポイントはカーネルの計算に必要な時間とメモリ量である．グラフは複雑な構造であるため，2 つのグラフを比較してその類似度や差分を何らかの値として求めることは困難かつ漠然としたタスクである．実際，単に 2 つのグラフが同一かどうかをチェックするグラフ同型性問題（graph isomorphism）においてすら，多項式時間アルゴリズムは知られていない．グラフ同型性問題を困難にしている大きな理由の 1 つは，ノード順列である．あるノードが別のグラフに出現しているか知るためには，近接ノード集合の埋め込みを，その順序に依らず調べなければならない．そのためにはさらに各近接ノードに近接する全ノード集合の埋め込みも考える必要がある．計算機上でノード間の接続は隣接行列の形で保持されているだけなので，これら埋め込みのチェックは数十ノードの小さなグラフに対してすら膨大な計算コストがかかる．実際，ランダムウォークを基にしたカーネルでも $O(n^3)$ の計算時間を要する．よって，サイズが $1,000$ を超えるグラフの詳細な比較は非現実的であり，既存のグラフカーネルでは扱うことができなかった．

表現力. グラフカーネルはまた，可能な限りグラフ構造に潜む隠れた重要な差異を扱う表現力を持たねばならない．単純に実行可能なグラフ比較方法は，ノードラベルの種類やヒストグラム，ノード次数の分布などの統計的性質を比べることである．しかし，そのようなナイーブな方法ではグラフの構造における特徴を考慮できないため，その応用範囲は限定的となってしまう．例えば，薬剤試験において，化学物質の毒性を予測するにはノードや枝レベルの統計的性質だけでなく，原子間の接続関係における微妙な差異を考慮する必要がある．一方，構造の性質を考慮しながらグラフを比較する単純な手法は，部分グラフ集合間のマッチングを行い類似度を計算する手法である．しかしながら，部分グラフの数はグラフサイズに対し指数的に増大するため，大きなグラフにおいて全部分グラフを列挙するのは非現実的である．

ここで，良いグラフカーネルに求められる性質をまとめる．

- 小さな計算コスト（$O(n^3)$ 未満）.
- 部分グラフ構造に対する高い表現力.

様々なグラフに対して多様なカーネルが提案されてきたが，それらは常にスケーラビリティと表現力の間の同じトレードオフに直面しており，その応用可能範囲は未だ限定的である．我々の知る限り，一般のラベル付きグラフに対して両方の性質を兼ね備えた効率よく性能の良いグラフカーネルはこれまで存在しなかった．我々が目指すのは，まさにそのようなカーネルである．

4 近傍ハッシュカーネル

本節で，グラフカーネルとして良い性質を備えた近傍ハッシュカーネルを提案する．鍵となるのは，ノードラベルを固定長のビット列（ビットラベル）として表現し，近接するノード集合のビットラベル間の論理演算によって各ノードを特徴付けることである．まず，どのようにビットラベル付きノードを扱うかを説明し，次にカーネルの計算アルゴリズムを導入する．

4.1 ビットラベルと論理演算

以下のように，D 個のビット（0 もしくは 1）から成るビット列をビットラベルと呼ぶ.

$$s = \{b_1, b_2, \cdots, b_D\}. \tag{2}$$

ここで，定数 D は $2^D - 1 \gg |\Sigma|$ を満たすようにする．ビットラベルは最大 $2^D - 1$ までの大きさの非負整数を表すことができる．ラベル付きグラフに対して，ノードラベル全集合 Σ は離散値の有限集合であり，グラフデータにおいて通常その種類数は千通り以下である．たとえば，化学物質の中には 118 の異なる原子のみが存在する．それゆえ，全てのノードラベルは対応するランダムに選ばれた D ビット列集合に一般性を失うことなく変換できる．実験においては，元のラベル集合とランダムな 16 ビット列への 1 対 1 のマッピング関数によって，全てのラベルを 16 ビットラベルで置き換えるとする．

次に，ビットラベルを扱ってハッシュ値を計算するために，符号理論で一般的によく用いられるビット演算を導入する．排他的論理和（Exclusive OR, XOR）は，2 ビット間の 2 値関数として定義される．ビット b_i とビット b_j の間の XOR は，$b_i \neq b_j$ の場合は 1，それでなければ 0 を返す.

ここで，$\mathrm{XOR}(s_i, s_j) = s_i \oplus s_j$ は 2 つのビットラベル s_i と s_j の間の XOR 演算とし，その出力である新たなビット列は各桁が s_i と s_j の各桁の XOR 操作の結果を表すとする．この操作はビット毎に行われるため，$s_i \oplus s_j$ の計算は最大でも D に線形の時間で実行可能である[1].

XOR 操作の特徴を表 1 にまとめた．表 1(a) 中の s_{zero} は空のビット列（$\{0, 0, \cdots, 0\}_{i=1}^{D}$）を表す．任意の 3 つのビット列 s，s'，s'' に対して，表 1(a-e) の性質が成り立つ.

[1] 一般的な CPU は，ビット長 D がプロセッサのビットサイズ（主に 32 か 64）を超えない場合，このようなビット毎の論理演算を 1 クロック（単位時間）で実行できる [7]．MATLAB においても，16 ビット列に対するビット毎の XOR 操作は 32 ビット列と同じ計算時間となる.

表 1: ビット毎 XOR 演算の性質

(a)	$s \oplus s = s_{zero}$
(b)	$s \oplus s_{zero} = s$
(c)	$s \oplus s' = s' \oplus s$
(d)	$(s \oplus s') \oplus s'' = s \oplus (s' \oplus s'')$
(e)	$s' \oplus s' \oplus s = s$

さらにもう 1 つのビット演算，ビット回転 (ROT) を導入する．ビット回転 ROT_o はビット列 $s = \{b_1, b_2, \cdots, b_D\}$ に対し，後半の $D-o$ ビットを o ビット左に移動させ，前半の o ビットをその後ろに移動させる．その結果は，次のようになる．

$$\mathrm{ROT}_o(s) = \{b_{o+1}, b_{o+2}, \cdots, b_D, b_1, \cdots, b_o\}. \quad (3)$$

ROT 演算もビットラベルを同じ長さの新たなビット列に変換する．さらに，ROT_o 操作も，o の値に関わらず，D に線形の時間で実行可能性である[2]．

4.2 近傍ハッシュ

ビットラベル付きグラフに対して，論理演算 XOR と ROT を用いて近傍ハッシュを計算する．

あるノード v に対して，まず近接ノード集合 $\mathrm{V}^{adj}(v)$ を得る．図 1 に，あるノードについて近接するノードとそのラベルが与えられた時の，近傍ハッシュ計算の例を示す．簡単のため，ここではノードラベルを 4 ビット列で表している．図 1(a) にあるように，ノード v の元のラベルと対応するビットラベルはそれぞれ A と #1000 である．このノードは 2 つの近接ノード $\{v_1^{adj}, v_2^{adj}\}$ を持ち，それらの元のラベルとビットラベルはそれぞれ B（#1110）と C（#1100）である．図 1(b) に計算方法を示した単純な近傍ハッシュは次の通り定義する．

Definition 1 Simple Neighborhood Hash: ノード v とその近接ノード集合 $\mathrm{V}^{adj}(node) = \{v_1^{adj}, \cdots, v_d^{adj}\}$ に対し，近傍ハッシュを以下の通り計算する．

$$\mathrm{NH}(v) = \mathrm{ROT}_1(\ell(v)) \oplus (\ell(v_1^{adj}) \oplus \cdots, \oplus \ell(v_d^{adj})).$$

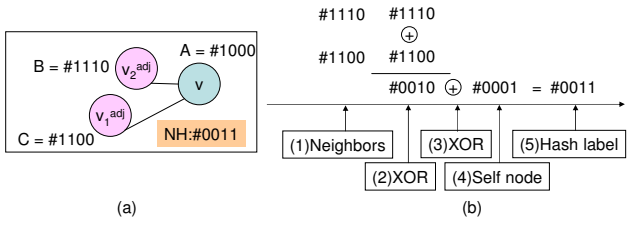

図 1: 単純な近傍ハッシュの例．(a) ノード v は 2 つの近接ノードを持つ．(b)XOR と ROT を用いた v の近傍ハッシュの計算手法.

ここで，出力 $\mathrm{NH}(v)$ もまた D ビット列である．図 1(b) のステップ (1)-(4) は 3 つのビットラベルに対する XOR 演算を表している．v 自身のビットラベルだけが 1 ビット回転（$\mathrm{ROT}_1(\ell(v))$）によって近接ノードのビットラベルとは区別されている．その後，ハッシュ値 $\mathrm{NH}(v) = \#0011$ を得て，v の新たなビットラベルと見なす．この値 $\mathrm{NH}(v)$ は v 周辺のノードラベルの分布（ヒストグラム）を一意に表現する．この 1 ノードに対するハッシュ演算は，d を v の近接ノード数とした時，$O(Dd)$ で実行可能である．

近傍ハッシュは（あるノードとその近接ノード集合の）ビットラベル集合から，同じ長さの新しいビット列へのハッシュ関数となっている．ここで，2 つのノード v_i と v_j は同じノードラベルを持つとする（$\ell(v_i) = \ell(v_j)$）．もし v_i と v_j の近接ノード集合のラベル集合も同一であった場合は，2 つのノードに対する近傍ハッシュのハッシュ値は同じになる．そうでなければ，ハッシュ衝突の場合を除いて，ハッシュ値は違うものとなる[3]．近傍ハッシュにとって重要なのは，XOR 演算の性質（表 1(c)）によって，近接ノード集合のノードラベルが処理される順序にハッシュ値が依存しないことである．これにより，我々は v_i と v_j に対して，近接ノードのラベル集合の共通部分のマッチング無しに，近傍ラベル集合が同一か否かを判断できる．これが計算量の面において，我々のグラフカーネルにとって主要な優位性となる．

ここまでで，我々はノード v のノードラベルを近傍ハッシュ値 $\mathrm{NH}(v)$ に置き換えるられるようになった．このラベル更新は各ノードの近接ノードの情報を統合したことになる．近傍ハッシュをグラフ G の全てのノードに適用してノードラベルを更新すると，新しいグラフ $G^1 = \{V, E, \ell^1(\cdot)\}$ が得られる．そこでは $\ell^1(v) = \mathrm{NH}(v)$ が全ての $v \in V$ について成り立つ．この操作を，グラフへの近傍ハッシュ関数（$G^1 = \mathrm{NH}(G)$）とする．更新されたグラフ G^1 において，各ノードラベル $\ell^1(v)$ は元のグラフ G において直接繋がっているノードのラベル集合に対応している．ここで，G^1 を保持するのに必要なメモリ量は G と同じである．

近傍ハッシュによって，離れたノード間の高次構造がどのように扱うかを説明する．$G^{r+1} = \mathrm{NH}(G^r)$ のように，近傍ハッシュは繰り返し適用できる．ここで，$\ell^{r+1}(v) = \mathrm{NH}(v^r)$ である．グラフ G^r における，更新されたノードラベル $\ell^r(v)$ は r-近傍ノード集合のラベル

[2]ROT 演算は MATLAB でサポートされている bitshift 操作を用いて実装する．

[3]D ビット列に対してハッシュ衝突が起こる確率はわずか 2^{-D} である（$D = 16$ ならば 10^{-5} 以下である）．

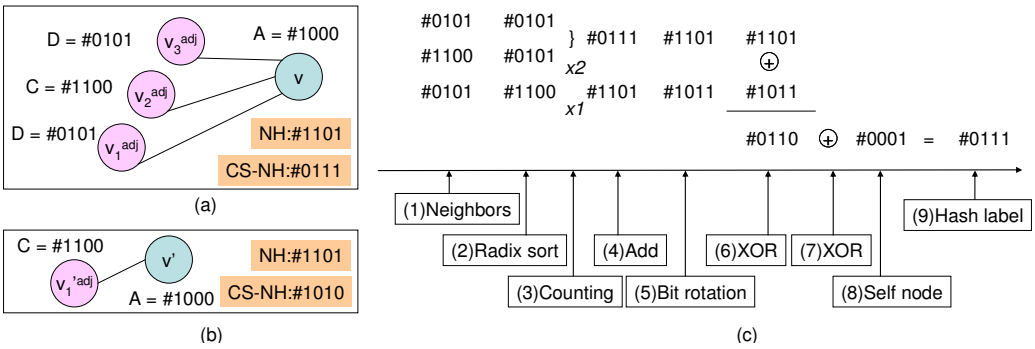

図 2: カウント考慮型近傍ハッシュの例．(a) ノード v とその近接ノード．(b) ノード v' は v と同じ近傍ハッシュ値を持つ．(c)v のカウント考慮型近傍ハッシュの計算手法.

分布を表している．それゆえ，近傍ハッシュが r 回適用された 2 つのグラフ中の 2 つのノード v_i と v_j は，その r-近傍ノード集合のラベルとその構造が等しい場合にのみ，同じノードラベルを持つ．

$$\ell^{r+1}(v_i) = \ell^{r+1}(v_j) \Rightarrow \mathrm{V}^r_{adj}(v_i) = \mathrm{V}^r_{adj}(v_j).$$

そうでなければ，異なるノードラベルとなる（ハッシュ衝突を除く）．近傍ハッシュは近接ノード集合内の順序に依存しないため，複雑なマッチング無しに，v_i と v_j の周辺ノード構造の一致を効率よく比較できる．

次に，ハッシュ衝突を削減するために，単純な近傍ハッシュの拡張を行う．実際，偶然によるハッシュ衝突でなくとも，2 つのノードに対する近傍ハッシュが同じ値となってしまう可能性がわずかながら存在する．例えば，図 2(a)(b) にあるように，ノード v と v' が同じノードラベル A（#1000）を持ち，近接ノードがラベル集合それぞれ｛D，C，D｝と｛C｝だとする．この場合，2 つの近傍ハッシュ値は同一になる（$\mathrm{NH}(v) = \mathrm{NH}(v') =$ #1101）．何故なら，図 2(a) に偶数個存在するノードラベル D（#0101）は XOR 演算において表 1 (e) の性質により打ち消されてしまうためである．このような都合の悪いハッシュ衝突は，近傍ハッシュを利用したカーネルの半正定値性とその性能に悪影響を与える可能性がある．

この問題に対処するため，ラベルカウントを用いた近傍ハッシュの拡張を行う．まず近接ノードのビットラベル集合に対しソートを適用する．これは，基数ソート（radix sort）等の比較無しソートアルゴリズムを用いることで，$O(Dd)$ で実行可能である．次に，含まれる全ラベルとその出現回数を線形時間で数え，重複無し近接ラベル集合 $\mathrm{V}'^{adj}(v)$ とその大きさ d' を得る．各ラベル $\ell(v^{adj})$ に対し，その出現回数 o（D ビット列で表現する）を用いて新しいラベルを次のように計算する．

$$\ell'(v^{adj}) = \mathrm{ROT}_o(\ell(v^{adj}) \oplus o).$$

ここでは，o との XOR 演算の後に o ビット回転が適用される．これらの処理により，ラベルの出現回数によって異なる近傍ハッシュ値が得られる．こうしてカウント考慮型近傍ハッシュを次のように定義する．

Definition 2 Count-sensitive Neighborhood Hash: ノード v とその重複無し近接ノード集合 $\mathrm{V}'^{adj}(v)$, に対し，カウント考慮型近傍ハッシュを以下の通り計算する．

$$\mathrm{CSNH}(v) = \mathrm{ROT}_1(\ell(v)) \oplus (\ell'(v_1^{adj}) \oplus, \cdots, \oplus \ell'(v_{d'}^{adj})).$$

この拡張された近傍ハッシュはハッシュ衝突を引き起こしていた図 2(a)(b) の v と v' に正しく異なる値を与える（$\mathrm{CSNH}(v) \neq \mathrm{CSNH}(v')$）．図 2(a) の v に対するカウント考慮型近傍ハッシュの計算の様子を図 2(c) に示す．入力ノードとラベル集合は単純な近傍ハッシュの場合と同じである．最初に近接ノードのラベル集合がステップ (1) でソートされる．ラベルの出現回数を順にカウントすることにより，2 種類のラベルが存在し，#0101 の出現が 2 回，#1100 の出現が 1 回であることがステップ (2)-(3) で分かる．そこでラベルの出現回数に応じ，ステップ (4)-(5) で XOR と ROT を適用して，#1101 と #1011 を求める．最後に，ステップ (6)-(9) においてもう一度 XOR を元のノード v との間で計算して，#0111 を結果として得る．一方で，v' に対し，カウント考慮型近傍ハッシュは図 2(b) の通り異なるハッシュ値 #0011 を与える．カウント考慮型近傍ハッシュの計算量は，依然として近接ノードの数 d（$d' \leq d$）に比例している．ここで，偶然のハッシュ衝突に対しては，十分に長いビットラベルを用いることによって，その確率を低減できる．我々の実装においては，16 ビットのビットラベルを用いる．

簡単に言えば，近傍ハッシュは各ノードラベルに，近接ノードのラベル集合という，より多くの情報量を含ま

せる操作である．近傍ハッシュを r 回適用されたグラフ G^{r+1} は，間に $r-1$ 個のエッジを持つノード間の高次の関係性も考慮した情報を持つ．また，更新されたグラフ集合を保持するのに必要なメモリ量は，元のグラフに比例した量でしかない．

4.3 アルゴリズム

近傍ハッシュを用いて更新されたグラフ集合を用いて，元のグラフ間の近傍ハッシュカーネル（Neighborhood Hash Kernel, NHK）を構築する．本節では，近傍ハッシュを用いてカーネル行列を計算するアルゴリズムを示す．

Algorithm 1 はグラフ集合を与えられてカーネル行列を計算する関数 Calculate_Similarity_Matrix の仮想コードである．近傍ハッシュを繰り返し適用し，グラフ間の高次構造を段階的に比較する．1-3 行目は，必要な入力である．各ステップ r（$1 \leq r \leq R$）において，5-13 行目の操作を繰り返す．5 行目で，部分カーネル行列 K^r を単位行列として初期化する．6-9 行目において，Γ の全グラフに近傍ハッシュを適用する．7 行目では，単純な近傍ハッシュとカウント考慮型近傍ハッシュのどちらでも選択可能である．同時に，各ノード集合のラベルを 8 行目でソートする．ノードラベルは固定長ビット列なので，基数ソート（radix sort）を用いることで D に比例する時間でソート可能である．古いグラフ G^r は以後使われないので，10 行目でそのデータをメモリ上から消去してもよい．11-13 行目で，全グラフペアに対して，ラベル集合間のマッチングによって行列の各要素を $K^r_{ij} = k(G_i, G_j)$ と計算する．ラベルマッチング関数 $Compare\_Labels$ の内容と，どのように K^r がカーネル行列になるかは後述する．上記処理を R 回繰り返した後，カーネルの加法性（式 1）に基づいて，最終的なカーネル行列 K は全部分カーネル行列 $\{K^r\}_{r=1}^R$ の平均として求める．

Algorithm 1 $Calculate\_Similarity\_Matrix$

Require: カーネル行列を計算するグラフ集合
1: $\Gamma = \{G_1^0, \ldots, G_h^0\}$: グラフ集合
2: $h = |\Gamma|$: グラフ数
3: R: 近傍ハッシュの最大適用数
Ensure: K 半正定カーネル行列
4: **for** $r = 1$ to R **do**
5: $K^r \leftarrow I$ {:h x h 単位行列 }
6: **for** $i = 1$ to h **do**
7: $G_i^r \leftarrow \mathrm{NH}(G_i^{r-1})$ {:単純かカウント考慮型 }
8: $G_i^r \leftarrow \{Radix\_Sort(V_i, \ell^r(\cdot)), E, \ell^r(\cdot)\}$
9: **end for**
10: remove G^{r-1}
11: **for** each pair $(G_i^r, G_j^r) \in \Gamma \times \Gamma$ **do**
12: $K^r_{ij} = K^r_{ij} \leftarrow Compare\_Labels(G_a, G_b)$
13: **end for**
14: **end for**
15: $K \leftarrow \frac{1}{R}\sum_{r=1}^{R} K^r$ {:繰り返しに対する正規化 }
16: **return** K {:カーネル行列 }

ラベルマッチング関数 $Compare\_Labels$ の仮想コードを Algorithm 2 に示す．2 つのグラフ G_a と G_b について，入力を 1-3 行目にまとめた．入力されたグラフのノードラベルリストはソート済みと仮定する．8 行目で，ノードリスト V_a^{sort} の i 番目の要素を選択し，もう一方のノードリスト V_b^{sort} 中のマッチングを順に数える．共通して存在するラベルのマッチング数を数えながら，インデックス i と j はそれぞれ n_a と n_b に達する．実際，7-19 行目の while ループは最大 $n_a + n_b$ 回の繰り返しで終了するため，このカウント処理は計算量 $O(D(n_a + n_b))$ で完了する．20 行目で，最終的なマッチング数 c を正規化し，カーネル行列の要素 (κ) として出力する．この式は離散集合間の類似度尺度としてよく用いられる Jaccard-Tanimoto 係数と同等のものである．

これらのアルゴリズムによって，近傍ハッシュを用いたカーネル行列は効率よく計算できる．

Algorithm 2 $Compare\_Labels$

Require: 比較する 2 つのグラフ
1: V_a^{sort}, V_b^{sort} :両グラフのソート済みノードリスト
2: $n_a = |V_a^{sort}|, n_b = |V_b^{sort}|$: 両グラフのノード数
3: ℓ_a, ℓ_b :両グラフのラベリング関数
Ensure: $0 \leq \kappa \leq 1$
4: $c \leftarrow 0$
5: $i \leftarrow 1$
6: $j \leftarrow 1$
7: **while** $i \leq n_a$ and $j \leq n_b$ **do**
8: $v_i \leftarrow V_a^{sort}[i]$
9: $v_j \leftarrow V_b^{sort}[j]$
10: **if** $\ell_a(v_i) = \ell_b(v_j)$ **then**
11: $c \leftarrow c + 1$
12: $i \leftarrow i + 1$
13: $j \leftarrow j + 1$
14: **else if** $\ell_a(v_i) < \ell_b(v_j)$ **then**
15: $i \leftarrow i + 1$
16: **else**
17: $j \leftarrow j + 1$
18: **end if**
19: **end while**
20: $\kappa = \frac{c}{n_a + n_b - c}$
21: **return** κ

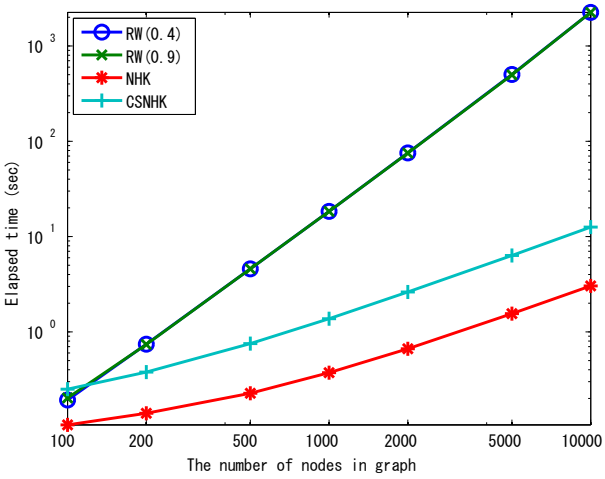

図 3: 各アルゴリズムの計算時間．近傍ハッシュカーネルはグラフサイズに対して線形の増加である．

5 考察

5.1 計算量

近傍ハッシュカーネルのスケーラビリティを考察する．

まず単純な近傍ハッシュとカウント考慮型近傍ハッシュ（NH と CSNH）の計算について考える．

4.2 節で示したように，あるノードに対する近傍ハッシュでは d 個の近接ノードを扱い，$d+1$ 個のビットラベルに対して XOR や ROT の論理演算を適用する．4.1 節で説明した通り，全ての論理演算は固定ビット長 D に比例する時間で計算できる．図 2(c) のステップ (2) にあるように，CSNH の中のソートも，基数ソートを用いることにより，高々ノード次数 d に線形の時間で計算可能である（$O(Dd)$）．ラベル種類数のカウントやカウントの追加など，図 2 のステップ (2-3) にあるその他の操作もまた $O(Dd)$ で実行できる．よって，グラフ G 中の全てのノードに対して操作を行うと，近傍ハッシュの計算量は NH と CSNH の両方に対して $O(D\bar{d}n)$ となる．ここで，$\bar{d}$ は平均ノード次数（近接ノード数）である．Algorithm 1 の 8 行目のソートもまたノード数に線形で計算できる．Algorithm 2 において 2 つのグラフを比較する時間も，2 つのグラフの最大サイズで抑えられる．それゆえ，近傍ハッシュカーネルの計算全体の計算量も $O(DR\bar{d}n)$ となる．また，計算に必要なメモリ用も最大グラフサイズとグラフ数の積に線形となる．

6 実験

本節で，近傍ハッシュカーネルの性能とスケーラビリティを評価する．

単純な近傍ハッシュカーネル（NHK）とカウント考慮型近傍ハッシュカーネル（CSNHK）の両方を MATLAB で実装した．ビットラベルの長さは 16 に固定し，近傍ハッシュの最大回数は 1 から 5 まで変化させる．比較のため，定点反復法を用いる効率的なランダムウォークカーネル（RW）も実装した [11]．ランダムウォークカーネルの対角要素は 1 ではないため，正規化を行って $K'_{ij} = K_{ij}/\sqrt{K_{ii} \cdot K_{jj}}$ としたカーネル行列 K' を計算する．ランダムウォークカーネルの停止確率 λ は $\{0.4, 0.9\}$ のいずれかとし，収束チェックの閾値は 10^{-10} とした．

表 2: 各データセットに関する性質．計算時間は RW（$\lambda = 0.4$）と，NHK，CSNHK（共に $R = 5$），全て Morgan インデックスを適用済みの値である．

			計算時間（秒）		
データ	#Graph	Max. size	RW	NHK	CSNHK
MUTAG	188	28	145.2	38.9	41.3
PTC MR	344	109	896.3	133.7	137.6

全ての実験は Intel Xeon® 3.16GHz，メインメモリ 12GB を搭載した Windows サーバ上で実行した．

6.1 人工グラフデータ

まず人工グラフを用いて近傍ハッシュカーネルのスケーラビリティを評価する．生成する人工グラフは 100 から $10,000$ までのサイズで，平均ノード次数 $\bar{d} = 5$，アルファベットの大きさ $|\Sigma| = 100$ とした．2 つの人工グラフに対して，カーネル行列（2×2 の大きさ）の計算に要した時間を CPU 時間で計測した．各パラメータ設定について 10 回ずつ計測を行った後，その平均時間を示す．図 3 は NHK，CSNHK，RW に対してグラフの大きさを変化させた時の計算時間の増加を示した両対数プロットである．

全体として，NHK と CSNHK は RW よりも一桁高速である．特に，サイズ $10,000$ のグラフに対しても NHK が要する時間はわずか 5 秒以下である．この結果は，近傍ハッシュカーネルを用いた学習が数千ノードを含むグラフデータセットに対しても適用であることを示す．一方，ランダムウォークカーネルはパラメータ λ の値に寄らず，グラフサイズに 2 乗の計算時間の増大が起きている．

6.2 ベンチマークデータ

グラフカーネルの性能評価に広く用いられている化学構造グラフの 2 値分類データセット，MUTAG と PTC Mice and Rats（MR）を用いて実験を行った [3, 6]．各データセットのグラフ数（#Graph）と最大グラフサイズ（Max. size）を表 2 に示した．また，元のデータセット

表 3: MUTAG と PTC MR における予測精度．2 行目の値は RW のパラメータ λ，NHK と CSNHK のパラメータ R.

		RW		NHK					CSNHK				
データ	Morgan	0.4	0.9	1	2	3	4	5	1	2	3	4	5
MUTAG	Off	74.94	74.50	81.84	79.15	78.74	80.85	81.90	81.84	81.37	86.17	**87.75**	**87.75**
	On	82.43	76.05	84.53	84.01	82.37	82.89	82.92	84.53	84.56	84.50	84.53	**85.06**
PTC MR	Off	59.87	58.11	60.77	59.61	59.59	59.00	57.84	60.77	**62.21**	61.05	60.48	60.77
	On	55.86	52.33	59.03	59.61	59.29	60.17	58.71	59.03	**61.01**	58.72	59.29	59.29

に対して，Morgan インデックス [9] を付与した．構築したカーネル行列を用いて予測モデルを学習するために，LIBSVM [2] の中の C-SVM を用いた．データセットを 10 個の部分セットに分割し，各評価セットに対して残りの 9 個の部分セットにおいて 10 段階交差検定を行い，パラメータ c を $\{10^0, 10^1, \cdots, 10^7\}$ から選択した．その後，選択された最適な c を用いて 9 個の部分セットにおいて学習を行い，評価セットにおける予測精度を求めた．

10 セットの平均精度を表 3 に示した．太字の数字は各設定（行）において最良の結果を表す．全体として，NHK と CSNHK は RW に比べて良い性能を発揮した．パラメータ R の影響は NHK については明らかではないが，$R = 5$ の CSNHK が Morgan インデックスの有無に関わらず最良の結果を残した．これはカウント考慮型近傍ハッシュが MUTAG データにおいて高次構造を正しく考慮することに成功したことを示している．MUTAG データの化学構造には数種類の原子しか含まれないため，Morgan インデックスは全てのアルゴリズムについて有効に働いている．PTC MR データセットにおいては，正しい予測が極めて難しいことが知られている．その中で，CSNHK は RW と NHK よりも高い性能を示した．最大ハッシュ回数 R の影響は不明だが，$R = 2$ の時の CSNHK が最良だった．これは，近傍ハッシュカーネルにおいて，最適な R の選択がデータセットに依存することを示している．ただし，計算時間は R に対して高々線形である．また，Morgan インデックスは PTC MR データセットにおいては上手く機能しない．

表 2 にアルゴリズム毎の最大計算時間を示した．実際，NHK と CSNHK はこれらの最大グラフサイズ 100 程度の比較的小さなグラフから成るデータセットに対しても 3 倍以上高速に動作した．

7 むすび

本論文において，高速な近傍ハッシュカーネルを提案した．その鍵は，ラベルをビット列で表現し，近接ノード集合のラベル分布を論理演算で比較することである．その結果，カーネル関数の計算量はわずか $O(D\bar{d}n)$ である．実験において，近傍ハッシュカーネルは数千ノードのグラフに適用可能であることが示され，ベンチマークにおいて既存カーネルよりも良い性能を示した．

参考文献

[1] K. M. Borgwardt. *Graph Kernels*. PhD thesis, Computer Science, Ludwig-Maximilians-University Munich, 2007.

[2] C.-C. Chang and C.-J. Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[3] A. Debnath, R. Lopez de Compadre, G. Debnath, A. Shusterman, and C. Hansch. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of Medicinal Chemistry*, 34:786–797, 1991.

[4] T. Gärtner, J. W. Lloyd, and P. A. Flach. Kernels and distances for structured data. *Machine Learning*, 57(3):205–232, 2004.

[5] D. Haussler. Convolution kernels on discrete structures. Technical report, 1999.

[6] C. Helma and S. Kramer. A survey of the predictive toxicology challenge 2000-2001. *Bioinformatics*, 19(10):1179–1182, 2003.

[7] Intel Corporation. *Intel 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z*, March 2009.

[8] H. Kashima, K. Tsuda, and A. Inokuchi. Marginalized kernels between labeled graphs. In *Proceedings of the 20th International Conference on Machine Learning (ICML)*, 2003.

[9] P. Mahe and T. Akutsu. Extensions of marginalized graph kernels. In *In Proceedings of the 21st International Conference on Machine Learning (ICML)*, pages 552–559. ACM Press, 2004.

[10] L. Ralaivola, S. J. Swamidass, H. Saigo, and P. Baldi. Graph kernels for chemical informatics. *Neural Networks*, 18(8):1093–1110, 2005.

[11] S. Vishwanathan, K. M. Borgwardt, and N. N. Schraudolph. Fast computation of graph kernels. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 1449–1456. MIT Press, Cambridge, MA, 2007.