

# 枝刈り探索によるソーシャルネットワークでの影響最大化アルゴリズム

大坂 直人<sup>†</sup> 秋葉 拓哉<sup>†</sup> 吉田 悠一<sup>††</sup> 河原林健一<sup>††</sup>

<sup>†</sup> 東京大学大学院情報理工学系研究科コンピュータ科学専攻 〒113-8654 東京都文京区本郷 7-3-1

<sup>††</sup> 国立情報学研究所 〒101-8430 東京都千代田区一ツ橋 2-1-2

E-mail: <sup>†</sup>{ohsaka,t.akiba}@is.s.u-tokyo.ac.jp, <sup>††</sup>{yyoshida,k.keniti}@nii.ac.jp

**あらまし** バイラルマーケティングは「口コミ」を通して商品の購入を効率的に促すマーケティング戦略である。この考えにもとづく影響最大化問題は、影響力の高い少数の頂点集合をソーシャルネットワーク上から選択する問題である。NP-hardに属するため現実的な時間で厳密解を計算することは望めないが、近似比  $1 - 1/e$  を保証する貪欲アルゴリズムが知られている。Monte-Carlo シミュレーションによる頂点集合の影響力の計算に膨大なコストがかかるため、このアルゴリズムを大規模なネットワークに適用することは困難である。我々は、枝刈り探索及び理論的解析にもとづく Monte-Carlo シミュレーションの回数の削減による影響最大化問題の高速アルゴリズムを提案する。実験により、提案手法が既存手法と同等以上の速度を保ちつつ、ほとんど常に最も影響力の高い頂点集合を選択可能であることを示した。

**キーワード** ソーシャルネットワーク, 影響最大化問題, バイラルマーケティング

## 1. はじめに

バイラルマーケティング [4], [11] とは、商品の無料サンプルや割引商品を影響力の高い少数のグループ（シードと呼ばれる）に与えることで、多数の人へ商品購入を効率的に促すマーケティング戦略である。広告による直接的なマーケティングとは異なり、商品の評判は「口コミ」を通して人から人へ間接的に広げられる。このような情報拡散は、ソーシャルネットワーク上（頂点為人、辺が人と人との関係に対応する）でモデル化される。バイラルマーケティング成功のためには、ソーシャルネットワークを解析し、少数で効果的なシード集合を見つけることが求められる。

「影響最大化問題」は、影響力が最大のシード集合を選ぶ問題であり、確率的伝搬モデルによる数学的定式化がなされている [8]。NP-hard に属する [8] ため、多くの近似アルゴリズムやヒューリスティクスが提案されてきたが、計算時間と精度のトレードオフはいまだ解決されていない。

Kempe ら [8] の Monte-Carlo シミュレーションにもとづく貪欲アルゴリズムは影響最大化問題の最初の手法である。目的関数が劣モジュラ性を有するため、貪欲アルゴリズムが最適解の定数近似解を出力することが理論的に保証される。さらに、実際のネットワーク上ではほぼ最適な解が得られることが実験的に示されている。しかしながら、貪欲アルゴリズムはスケラビリティに乏しく、様々な高速化手法 [2], [3], [9], [12], [14] をもってしても、数百万頂点規模のグラフに適用することが困難であった。

その結果として、様々なヒューリスティクスが提案されてきた [1], [2], [5], [6], [7], [13]。これらの手法はシード集合の影響力を Monte-Carlo シミュレーションではなく高速なヒューリスティクスで近似するため、解の精度に関する理論的保証が無い。したがって、貪欲アルゴリズムよりも高いスケラビリティを持

つ一方、得られる解の質がネットワーク構造やパラメータ設定に大きく依存するという欠点がある。

以上の問題を解決するため、本論文は影響最大化問題に対する効率的かつ高精度なアルゴリズムを提案する。ヒューリスティクスにもとづく手法とは異なり、提案手法は貪欲アルゴリズムにもとづきシード集合の影響力を Monte-Carlo シミュレーションによって計算するため、解の精度が理論的に保証される。大規模なネットワーク上での動作を実現するため、提案手法に以下の 3 つの高速化手法を導入する。(1) Monte-Carlo シミュレーションにより得られたランダムグラフの保持・更新によるシミュレーション回数の抑制及びその理論的保証、(2) 影響力評価に伴う到達可能性検査のための幅優先探索 (BFS) の効率的な枝刈り、(3) 不必要な影響力評価の検知及び除去。

計算機実験により、(1) 提案手法の与える解が既存手法よりもほとんど常に優れていること、(2) 数千万辺規模の実ソーシャルネットワークに対して既存のヒューリスティクスと同等の速度で動作することを示した。

## 2. 既存研究

### 2.1 Monte-Carlo シミュレーションにもとづく手法

Monte-Carlo シミュレーションにもとづく手法は、情報拡散をシミュレートすることで頂点集合の影響力を高い精度で得られるため、影響力の高いシード集合を出力する。一方で、これらが大規模なネットワークに直接適用することは計算時間の観点から困難である。

Kempe ら [8] は影響最大化問題に対する Monte-Carlo シミュレーションにもとづく貪欲アルゴリズムを提案した。影響力関数が劣モジュラ性を有するため、貪欲アルゴリズムがほぼ最適な解を与えることが理論的に保証される。しかし、情報拡散のシミュレーションコストが膨大なため、スケラビリティに乏しいという欠点を持つ。Leskovec ら [9] が提案した

Cost-Effective Lazy Forward (CELF) は、劣モジュラ性を利用することで、Kempe らの貪欲アルゴリズムと同じ性能を保ちつつ、必要な影響力評価の回数を抑える。実験より、貪欲アルゴリズムと比較して 700 倍の高速化が示されたが、数万頂点のグラフに対しては依然として数時間以上を要する。Cheng ら [3] が提案した StaticGreedy は、入力グラフより生成したランダムグラフを再利用することで、解の質を保ちつつ、Monte-Carlo シミュレーションの回数を 10,000 から 100 に抑えられることを実験的に示した。しかしながら、実行時間がパラメータ設定に大きく依存することが我々の実験の結果より示された。

## 2.2 ヒューリスティクスにもとづく手法

Monte-Carlo シミュレーションの使用を避けるべく、様々なヒューリスティクスが提案された。

Jiang ら [6] が提案した Simulated Annealing with Effective Diffusion Values (SAEDV) は、シード集合の近傍の情報のみから影響力を近似し、焼きなまし法により高影響力の頂点集合を探索する。Chen ら [1] が提案した PMIA は、伝搬確率の高い経路を探索し、影響力を近似した。Jung ら [7] が提案した Influence Rank Influence Estimation (IRIE) は各頂点の影響力を連立方程式で表現し効率的に計算する。

これらのヒューリスティクスは Monte-Carlo シミュレーションにもとづく手法よりも高いスケーラビリティを持つが、解の質がネットワーク構造やパラメータ設定に大きく依存する。

## 3. 予備知識

### 3.1 論文中で用いる表記

$G = (V, E)$  をサイズ  $n$  の頂点集合  $V$  とサイズ  $m$  の頂点集合  $E$  からなる有向グラフとする。  $u \rightsquigarrow v$  は、グラフ  $G$  上で頂点  $u$  が頂点  $v$  に到達可能であることを意味する。すなわち、 $G$  において  $u$  から  $v$  への経路が存在する。

### 3.2 影響最大化問題の定式化

本論文では、最もスタンダードな情報拡散モデルである独立カスケードモデル [8] を採用する。独立カスケードモデルは、有向グラフ  $G = (V, E)$  と伝播確率関数  $p: E \rightarrow [0, 1]$ 、シードと呼ばれる頂点集合  $S \subseteq V$  を入力とし、以下の確率過程を繰り返す：

- (1)  $S$  に含まれる頂点の状態をアクティブに設定し、 $V \setminus S$  に含まれる頂点の状態を非アクティブに設定する。
- (2) 新しくアクティブになった各頂点  $u$  は、非アクティブな  $u$  の各近傍  $v$  を確率  $p_{uv}$  でアクティブに変更する。
- (3) 新しくアクティブになった頂点が存在すれば (2) へ戻り、そうでなければ終了する。

独立カスケードモデルにおけるシード集合  $S$  の「影響拡散」を「独立カスケードモデルにおける  $S$  をシードとする情報拡散において、アクティブになる頂点の数の期待値」で定義する。以後、 $S$  の影響拡散を  $\sigma(S)$  と表記する。影響最大化問題 [8] は、独立カスケードモデルにおいて影響拡散を最大化する大きさ  $k$  の頂点集合を求める問題である。

### 3.3 貪欲アルゴリズム

影響最大化問題は NP-hard に属する [8] ため、多項式時間の

---

### Algorithm 1 独立カスケードモデルにおける影響最大化問題の貪欲アルゴリズム [8]

---

```

1:  $S \leftarrow \emptyset$ 
2: while  $|S| < k$  do
3:    $t \leftarrow \arg \max_{v \in V \setminus S} \sigma(S \cup \{v\}) - \sigma(S)$ 
4:    $S \leftarrow S \cup \{t\}$ 
5: return  $S$ 

```

---

厳密解法は望めないが、影響拡散関数  $\sigma(\cdot)$  の有する性質を利用した貪欲アルゴリズムが最適解の定数近似解を出力することが知られている。集合関数  $f: 2^V \rightarrow \mathbb{R}$  は、任意の  $S \subseteq V$  について  $f(S) \geq 0$  ならば非負であるといい、任意の  $S \subseteq T \subseteq V$  について  $f(S) \leq f(T)$  ならば単調であるといい、任意の  $S \subseteq T \subseteq V$  と  $v \in V$  について  $f(S \cup \{v\}) - f(S) \geq f(T \cup \{v\}) - f(T)$  ならば劣モジュラであるという。Kempe ら [8] による影響最大化問題の貪欲アルゴリズムを Algorithm 1 に示す。貪欲アルゴリズムは、空のシード集合  $S = \emptyset$  から開始し、影響拡散の増加量が最大の頂点  $t = \arg \max_v \sigma(S \cup \{v\}) - \sigma(S)$  を  $S$  に追加していく。劣モジュラ関数最大化問題に対する貪欲アルゴリズムについて Theorem 3.1 が成立し、独立カスケードモデル上の  $\sigma(\cdot)$  について Theorem 3.2 が成立する。

**Theorem 3.1.** [10] 非負かつ単調な劣モジュラ関数  $f$  について、 $S$  を貪欲アルゴリズムにより得られた集合、 $S^*$  を  $f$  の最大化問題の最適解とすると、 $f(S) \geq (1 - 1/e)f(S^*)$  が成立する。

**Theorem 3.2.** [8] 独立カスケードモデルにおいて影響拡散関数  $\sigma(\cdot)$  は非負、単調、劣モジュラである。

したがって、貪欲アルゴリズムは、影響最大化問題の最適解の約 63% 近似解を出力する。

### 3.4 影響拡散の計算方法

$\sigma(\cdot)$  の厳密計算は #P-hard に属する [1] ため、既存研究の多くでは Monte-Carlo シミュレーションが用いられている [2], [3], [8], [9], [12], [14]。シミュレーション回数を  $R$  とすると、 $\sigma(\cdot)$  の時間計算量は  $O(mR)$  となる。ゆえに、貪欲アルゴリズムの総時間計算量は  $O(kmR)$  となる。さらに、高い精度で  $\sigma(\cdot)$  の値を得るためには、膨大な回数のシミュレーションを必要とする（典型的には  $R \approx 10,000$ ）。以上より、貪欲アルゴリズムは大規模なグラフにスケールしない。この問題を解決するため、次節で影響最大化問題の高速アルゴリズムを提案する。

## 4. 提案手法

本節では、独立カスケードモデルにおける影響最大化問題の高速アルゴリズムを提案する。以下では、まず、「コインフリップテクニック」を説明し、独立カスケードモデルのシミュレーションがランダムグラフ上での到達可能性判定と等価であることを示す。次に、高速化手法を含まない提案手法を説明し、最後に、2つの高速化手法を導入する。

### 4.1 コインフリップテクニック

Kempe ら [8] による「コインフリップテクニック」を説明する。シード集合を  $S$  として独立カスケードモデルをシミュレー

---

**Algorithm 2** 独立カスケードモデル下の影響最大化問題の高速アルゴリズム
 

---

**Require:**  $G = (V, E), p: E \rightarrow [0, 1], k, R$  (# of DAGs)

```

1: for  $i = 1$  to  $R$  do
2:    $E'_i \leftarrow$  各辺  $e$  を確率  $p_e$  で残すことで得られる辺集合
3:    $G'_i = (V, E'_i)$  の強連結成分を計算
4:    $G_i = (V_i, E_i) \leftarrow G'_i$  より構築された頂点重み付き DAG
5:    $h_i \leftarrow V_i$  において最大の次数をもつ頂点
6:    $D_i \leftarrow \{v \in V_i \mid h_i \overset{G_i}{\rightsquigarrow} v\}$ 
7:    $A_i \leftarrow \{v \in V_i \mid v \overset{G_i}{\rightsquigarrow} h_i\} \setminus \{h_i\}$ 
8:    $\text{latest}_i[v] \leftarrow \text{false}$  for all  $v \in V_i$ 
9:  $S \leftarrow \emptyset$ 
10: while  $|S| < k$  do
11:    $t \leftarrow \arg \max_{v \in V} \frac{1}{R} \sum_{i=1}^R \text{GAIN}(i, v)$ 
12:    $S \leftarrow S \cup \{t\}$ 
13:   for  $i = 1$  to  $R$  do
14:      $\text{UPDATEDAG}(i, t)$ 
15: return  $S$ 

```

---

トした結果、アクティブになる頂点集合の分布を  $\mathcal{D}_G(S)$  と定義する。各辺  $e \in E$  を確率  $p_e$  で残すことで得られるランダムグラフを  $G_p$  とし、 $G_p$  において頂点集合  $S$  から到達可能な頂点集合の分布を  $\mathcal{D}'_G(S)$  と定義する。ここで、2つの分布  $\mathcal{D}_G(S)$  と  $\mathcal{D}'_G(S)$  は等しい [8]。したがって、以後はランダムグラフ上での到達可能性のみを考える。

#### 4.2 提案手法のあらまし

提案手法は次の5ステップから構成される：(1) 入力から複数のランダムグラフを生成し、(2) 各ランダムグラフから頂点重み付き非循環有向グラフ (DAG) を構築し、(3) 各頂点の影響拡散の増加量を、各 DAG においてその頂点から到達可能な頂点の総重みの平均で近似し、(4) 影響拡散の増加量が最大の頂点をシードとして選択し、(5) 各 DAG を更新する。

Algorithm 2 に提案手法の擬似コードを示す。  $R$  は生成する DAG の個数を表すパラメータである。提案手法は2つのパートに分かれる。前半のパートでは、 $R$  個の頂点重み付き DAG  $G_1, G_2, \dots, G_R$  を生成する。  $i$  個目の頂点重み付き DAG  $G_i$  の構築方法は次の通り。まず、各辺  $e \in E$  を確率  $p_e$  で残した辺集合  $E'_i$  を得る。次に、 $G'_i = (V, E'_i)$  の強連結成分分解を行い、 $v \in V$  を含む強連結成分  $\text{comp}_i[v]$ 、強連結成分  $v$  に含まれる頂点の数  $\text{weight}_i[v]$  を計算する。最後に頂点重み付き DAG  $G_i = (V_i, E_i)$  を次のように構築する： $V_i = \{\text{comp}_i[v] \mid v \in V\}$ ,  $E_i = \{(\text{comp}_i[u], \text{comp}_i[v]) \mid uv \in E'_i\}$ 。ここで、ランダムグラフ  $G'_i$  において頂点集合  $S$  から到達可能な頂点数と頂点重み付き DAG  $G_i$  において  $\{\text{comp}_i[t] \mid t \in S\}$  から到達可能な頂点の総重みは等しい。以下、この値を  $\sigma_{G_i}(S)$  と表記する。すなわち、

$$\begin{aligned} \sigma_{G_i}(S) &= \{v \in V \mid \exists t \in S, t \overset{G_i}{\rightsquigarrow} v\} \\ &= \sum_{v \in V_i: \exists t \in S, \text{comp}_i[t] \overset{G_i}{\rightsquigarrow} v} \text{weight}_i[v]. \end{aligned}$$

また、 $\sigma_{G_i}(S \cup \{v\}) - \sigma_{G_i}(S)$  の値を  $G_i$  における  $S$  に関する

$v$  の「利得」と呼び  $\sigma_{G_i}(v \mid S)$  と表記する。  $G_i$  の構築後、次数最大の頂点  $h_i$  を選択し、 $h_i$  の子孫からなる集合  $D_i$  と  $h_i$  の先祖からなる集合  $A_i$  を計算する。また、利得の再計算フラグを表す変数  $\text{latest}_i[v]$  を初期化する。これらは後述の高速化手法に用いられる。

後半のパートでは、利得の近似値を計算し、シード集合を選択する。以後、 $j$  番目のシードを選択するフェーズを「 $j$  番目のフェーズ」と呼ぶ。関数  $\text{GAIN}(i, v)$  は、 $\text{comp}_i[v]$  を始点とする BFS を行い、 $G_i$  における  $v$  の利得を返す。 $v$  の影響拡散の増加量  $\sigma(S \cup \{v\}) - \sigma(S)$  の近似値を各 DAG の  $\sigma_{G_i}(v \mid S)$  の平均値で計算し、影響拡散の増加量が最大の頂点  $t$  をシードとして選択する。関数  $\text{UPDATEDAG}(i, t)$  は  $\text{comp}_i[t]$  から到達可能な頂点を  $G_i$  から除く。したがって、 $|S| + 1$  番目のフェーズでは、 $\text{comp}_i[v]$  から到達可能な頂点の重みの合計値は新しい利得  $\sigma_{G_i}(v \mid S \cup \{t\})$  に一致する。

BFS がおよそ  $knR$  回行われるため、Algorithm 2 の時間計算量は貪欲アルゴリズムから本質的には改善されていない。提案手法を高速にするため、2つの高速化手法を提案する。1つ目は BFS 内で探索される頂点の数を抑える「枝刈り BFS」、2つ目は「不必要な利得の再計算を避けることによる高速化手法」である。これらの高速化手法は影響拡散の近似値に影響を与えないため、提案手法の精度は保たれる。

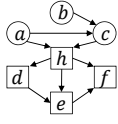
#### 4.3 高速化手法 1：枝刈り BFS

BFS 内で探索される頂点の数を抑える方法を議論する。まず、全ての頂点から BFS を行うときの時間計算量を考える。各 BFS で訪れられる頂点数の総和は、任意の頂点  $h$  に対して ( $h$  の先祖の数)  $\times$  ( $h$  の子孫の数) 以上である。この値は頂点数の自乗オーダーになりうるため、BFS 全体のコストは大きい。枝刈り BFS のアイデアは次の通り：

頂点  $v$  が頂点  $h$  に到達可能ならば、 $v$  から BFS を行う際に  $h$  から到達可能な頂点を枝刈りすることができる。

図 1 を例に枝刈り BFS を説明する。頂点  $h$  は最大の次数を有するため、図のグラフにおいてハブであるとみなせる。また、 $h$  の先祖と子孫はそれぞれ  $\{a, b, c\}$  と  $\{h, d, e, f\}$  である。頂点  $a$  から普通の BFS を行うと  $\{a, c, h, d, e, f\}$  を訪れる。しかし、( $h_i$  の先祖・子孫を前計算すれば)  $a$  が  $h$  を訪れることは既知であり、したがって、 $h$  の子孫を訪れることが分かるため、この BFS は冗長である。枝刈り BFS では、 $\{h, d, e, f\}$  を枝刈りすることで  $\{a, c\}$  のみを訪れる。同様に、 $b$  からは、普通の BFS を行うと  $\{b, c, h, d, e, f\}$  を訪れるが枝刈り BFS では  $\{b, c\}$  のみを訪れる。

枝刈り BFS を Algorithm 3 に示す。頂点  $v$  から BFS を開始する前に、 $v$  が  $h_i$  の先祖であるかを調べる。もし  $v$  が  $h_i$  の先祖であれば、 $h_i$  から到達可能な頂点を除いたとみなした枝刈り BFS を行い、そうでなければ、通常の BFS を行う。枝刈り BFS を行った場合の頂点  $v$  の利得は、 $h_i$  の利得と枝刈り BFS で訪れた頂点の総重みの和に厳密に一致する。2番目のフェーズ以降に行われる BFS の回数が次に提案する高速化手法によ



Vertex	# of vertices visited during	
	normal BFS	pruned BFS
a	6	2
b	6	2
c	5	1

図1 枝刈り BFS の例

**Algorithm 3** 高速化手法 1: 枝刈り BFS による到達可能な頂点の重みの合計値の計算

```

1: function GAIN( $i, v_V \in V$ )
2:    $v \leftarrow \text{comp}_i[v_V]$ 
3:   return 0 if  $v \notin V_i$ 
4:   return  $\delta_i[v]$  if latest $_i[v]$ 
5:   latest $_i[v] \leftarrow \text{true}$ 
6:   if  $v \in A_i \wedge |S| = 0$  then
7:      $\delta_i[v] \leftarrow \text{GAIN}(i, h_V)$  s.t.  $\text{comp}_i[h_V] = h_i$ 
8:   else
9:      $\delta_i[v] \leftarrow 0$ 
10:   $Q \leftarrow$  頂点  $v$  のみを有するキュー
11:   $X \leftarrow \{v\}$ 
12:  while  $Q \neq \emptyset$  do
13:    頂点  $u$  をキュー  $Q$  からデック
14:    if  $v \in A_i \wedge u \in D_i \wedge |S| = 0$  then
15:      continue
16:     $\delta_i[v] \leftarrow \delta_i[v] + \text{weight}_i[u]$ 
17:    for all  $uw \in E_i$  do
18:      if  $w \notin X \wedge w \in V_i$  then
19:        頂点  $w$  をキュー  $Q$  にエンキュー
20:       $X \leftarrow X \cup \{w\}$ 
21:  return  $\delta_i[v]$ 

```

**Algorithm 4** 高速化手法 2: 利得再計算の抑制

```

1: function UPDATEDAG( $i, t_V \in V$ )
2:    $t \leftarrow \text{comp}_i[t_V]$ 
3:   for all  $v \in V_i : \exists u, t \overset{G_i}{\rightsquigarrow} u \wedge v \overset{G_i}{\rightsquigarrow} u$  do
4:     latest $_i[v] \leftarrow \text{false}$ 
5:    $V_i \leftarrow V_i \setminus \{v \mid t \overset{G_i}{\rightsquigarrow} v\}$ 

```

り減らされるため、枝刈り BFS は最初のフェーズにおいてのみ行われる。

#### 4.4 高速化手法 2: 利得再計算の抑制

利得の再計算を抑制する方法を議論する。新しいシード頂点  $t$  をシード集合に追加後に頂点  $v$  の利得が変化する必要条件は「 $v$  から到達可能なある頂点が  $t$  から到達可能」であるため、この条件を満たさない頂点については利得の再計算を行わない。

Algorithm 4 に利得再計算の抑制手法を示す。まず、新しいシード  $t$  を始点とする BFS を行い、次に、 $t$  から到達可能な頂点から逆方向 BFS を行う。逆方向 BFS で訪れた各頂点  $v$  について利得の再計算フラグを設定し (latest $_i[v]$  を false に設定する)、最後に  $t$  の子孫を  $G_i$  から取り除く。

### 5. 理論的解析

本節では、提案手法の Monte-Carlo シミュレーションの回数 (=生成する DAG の数) が Kempe ら [8] の貪欲アルゴリズムと

比較して理論的に抑えられることを示す。

以後、2 つのグラフサンプリングにもとづく手法を考える。1 つ目は、事前に生成したサンプルを全体のフェーズを通して再利用する「再利用アルゴリズム」(提案手法に対応する)、2 つ目は、各フェーズでグラフをサンプリングし直す「再標本アルゴリズム」である。本節では、再利用アルゴリズムが再標本アルゴリズムよりもサンプルの数において効率的であることを示す。

まず、Hoeffding の不等式を導入する。

**Theorem 5.1** (Hoeffding の不等式).  $X_1, \dots, X_n$  を  $[0, 1]$  の範囲の値を取る独立確率変数とする。  $\bar{X} = \frac{1}{n} \sum_{i=1}^n X_i$  と定義すると

$$\Pr[|\bar{X} - X| > t] \leq 2 \exp(-2nt^2).$$

**Lemma 5.2.**  $G$  をグラフ、 $S$  を頂点集合族とする。  $R = O(\frac{1}{2} \log |S| \log \frac{1}{\delta})$  とし、  $G_1, \dots, G_R$  を  $G$  からサンプルしたグラフの集合とする。  $\bar{\sigma}(S) = \frac{1}{R} \sum_{i=1}^R \sigma_{G_i}(S)$  とすると、  $1 - \delta$  以上の確率で、任意の  $S \in \mathcal{S}$  に対して  $\bar{\sigma}(S) = \sigma_G(S) \pm \epsilon n$  が成立する。

*Proof.*  $\sigma_{G_i}(S) \in [0, n]$  であるから、頂点集合  $S \in \mathcal{S}$  について、  $\frac{1}{n} \sigma_{G_1}(S), \dots, \frac{1}{n} \sigma_{G_R}(S)$  に Hoeffding の不等式を適用すると、次が得られる:  $\Pr[|\bar{\sigma}(S) - \sigma(S)| > \epsilon n] \leq 2 \exp(-2\epsilon^2 R)$ .  $S$  内の全集合について union bound をとると、全ての  $S \in \mathcal{S}$  について  $|\bar{\sigma}(S) - \sigma(S)| \leq \epsilon n$  が成立する確率は  $1 - 2 \exp(-2\epsilon^2 R)|\mathcal{S}|$  以上である。  $R$  の値を  $R = O(\frac{1}{2} \log |S| \log \frac{1}{\delta})$  とすることで、所望の確率が得られる。  $\square$

次に、再利用アルゴリズムと再標本アルゴリズムが必要とするサンプルの数を解析する。

$G$  を  $n$  頂点グラフとし、 $k$  をシード集合のサイズとする。  $i \in \{1, \dots, k\}$  について  $S_i$  を  $i$  番目のフェーズで選ばれたシード頂点集合とする。  $i \in \{0, \dots, k-1\}$  について  $i$  番目のフェーズにおいて  $\sigma(\cdot)$  の値を計算すべき頂点集合の族を  $\mathcal{S}_i$  とする。すなわち、  $\mathcal{S}_{i+1} = \{S_i \cup \{v\} \mid v\}$ .  $\mathcal{S} = \cup_{i=0}^{k-1} \mathcal{S}_{i+1}$  とする。  $\mathcal{S}_i$  ( $i = 1, \dots, k-1$ ) 及び  $\mathcal{S}$  は確率変数である。

再利用アルゴリズムは、まずグラフ  $G_1, \dots, G_R$  をサンプリングし、各頂点集合  $S$  について、  $\sigma_G(S)$  を  $\bar{\sigma}(S) := \frac{1}{R} \sum_{i=1}^R \sigma_{G_i}(S)$  で見積もる。以下の定理が成り立つ。

**Theorem 5.3.**  $R = O(\frac{\log k + \log n}{\epsilon^2} \log \frac{1}{\delta})$  と設定する。  $1 - \delta$  以上の確率で、再利用アルゴリズムによる任意の頂点集合  $S \in \mathcal{S}$  の  $\sigma_G(S)$  の見積もりは、  $\bar{\sigma}(S) = \sigma_G(S) \pm \epsilon n$  を満たす。

*Proof.*  $|\mathcal{S}| \leq kn$  であるから、定理は Lemma 5.2 から直ちに得られる。  $\square$

再標本アルゴリズムは、 $i$  番目のフェーズにおいて、  $G_1^i, \dots, G_R^i$  をサンプリングし、各頂点集合  $S$  について  $\sigma_G(S)$  の値を  $\bar{\sigma}^i(S) := \frac{1}{R} \sum_{j=1}^R \sigma_{G_j^i}(S)$  で見積もる。

**Theorem 5.4.**  $R = O(\frac{\log n}{\epsilon^2} \log \frac{1}{\delta})$  とすると、  $1 - \delta$  以上の確率で、任意の頂点集合  $S \in \mathcal{S}_i$ 、インデックス  $i$  について再標本アルゴリズムの見積もりは  $\bar{\sigma}^i(S) = \sigma_G(S) \pm \epsilon n$  を満たす。

データセット	$ V $	$ E $	辺の向き
Epinions	76K	509K	有向
DBLP	655K	2.0M	無向
LiveJournal	4.8M	69M	有向
Twitter30days	6.2M	62M	有向

*Proof.*  $|S_i| \leq n$  であるから、定理は Lemma 5.2 から直ちに得られる。□

再標本アルゴリズムはサンプル数  $O(\frac{k \log n}{\epsilon^2} \log \frac{1}{\delta})$  を必要とする一方、再利用アルゴリズムは、サンプル数  $O(\frac{\log k + \log n}{\epsilon^2} \log \frac{1}{\delta})$  で十分であるため、より効率的である。

## 6. 計算機実験

複数のソーシャルネットワークに対する計算機実験を行い、提案手法と既存手法との性能比較を行う。

### 6.1 実験設定

#### 6.1.1 データセット

表1に示す有向グラフ3つと無向グラフ（双方向グラフとみなす）1つを用いる。各ネットワークの詳細を以下に示す。

**Epinions**<sup>(注1)</sup>: Epinions.com (www.epinions.com) から得られたネットワークである。各頂点はユーザーに対応し、各辺はユーザー間の信頼関係に対応する。

**DBLP**<sup>(注2)</sup>: Michael Ley による DBLP Computer Science Bibliography Database から得られた共著ネットワークである。

**LiveJournal**<sup>(注1)</sup>: LiveJournal (www.livejournal.com) のソーシャルネットワークである。各頂点はユーザーに対応し、各辺はユーザー間の信頼関係に対応する。

**Twitter30days**: Twitter (twitter.com) より抽出した2013年4月分の日本語のツイート情報からなるグラフである。各頂点はユーザーに対応し、各辺はユーザーからユーザーへのリプライに対応する。

#### 6.1.2 伝播確率

各辺の伝播確率はパラメータ  $P$  に設定する。様々な伝播確率に対するアルゴリズムの振舞を調査するため、 $P$  の値は  $P = 0.01, 0.025, 0.05, 0.075, 0.1$  とする。

#### 6.1.3 アルゴリズム

アルゴリズムの一覧を以下に示す。

**Ours**( $R$ ): 提案手法。DAG の数  $R$  の値は200に設定する。

**StaticGreedy**(200) [3]: 入力グラフからランダムグラフを生成し、影響拡散を計算する。ランダムグラフの数は200に設定する。

**PMIA** [1]: 最も伝搬確率の高い経路を探し、影響拡散の値を計算する。パラメータ  $\theta$  の値は [1] と同様に  $1/320$  に設定する。

**IRIE** [7]: 連立方程式で近似した影響拡散を効率的に計算する。パラメータ  $\alpha$  と  $\theta$  の値は [7] と同様に  $0.7$  と  $1/320$  にそれぞれ設定する。停止条件も [7] と同様に設定する。

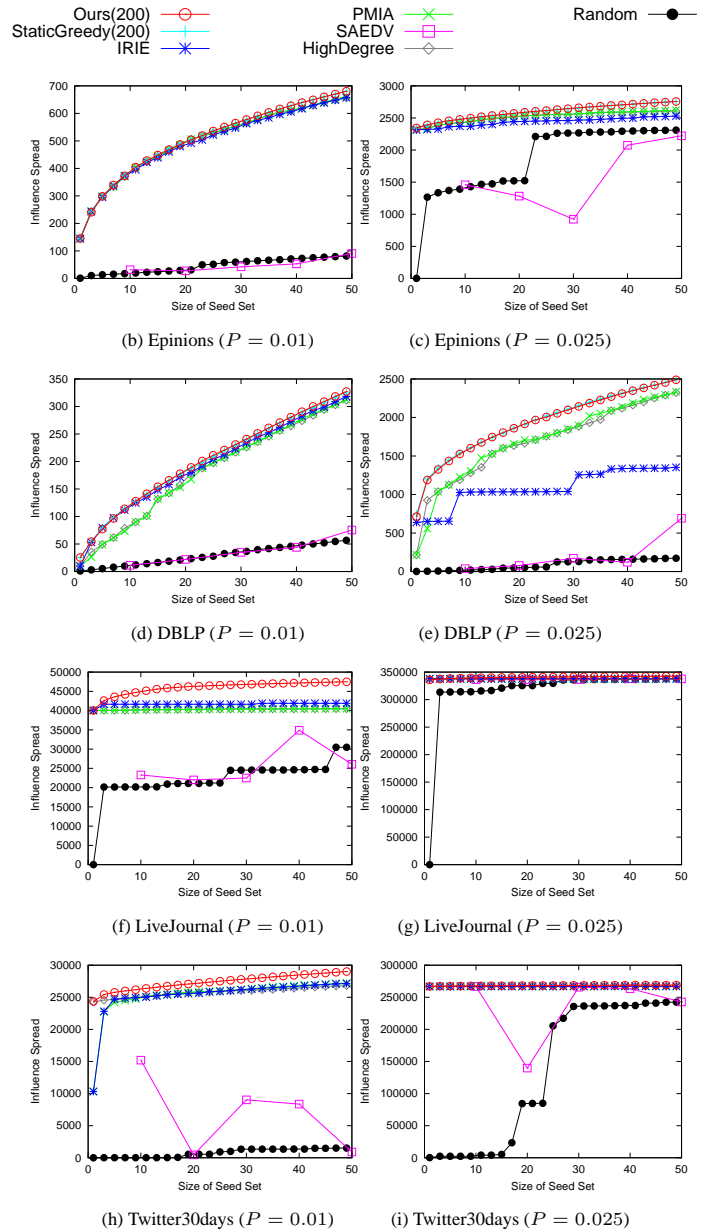


図2 設定ごとの影響拡散の値の比較

**SAEDV** [6]: 焼きなまし法によるアルゴリズムである。各パラメータの値は [6] と同様に設定する。

**HighDegree**:  $k$  頂点を度数順に選ぶ。

**Random**:  $k$  頂点をランダムに選ぶ。

#### 6.1.4 実験環境

C++で実装された各アルゴリズムをLinuxサーバー（CPU: Intel Xeon X5670 (2.93 GHz), メモリ: 48GB）で実行した。

## 6.2 結果・考察

### 6.2.1 影響拡散の比較

図2に設定ごとに各アルゴリズムが出力したシード集合の影響拡散を示す。シード集合のサイズ  $k$  の値は1から50まで設定する。SAEDVは貪欲法にもとづいていないため、 $k$  の値を10, 20, 30, 40, 50に設定する。PMIAはLiveJournal ( $P \geq 0.075$ ) においてメモリ不足により停止し、StaticGreedy(200)はEpinions ( $P \geq 0.05$ ), DBLP ( $P \geq 0.05$ ), LiveJournal ( $P \geq 0.01$ ), Twitter30days ( $P \geq 0.01$ ) においてメモリ不足により停止したた

(注1): <http://snap.stanford.edu/data/> より入手可能

(注2): <http://research.microsoft.com/en-us/people/weic/> より入手可能

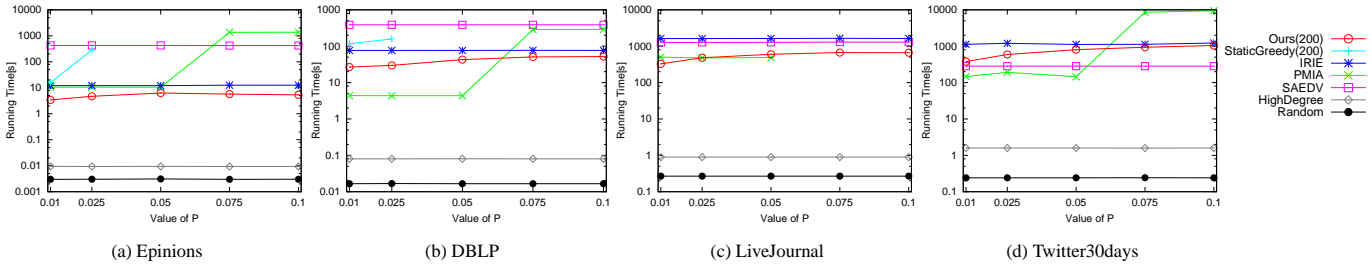


図3 設定ごとの実行時間の比較

め、これらの設定の結果は得られなかった。得られたシード集合の影響拡散の値を得るため、10,000回のMonte-Carloシミュレーションを行い、平均値を影響拡散の値とした。アルゴリズム間の影響拡散の差が比較的小さいため、 $P = 0.05, 0.075, 0.1$ についての結果は省略する。

提案手法 Ours(200) は全ての設定においてほぼ最良の解を出力している。StaticGreedy(200) もほぼ同様の結果を示した。ヒューリスティクスにもとづく手法 (PMIA, IRIE, SAEDV) は LiveJournal ( $P = 0.01$ ) と Twitter30days ( $P = 0.01$ ) において、Ours(200) よりもそれぞれ 13%・7%以上低い影響拡散の値を示した。特に、DBLP ( $P = 0.025, k = 50$ ) における IRIE の影響拡散 (= 1354) は Ours(200) の影響拡散 (= 2510) のおよそ半分であり、DBLP ( $P = 0.025, k = 1$ ) における PMIA の影響拡散 (= 216) は Ours(200) の影響拡散 (= 713) の三分の一以下であった。SAEDV はほぼランダムなシード集合を返していると思われる。明らかに、Random と HighDegree の計算した解は芳しくない。結果より、ヒューリスティクスにもとづく既存手法はグラフや確率の設定に性能が影響される一方、提案手法はほとんど常に最良の解を出力することが示された。

### 6.2.2 実行時間の比較

図3に各アルゴリズムの計算時間 ( $k = 50$ ) を示す。実行時間にはグラフの読込時間は含まれていない。PMIA と StaticGreedy(200) は、上述の設定においてメモリ不足により停止したため、実行時間が得られなかった。最も速い手法は Random、2番目に速い手法は HighDegree である。StaticGreedy(200) は多くの設定において1時間以上を要した後、メモリ不足により動作を停止した。PMIA の実行時間は  $P$  の値が 0.075 を超えると爆発的に増加している。IRIE と SAEDV は  $P$  の値による実行時間の変化が見られなかった。Ours(200) は既存手法と比較して最速ではないが、IRIE と SAEDV とほぼ同速であり、StaticGreedy(200) と PMIA に比較してロバストな性能を示した。

以上の結果より、提案手法の性能は様々なグラフ・確率の設定に対して既存手法より頑健でありながら、ほぼ最も影響拡散の高い解を出力すると結論づけられる。

## 7. おわりに

本論文では、独立カスケードモデル上の影響最大化問題に対する効率的かつ高精度な手法を提案した。ネットワークに存在するハブを利用することで、提案手法は幅優先探索の大幅な高速化を可能とした。さらに、高い精度のシード集合を選ぶた

めに必要な Monte-Carlo シミュレーションの回数が既存の貪欲アルゴリズムと比較して抑えられる理論的根拠を示した。実験結果より、提案手法は数千万辺を持つグラフに対して既存のヒューリスティクスと遜色ない速度で動作し、様々な設定下において既存手法と比較してほとんど常に最良の解を与えることを示した。

## 文献

- [1] Wei Chen, Chi Wang, and Yajun Wang. Scalable Influence Maximization for Prevalent Viral Marketing in Large-Scale Social Networks. In *KDD*, pages 1029–1038, 2010.
- [2] Wei Chen, Yajun Wang, and Siyu Yang. Efficient Influence Maximization in Social Networks. In *KDD*, pages 199–208, 2009.
- [3] Suqi Cheng, Huawei Shen, Junming Huang, Guoqing Zhang, and Xueqi Cheng. StaticGreedy: Solving the Scalability-Accuracy Dilemma in Influence Maximization. In *CIKM*, pages 509–518, 2013.
- [4] P. Domingos and M. Richardson. Mining the Network Value of Customers. In *KDD*, pages 57–66, 2001.
- [5] Amit Goyal, Wei Lu, and Laks V. S. Lakshmanan. SIMPATH: An Efficient Algorithm for Influence Maximization under the Linear Threshold Model. In *ICDM*, pages 211–220, 2011.
- [6] Qingye Jiang, Guojie Song, Gao Cong, Yu Wang, Wenjun Si, and Kunqing Xie. Simulated Annealing Based Influence Maximization in Social Networks. In *AAAI*, pages 127–132, 2011.
- [7] Kyomin Jung, Wooram Heo, and Wei Chen. IRIE: Scalable and Robust Influence Maximization in Social Networks. In *ICDM*, pages 918–923, 2012.
- [8] D. Kempe, J. Kleinberg, and É. Tardos. Maximizing the Spread of Influence through a Social Network. In *KDD*, pages 137–146, 2003.
- [9] Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne VanBriesen, and Natalie Glance. Cost-effective Outbreak Detection in Networks. In *KDD*, pages 420–429, 2007.
- [10] G. Nemhauser, L. Wolsey, and M. Fisher. An analysis of the approximations for maximizing submodular set functions. *Mathematical Programming*, 14:265–294, 1978.
- [11] M. Richardson and P. Domingos. Mining Knowledge-Sharing Sites for Viral Marketing. In *KDD*, pages 61–70, 2002.
- [12] Yu Wang, Gao Cong, Guojie Song, and Kunqing Xie. Community-based Greedy Algorithm for Mining Top-K Influential Nodes in Mobile Social Networks. In *KDD*, pages 1039–1048, 2010.
- [13] Hwanjo Yu, Seung-Keol Kim, and Jinha Kim. Scalable and Parallelizable Processing of Influence Maximization for Large-scale Social Networks. In *ICDE*, pages 266–277, 2013.
- [14] Chuan Zhou, Peng Zhang, Jing Guo, Xingquan Zhu, and Li Guo. UBLF: An Upper Bound Based Approach to Discover Influential Nodes in Social Networks. In *ICDM*, pages 907–916, 2013.