情報処理学会論文誌 Vol.59 No.7 1415–1424 (July 2018)

最大辺重みクリーク問題に対する局所探索法のためのデータ構造

清水 悟司[1,a)] 石原 諒大[1,b)] 山口 一章[1,c)] 増田 澄男[1,d)]

受付日 2017年11月10日, 採録日 2018年4月4日

概要：辺に重みが付与された無向グラフが与えられたとき，重みの和が最大となるクリークを求める問題を最大辺重みクリーク問題という．本稿ではこの問題に対する局所探索法において近傍を管理するためのデータ構造を 2 つ提案する．計算機実験により 2 つの局所探索法でデータ構造を比較し，提案法は従来法より性能が良いことを確認した．また，グラフの性質や計算機のメモリに応じて提案するデータ構造を使い分けることで，効率良く探索を行えることを確認した．

キーワード：最大辺重みクリーク問題，NP 困難，局所探索法，データ構造

Data Structures for Local Search Algorithms for the Maximum Edge-weight Clique Problem

Satoshi Shimizu[1,a)] Ryota Ishihara[1,b)] Kazuaki Yamaguchi[1,c)] Sumio Masuda[1,d)]

Received: November 10, 2017, Accepted: April 4, 2018

Abstract: The maximum edge-weight clique problem is to find the clique of maximum weight for a given edge-weighted undirected graph. In this paper, we propose two data structures to manage neighborhood in local search algorithms. By some computational experiments, we compared our proposal data structures with two local search algorithms and confirmed ours work better than a previous one. By applying proposal data structures properly, we confirmed local search algorithms work efficiently.

Keywords: maximum edge-weight clique problem, NP-hard, local search, data structure

1. はじめに

無向グラフ $G = (V, E)$ において，$C \subseteq V$ の任意の 2 頂点間に辺が存在するとき，C はクリークと呼ばれる．グラフ $G = (V, E)$ が与えられたとき，要素数最大のクリークを求める問題を最大クリーク問題（MCP）と呼ぶ．頂点に重みが付与されたグラフが与えられたとき，重みの和が最大のクリークを求める問題を最大重みクリーク問題（MWCP）と呼ぶ．辺に重みが付与されたグラフが与えられたとき，重みの和が最大のクリークを求める問題を最大辺重みクリーク問題（MEWCP）と呼ぶ．MCP やその一般化である MWCP や MEWCP は NP 困難である [1]．

MCP，MWCP および MEWCP の応用として，符号理論 [2]，ネットワークデザイン [3]，コンピュータビジョン [4]，バイオインフォマティクス [5], [6], [7]，オークション [8]，マーケットバスケット解析 [9]，コミュニケーション解析 [10], [11] などがあり，辺へ重みを付与することでより多くの情報が得られる場合もある．

MEWCP の厳密解を求める方法として数理計画問題による定式化を利用した方法 [12], [13] が提案されているものの，扱うことのできるグラフの頂点数は数十から数百程度である．一方，実応用では厳密な最適解が得られる保証がなくても，短い時間で良質な解が得られる手法が求められる．MCP や MWCP に対しては，局所探索に基づく解

[1] 神戸大学大学院工学研究科
Graduate School of Engineering, Kobe University, Kobe, Hyogo 657–8501, Japan
[a)] ss81054@gmail.com
[b)] 164t202t@stu.kobe-u.ac.jp
[c)] ky@kobe-u.ac.jp
[d)] masuda@kobe-u.ac.jp

法がいくつか提案されている [14], [15], [16], [17]．局所探索法では解を近傍へと遷移させる操作を繰り返すことによって解の探索を行うが，近傍を効率良く計算する方法としては，グラフが隣接リストで与えられた場合のデータ構造が提案されている [17]．MEWCP に対する解法としては MCP に対する手法 [14] を基にした，PLS（Phased Local Search）が提案されている [18]．

本稿では，MEWCP に対する局所探索法における近傍管理のためのデータ構造を 2 つ提案する．それぞれ，グラフが隣接リストで与えられた場合と，グラフが隣接行列で与えられた場合に用いることができる．MEWCP では辺重みがあるために，MWCP と比べて解の重みを計算するために時間がかかるが，提案する近傍管理手法を用いると重み計算の時間計算量を小さくすることができる．

計算機実験により，MEWCP に対する局所探索法に対し，提案する 2 つのデータ構造および従来のデータ構造のすべての組合せを比較した．実験結果から，提案したデータ構造の有効性を確認した．また，グラフの頂点数や辺密度，計算機のメモリ容量によって提案したデータ構造を使い分けることで効率良く探索が行えることを確認した．

本稿の構成は以下のとおりである．2 章で記号の定義や従来法について述べる．3 章で提案法について述べる．4 章で計算機実験の方法と結果について述べる．5 章でまとめと今後の課題について述べる．

2. 準備

2.1 記号の定義

単純な無向グラフ $G = (V, E)$ の各辺 $(u, v) \in E$ の重みを $w(u, v)$ と記す．クリーク $C \subseteq V$ に対し，$W(C) = \sum_{u,v \in C} w(u, v)$ と定める．辺密度を $d = \frac{2|E|}{|V|(|V|-1)}$ とする．頂点 v に隣接する頂点の集合を $N(v)$ と表す．クリーク C に対し，C_0 を「C の全頂点と隣接している頂点」の集合と定める．すなわち，C_0 は以下を満たす集合である：

$$C_0 = \{v \mid C \subseteq N(v)\}.$$

ここで任意の $v \in C$ は v 自身と非隣接であるため C_0 には含まれない．クリーク C に対し C_1 を「C の中に非隣接頂点がちょうど 1 つだけ存在し，かつ C に含まれない頂点」の集合と定める．すなわち C_1 は以下を満たす集合である：

$$C_1 = \{v \notin C \mid |C \setminus N(v)| = 1\}.$$

文献 [15], [17], [18] などと同様，クリーク C に対し，以下の 3 つの近傍を定義する．

Add 近傍：$N_{add}(C) = \{C \cup \{v\} \mid v \in C_0\}$ なる集合族．すなわち，C に頂点 $v \in C_0$ を加えてできるクリークの集合．

Drop 近傍：$N_{drop}(C) = \{C \setminus \{v\} \mid v \in C\}$ なる集合族．すなわち，C から頂点を 1 つ除いてできるクリークの集合．

Swap 近傍：$N_{swap}(C) = \{(C \cap N(v)) \cup \{v\} \mid v \in C_1\}$ なる集合族．すなわち，頂点 $v \in C_1$ を 1 つ選び，C から v に隣接しない頂点を取り除いたあと，頂点 v を加えてできるクリークの集合．C_1 の定義より，$|C \setminus N(v)| = 1$ であるため，Swap 近傍のクリークの要素数と C の要素数は等しくなる．

局所探索法では，クリーク C をこれらの近傍へと遷移させる操作を繰り返し行い，重みの大きいクリークの探索を行う．

2.2 従来法

局所探索を行うとき，近傍の走査は頻繁に行われる．以下では，グラフが隣接リストで表現されているときに近傍を管理・更新するためのデータ構造 [17] の概略を示す．各頂点 v に対し，クリーク C に含まれる v の隣接点の数を $\kappa(v, C)$ と定める．すなわち，$\kappa(v, C)$ は以下の等式を満たす：

$$\kappa(v, C) = |C \cap N(v)|.$$

また，クリーク C に対し，頂点集合 $N(C)$ を「クリーク内に少なくとも 1 つの隣接頂点が存在するような頂点の集合から，クリークに含まれる頂点を除いたもの」と定める．すなわち，$N(C)$ は以下の集合となる：

$$N(C) = \{v \notin C \mid \exists u \in C, v \in N(u)\}.$$

$\kappa(v, C)$ および $N(C)$ を用いて，頂点集合 S_{add} および S_{swap} を定義する：

$$S_{add} = \{u \in N(C) \mid \kappa(u, C) = |C|\},$$
$$S_{swap} = \{u \in N(C) \mid \kappa(u, C) = |C| - 1\}.$$

以上の定義より，$|C| > 0$ のとき $C_0 = S_{add}$ となり，$|C| > 1$ のとき $C_1 = S_{swap}$ となるため，文献 [17] では，$\kappa(v, C)$，$N(C)$，S_{add}，S_{swap} を用いて C_0，C_1 を管理している．文献 [17] では $|C| \leq 1$ の場合は考慮されていないが，$|C| \leq 1$ となるのはグラフの頂点数または辺数が極端に小さい場合のみである．

以降，集合への要素の追加削除の時間計算量は $O(1)$ と仮定する．そのような集合を実装する方法としては，追加削除の時間計算量の期待値が $O(1)$ のハッシュテーブルや，「双方向リストで集合を実装し，各頂点の所在を定数時間で参照できるようアドレスを格納した配列を持つ」などの方法が考えられる．

$\kappa(v, C)$, $N(C)$, S_{add}, S_{swap} は，クリーク C が近傍へと遷移するたびに更新される．クリークに追加または削除された頂点を v とすると，各 $u \in N(v)$ に対し，$\kappa(u, C)$ を更新する必要があり，そのための時間計算量は $O(|N(v)|)$ である．$\kappa(u, C)$ を更新した後，各 $u \in N(v)$ に対し，$\kappa(u, C) > 0$ かつ $u \notin C$ であれば u を $N(C)$ に加える．$N(C)$ 更新の時間計算量も $\kappa(u, C)$ の更新と同様，$O(|N(v)|)$ である．最後に，$N(C)$ を走査し，S_{add}，S_{swap} を $O(|N(C)|)$ の時間

計算量で更新する．

以上より，文献 [17] におけるクリーク C が近傍へ遷移する際の更新にかかる時間計算量は $O(|N(v)|+|N(C)|)$ である．

3. 提案法

本章では，近傍管理のためのデータ構造を 2 つ提案する．一般に頂点数が多く辺疎なグラフに対してはグラフを表現するデータ構造として空間計算量の小さい隣接リストが用いられる．提案法 L はそのような場合に効率良く近傍を管理するための手法である．一方，提案法 M は頂点数が少なく辺密なグラフに対し，グラフの表現に隣接行列を用い，非隣接リストを併用することで計算量を抑える手法である．各手法の概要を**表 1** に示す．

また，MEWCP では特に隣接リストでグラフが表現されている場合に，辺重みの参照に時間がかかるため，近傍の重みの計算に時間がかかってしまう．そこで，近傍の重みを効率良く計算するためのデータ構造も提案法 L とあわせて提案する．

3.1 提案法 L

クリーク C，各頂点 v に対し，文献 [17] と同様に $\kappa(v,C) = |C \cap N(v)|$ と定める．提案法 L では $\kappa(v,C)$ を用いて集合 $S(i,C)$ を以下のように定義する：

$$S(i,C) = \{v \in V \setminus C \mid \kappa(v,C) = i\}.$$

以降，特に必要のない限り $\kappa(v,C)$ を $\kappa(v)$ と記し，$S(i,C)$ を $S(i)$ と記す．$C_0 = S(|C|)$ となり，$C_1 = S(|C|-1)$ となる．提案法 L では，双方向リストで実装された $S(\cdot)$ を管理・更新することによって近傍を管理する．任意の頂点 v に対し，$\kappa(v) \le |N(v)|$ が成り立つため，提案法 L は $S(0), S(1), \ldots, S(\Delta)$ の $\Delta+1$ 個の双方向リストを管理する．ただし，Δ は全頂点のうち最大の次数である．双方向リストに追加される頂点はすべての $S(\cdot)$ で合わせて最大 $|V|$ 個であるため，これらの双方向リストすべての空間計算量は $O(|V|)$ である．初期状態（$C = \emptyset$）では，全頂点 $v \in V$ に対し $\kappa(v) = 0$ となり，すべての頂点が $S(0)$ に属する．このとき $S(1), S(2), \ldots, S(\Delta)$ は空の双方向リスト（Head 要素のみを持つ）である．従来法で用いられる集合 S_{add}，S_{swap} と提案法 L で用いられる集合 $S(i)$ の，C_0，C_1 に対する関係をまとめると以下のとおりである．

集合 S_{add}：$|C| = 0$ の場合を除き $S_{add} = C_0$ となる．$S_{add} \subseteq N(C)$ であり，C が近傍へ遷移するたびに集合 $N(C)$ を走査し $\kappa(v) = |C|$ の頂点を選択することで更新される．

集合 S_{swap}：$|C| \le 1$ の場合を除き $S_{swap} = C_1$ となる．$S_{swap} \subset N(C)$ であり，C が近傍へ遷移するたびに集合 $N(C)$ を走査し $\kappa(v) = |C|-1$ の頂点を選択することで更新される．

集合 $S(i)$：$\Delta+1$ 個の集合族（$i = 0, \ldots, \Delta$）．$S(|C|) = C_0$ となり，$S(|C|-1) = C_1$ となる．更新方法の詳細は以下で述べるが，C が近傍へ遷移するたびに必要最小限の頂点のみ適切な $S(i)$ へ移動させることで更新される．

双方向リストからの頂点の削除を定数時間で行うために，提案法 L は各頂点 v のアドレスを格納した配列 $pos(v)$ を用いる．頂点 $v \in S(\kappa(v))$ を双方向リストから削除する際，配列 $pos(v)$ を参照すれば v のアドレスが定数時間で分かるため，定数時間で v を削除することが可能となる．探索開始時（$C = \emptyset$）には各 $v \in V$ に対し，$pos(v)$ は $S(0)$ における v のアドレスを格納しておく．

また，提案法 L では近傍の重みを効率良く計算するために，クリーク C と各頂点 $v \in V$ に対し $\sigma(v,C)$ を以下のように定義する：

$$\sigma(v,C) = \sum_{u \in C \cap N(v)} w(u,v).$$

以降，特に必要のない限り $\sigma(v,C)$ を $\sigma(v)$ と記す．初期状態（$C = \emptyset$）では，全頂点 $v \in V$ に対し $\sigma(v) = 0$ となる．$\sigma(v)$ の値が分かっていれば，近傍の重みは以下のように $O(1)$ の時間で計算できる．クリーク C に対し，Add 近傍のクリーク $C \cup \{v\}$ の重みは $W(C) + \sigma(v)$ となる．Drop 近傍のクリーク $C \setminus \{u\}$ の重みは $W(C) - \sigma(u)$ となる．Swap 近傍のクリーク $(C \setminus \{u\}) \cup \{v\}$ については，u，v 間に辺が存在せず，Swap 近傍への遷移は Drop 近傍への遷移と Add 近傍への遷移を逐次的に実行するのと同じになるため，重みは $W(C) - \sigma(u) + \sigma(v)$ となる．**図 1** に示したグラフ G_{ex} において，クリーク $C = \{v_4, v_6, v_7\}$ に対する提案法 L のデータ構造を**図 2** に示す．$\Delta = 5$ であるため，提案法 L は $S(0), S(1), \ldots, S(5)$ の 6 つのリストを管理する．この例では $|C| = 3$ であり，$C_0 = S(3) = \{v_5\}$，$C_1 = S(2) = \{v_3, v_8\}$ となる．

$\kappa(\cdot)$，$S(\cdot)$，$pos(\cdot)$ および $\sigma(\cdot)$ の更新はクリーク C が近

表 1 各手法の概要
Table 1 Summary of methods.

手法	グラフのデータ構造	近傍遷移にともなう近傍更新の時間計算量	近傍管理データ構造の空間計算量
従来法 [17]	隣接リスト	$O(\|N(v)\|+\|N(C)\|)$	$O(\|V\|)$
提案法 L	隣接リスト	$O(\|N(v)\|)$	$O(\|V\|)$
提案法 M	隣接行列+非隣接リスト	$O(\|V \setminus N(v)\|)$	$O(\|V\|)$

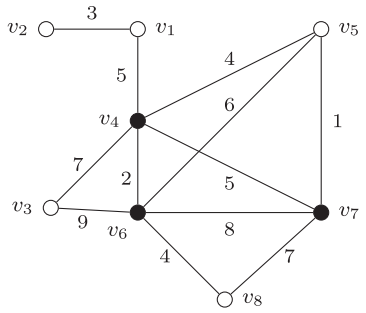


図 1 グラフ G_{ex}（黒い頂点がクリーク C）
Fig. 1 A graph G_{ex} (black vertices are in the clique C).

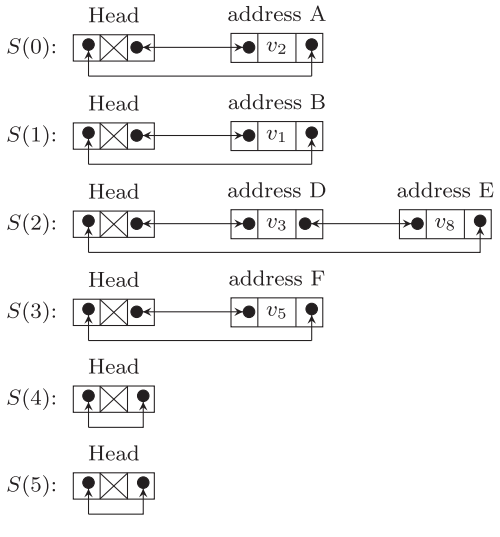


図 2 提案法 L のデータ構造
Fig. 2 Data structures of proposed method L.

傍へと遷移する際に行う．クリークに頂点 v を追加する際の処理と，クリークから頂点 v を削除する際の処理を Algorithm 1 に示す．はじめに C の更新を行い，v が C に追加される場合は v を $S(|C|)$ から削除，v が C から削除される場合は v を $S(|C|)$ に追加する．次に各 $u \in N(v)$ に対し $\kappa(u)$，$\sigma(u)$ を更新し，$u \notin C$ の場合は u を $S(\kappa(u))$ に移動する．Algorithm 1 に示した処理の時間計算量は追加の場合も削除の場合も $O(|N(v)|)$ である．7，21 行目の辺重みの参照は，隣接リストの各要素に対する for 文の中での辺重み参照のため，定数時間でリストに格納されている辺重みを得ることができる．提案法 L ではクリークに含まれないすべての頂点がそれぞれ適切な $S(i)$ に含まれており，文献 [17] における $N(C)$ よりも走査の対象が少ないため，計算量が小さくなっている．

Algorithm 1 提案法 L：データ構造の更新

```
 1: procedure AddToClique(v)
 2:     remove v from S(|C|)                          ▷ Using pos(v)
 3:     pos(v) ← null                                 ▷ not in any S(·)
 4:     add v to C
 5:     for all u ∈ N(v) do
 6:         κ(u) ← κ(u) + 1
 7:         σ(u) ← σ(u) + w(v, u)
 8:         if u ∉ C then
 9:             remove u from S(κ(u) − 1)             ▷ Using pos(u)
10:             insert u to S(κ(u))
11:             pos(u) ← (address of u in S(κ(u)))
12:         end if
13:     end for
14: end procedure

15: procedure DropFromClique(v)
16:     remove v from C
17:     add v to S(|C|)                               ▷ κ(v) = |C|
18:     pos(u) ← (address of u in S(|C|))
19:     for all u ∈ N(v) do
20:         κ(u) ← κ(u) − 1
21:         σ(u) ← σ(u) − w(v, u)
22:         if u ∉ C then
23:             remove u from S(κ(u) + 1)             ▷ Using pos(u)
24:             insert u to S(κ(u))
25:             pos(u) ← (address of u in S(κ(u)))
26:         end if
27:     end for
28: end procedure
```

3.2 提案法 M

はじめに，提案法 M はデータ構造の更新の時間計算量を小さくするために隣接行列から非隣接リストを作成しておく．非隣接リストは各頂点 v に対し，$V \setminus N(v)$ を保持したリストである．隣接行列に加えて非隣接リストを保持しても空間計算量は $O(|V|^2)$ のままである．提案法 M ではクリーク C，各頂点 v に対し，以下の値 $s(v, C)$ を定める：

$$s(v, C) = |C \setminus N(v)|.$$

以降，特に必要のない限り $s(v, C)$ を $s(v)$ と記す．$v \notin C$ に対し，$s(v) = 0$ であるときかつそのときに限り $v \in C_0$ となる．$v \notin C$ に対し，$s(v) = 1$ であるときかつそのときに限り $v \in C_1$ となる．提案法 M では，C_0 および C_1 は提案法 L の $S(i)$ と同じく，双方向リストで実装し，各頂点のアドレスを配列 $pos(v)$ に格納しておく．これにより頂点の追加削除が定数時間で実行可能となる．図 1 に示したグラフ G_{ex} において，クリーク $C = \{v_4, v_6, v_7\}$ に対する提案法 M のデータ構造を**図 3** に示す．初期状態（$C = \emptyset$）では，全頂点 $v \in V$ に対し $s(v) = 0$ となり，すべての頂点が C_0 に属する．各 $v \in V$ に対し，$pos(v)$ は C_0 における v のアドレスを格納しておく．このとき C_1 は空の双方向リスト（Head 要素のみを持つ）である．

$s(\cdot)$，C_0，C_1，$pos(\cdot)$ の更新はクリーク C が近傍へと遷移する際に行う．クリークに頂点 v を追加する際の処理と，

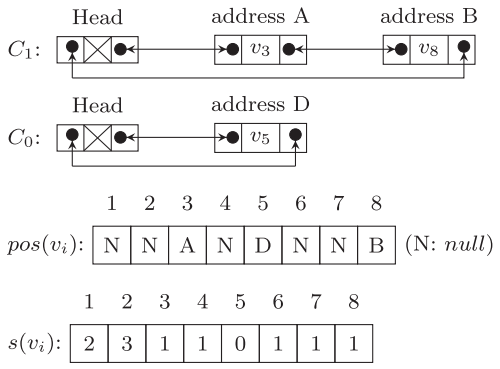


図 3 提案法 M のデータ構造
Fig. 3 Data structures of proposed method M.

Algorithm 2 提案法 M：データ構造の更新

```
 1: procedure AddToClique(v)
 2:     remove v from C_0                          ▷ Using pos(v)
 3:     pos(u) ← null                              ▷ not in C_0 or C_1
 4:     s(v) ← 1
 5:     add v to C
 6:     for all u ∈ V \ (N(v) ∪ {v}) do
 7:         s(u) ← s(u) + 1
 8:         if s(u) = 1 then
 9:             remove u from C_0                  ▷ Using pos(u)
10:             insert u to C_1
11:             pos(u) ← (address of u in C_1)
12:         end if
13:         if s(u) = 2 then
14:             remove u from C_1                  ▷ Using pos(u)
15:             pos(u) ← null                      ▷ not in C_0 or C_1
16:         end if
17:     end for
18: end procedure

19: procedure DropFromClique(v)
20:     remove v from C
21:     s(v) ← 0
22:     add v to C_0
23:     pos(u) ← (address of u in C_0)
24:     for all u ∈ V \ (N(v) ∪ {v}) do
25:         s(u) ← s(u) − 1
26:         if s(u) = 0 then
27:             remove u from C_1                  ▷ Using pos(u)
28:             insert u to C_0
29:             pos(u) ← (address of u in C_0)
30:         end if
31:         if s(u) = 1 then
32:             insert u to C_1
33:             pos(u) ← null                      ▷ not in C_0 or C_1
34:         end if
35:     end for
36: end procedure
```

クリークから頂点 v を削除する際の処理を Algorithm 2 に示す．非隣接行列を用いれば，Algorithm 2 に示した処理の時間計算量は追加の場合も削除の場合も $O(|V \setminus N(v)|)$ である．

提案法 M では，近傍の解の重みが必要になるたびに以下のように隣接行列を用いて解の重みを計算する．クリーク C に対し，Add 近傍のクリーク $C \cup \{v\}$ の重みは $W(C) + \sum_{s \in C} w(s, v)$ を計算すれば得られる．Swap 近傍のクリーク $(C \setminus \{u\}) \cup \{v\}$ の重みは $W(C) - \sum_{t \in C \setminus \{u\}} w(u, t) + \sum_{s \in C \setminus \{u\}} w(s, v)$ となり，Drop 近傍のクリーク $C \setminus \{u\}$ の重みは $W(C) - \sum_{t \in C \setminus \{u\}} w(u, t)$ となる．以上より，近傍の重み計算の時間計算量は近傍の解 1 つあたり $O(|C|)$ である．

4. 計算機実験

提案した 2 つのデータ構造と従来法 [17] のデータ構造を C++で実装し，計算機実験を行った．データ構造を比較するための局所探索法としては PLS [18] に加えて MWCP に対する解法である Multi neighborhood tabu search [15] を基にした解法（以降 MN/TS と記す）を用いた．MN/TS は解の評価値の計算を頂点重みの和から辺重みの和へと変更するだけで MEWCP を解くことができる．

従来法のデータ構造 [17] は MWCP に対して提案されているが，以下の方法で辺重みを扱うことで MEWCP に用いた．文献 [17] では各辺 $(u, v) \in E$ に対し，何らかの対関数 $f(u, v)$ で計算した値をキーとしてハッシュテーブルに要素を格納しておく方法が提案されている．頂点 u，v が隣接しているかどうかはハッシュテーブルのキーとして対関数 $f(u, v)$ の値が存在するかどうかで判定できる．オーバヘッドを考慮すると隣接行列ほど実時間は短くならないものの，隣接判定の時間計算量は平均で $O(1)$ となる．本節の計算機実験では，対関数 $f(u, v)$ をキーとして辺重み $w(u, v)$ をハッシュテーブルに格納しておく方法を用いた．

計算機の CPU は Intel® Core™ i7-6700 3.40 GHz，メモリは 16 GB，OS は Linux 4.4.0 である．使用したコンパイラは g++ 5.4.0 で最適化オプション-O2 を利用した．

以降，すべての実験において，各インスタンスに対し各アルゴリズムを乱数の種を変えて 10 回実行した．制限時間 60 秒以内に到達した最良の解と，その解に到達するまでにかかった時間を計測した．

データ構造によって近傍の管理の方法が異なるため，乱数によって C_0 または C_1 から頂点を選ぶ際に，異なる頂点が選択されることがある．そのため，局所探索のアルゴリズムが同じでも，到達する解が異なる場合がある．

4.1 DIMACS

DIMACS [19] は MCP でよく使用されているベンチマークである．オリジナルの DIAMCS では頂点や辺に重みは付与されていないが，頂点や辺に重みを付与して MWCP や MEWCP のベンチマークとしてもよく使われている [15], [18]．今回の実験では各辺 (v_i, v_j) に対し，$(i + j) \bmod 200 + 1$ の重みを与えるという方法 [18] を用いた．DIMACS の実験結果を**表 2** に示す．W_{best} は全試

表 2 DIMACS の実験結果
Table 2 Experimental result for DIMACS.

				MN/TS		MN/TS + 提案データ構造				PLS		PLS + 提案データ構造			
				文献 [17] の構造		提案法 L		提案法 M		文献 [17] の構造		提案法 L		提案法 M	
Instance	$\|V\|$	d	W_{best}	suc	time	suc	time	suc	time	suc	time	suc	time	suc	time
brock200_1	200	0.745	21,230	9	15.27	10	1.99	10	0.42	10	0.05	10	0.03	10	$< \epsilon$
brock200_2	200	0.496	6,542	8	13.00	10	5.11	10	1.37	10	0.04	10	0.02	10	$< \epsilon$
brock200_3	200	0.605	10,303	9	20.42	10	3.34	10	2.13	10	$< \epsilon$	10	$< \epsilon$	10	$< \epsilon$
brock400_1	400	0.748	35,257	0	-	1	21.18	5	25.75	10	3.12	10	2.01	10	0.13
brock400_2	400	0.749	40,738	1	50.06	7	21.13	10	9.31	10	0.53	10	0.26	10	0.04
brock400_3	400	0.748	46,785	7	23.82	10	2.81	10	0.67	10	0.15	10	0.12	10	0.02
brock400_4	400	0.749	54,304	10	15.00	10	1.69	10	0.34	10	0.09	10	0.04	10	$< \epsilon$
brock800_1	800	0.649	25,050	10	3.80	10	0.83	10	0.14	10	3.30	10	1.29	10	0.35
brock800_2	800	0.651	27,932	0	-	0	-	0	-	9	16.18	9	25.02	10	2.67
brock800_3	800	0.649	30,972	0	-	0	-	2	21.44	10	19.24	9	19.27	10	3.07
brock800_4	800	0.650	30,950	0	-	1	59.37	0	-	10	7.65	10	6.86	10	0.67
C1000.9	1,000	0.900	234,013	1	41.79	8	13.96	10	10.79	2	21.62	3	30.94	10	10.38
C2000.5	2,000	0.500	14,927	6	18.76	9	22.01	10	5.87	4	31.33	4	27.71	8	17.89
C2000.9	2,000	0.900	320,715	0	-	0	-	1	41.44	0	-	0	-	0	-
C4000.5	4,000	0.500	19,304	0	-	2	37.09	3	44.13	0	-	0	-	1	43.79
C500.9	500	0.900	164,953	10	4.89	10	0.14	10	0.07	10	4.36	10	0.98	10	0.17
c-fat500-10	500	0.374	804,000	10	14.80	10	0.01	10	0.21	10	0.05	10	$< \epsilon$	10	$< \epsilon$
c-fat500-2	500	0.073	38,350	10	1.14	10	$< \epsilon$	10	0.05	10	$< \epsilon$	10	$< \epsilon$	10	$< \epsilon$
c-fat500-5	500	0.186	205,864	10	5.42	10	$< \epsilon$	10	0.10	10	$< \epsilon$	10	$< \epsilon$	10	$< \epsilon$
DSJC1000_5	1,000	0.500	12,054	10	2.01	10	0.55	10	0.13	10	4.96	10	1.94	10	0.43
gen400_p0.9_55	400	0.900	150,981	10	2.80	10	0.09	10	0.07	10	1.26	10	0.41	10	0.10
hamming10-2	1,024	0.990	13,140,816	3	9.48	10	0.07	10	1.11	10	3.74	10	0.03	10	0.04
hamming10-4	1,024	0.829	83,280	0	-	3	20.12	8	36.64	0	-	7	34.32	8	26.02
johnson32-2-4	496	0.879	16,330	10	6.25	10	0.51	10	0.13	0	-	0	-	0	-
keller5	776	0.752	38,901	8	22.42	10	3.83	10	0.62	6	22.28	7	22.09	10	5.78
keller6	3,361	0.818	178,189	0	-	1	29.26	0	-	0	-	0	-	0	-
MANN_a27	378	0.990	802,575	0	-	0	-	0	-	0	-	0	-	2	20.92
MANN_a45	1,035	0.996	5,874,190	0	-	0	-	0	-	0	-	0	-	1	20.44
MANN_a81	3,321	0.999	59,893,215	0	-	0	-	0	-	0	-	1	28.63	0	-
p_hat1500-2	1,500	0.506	211,069	10	8.57	10	0.31	10	0.25	10	1.02	10	0.49	10	0.09
p_hat1500-3	1,500	0.754	441,998	5	33.06	10	1.69	10	1.80	10	3.04	10	0.67	10	0.05
san1000	1,000	0.502	10,661	2	43.96	2	12.94	9	19.00	4	39.28	7	24.45	10	13.06
san200_0.7_2	200	0.700	15,073	10	6.53	10	0.37	10	0.15	10	0.35	10	0.04	10	0.01
san400_0.5_1	400	0.500	7,442	9	23.24	10	5.50	10	1.78	10	1.11	10	0.18	10	0.08
san400_0.7_1	400	0.700	77,719	10	19.65	10	2.16	10	0.52	10	4.37	10	0.76	10	0.11
san400_0.7_2	400	0.700	44,155	10	2.09	10	0.15	10	0.08	10	1.66	10	0.32	10	0.04
san400_0.7_3	400	0.700	24,727	10	1.28	10	0.22	10	0.03	10	0.53	10	0.17	10	0.04
san400_0.9_1	400	0.900	496,874	10	3.72	10	0.08	10	0.04	10	1.03	10	0.07	10	0.02
suc の合計回数				638		684		708		695		707		730	

行で得られたクリークの中で，最も大きい重みの値である．suc は 10 回の試行で W_{best} に到達した回数，time は W_{best} に到達した試行のみの平均時間（秒）を示している．記号 $< \epsilon$ は実験環境において計測可能な最短時間 0.01 秒より短い計算時間で W_{best} に到達したことを表している．DIMACS のインスタンスのうち，すべての手法が 10 回とも短時間で W_{best} にたどり着いたものに関しては実験結果を省略した．ただし，表の最下部に示した suc の合計回数については記載を省略したインスタンスも含んでいる．

文献 [17] のデータ構造と提案法 L について suc の回数を比較すると，多くのインスタンスで提案法 L のほうが suc の回数が多いことが確認できた．また，suc の回数が同じインスタンスの多くで，提案法 L のほうが計算時間が小さくなっており，近傍更新の際の時間計算量が小さくなっていることが有利に働いていることが分かった．

提案した 2 つのデータ構造を比較すると，提案法 M のほうが提案法 L よりも suc の回数が多かった．DIMACS は辺密度の高いグラフが多いため，近傍更新のための計算量

が $O(|V \setminus N(v)|)$ である提案法 M のほうが有利であったためだと考えられる．

4.2 BHOSLIB

BHOSLIB [20] は DIMACS と同じく，MCP でよく使用されているベンチマークである．BHOSLIB も頂点や辺に重みは付与されていないが，こちらも DIMACS と同様，文献 [18] と同じ方法で，辺に重みを付与して MEWCP のベンチマークとして使用した．BHOSLIB の実験結果を**表 3** に示す．

suc の合計回数を比較すると，提案法は文献 [17] の構造よりも性能が良いことが確認できた．提案法は提案法 M のほうが提案法 L よりも suc の回数が多かった．DIMACS では PLS の方が suc が多く，BHOSLIB では MN/TS の方

表 3 BHOSLIB の実験結果
Table 3 Experimental result for BHOSLIB.

				MN/TS		MN/TS + 提案データ構造				PLS		PLS + 提案データ構造			
				文献 [17] の構造		提案法 L		提案法 M		文献 [17] の構造		提案法 L		提案法 M	
Instance	$\|V\|$	d	W_{best}	suc	time	suc	time	suc	time	suc	time	suc	time	suc	time
frb30-15-1	450	0.824	44,069	10	0.59	10	0.10	10	0.03	10	0.44	10	0.55	10	0.05
frb30-15-2	450	0.823	44,078	10	0.53	10	0.08	10	0.02	10	0.19	10	0.15	10	0.03
frb30-15-3	450	0.824	43,414	6	30.14	10	5.21	10	1.54	10	9.67	10	7.92	10	1.73
frb30-15-4	450	0.823	43,884	10	2.89	10	0.36	10	0.13	10	0.20	10	0.13	10	0.01
frb30-15-5	450	0.824	43,675	9	35.72	10	4.44	10	1.01	10	9.58	10	5.53	10	0.79
frb35-17-1	595	0.842	59,629	1	11.07	10	20.98	10	6.83	4	22.67	5	34.47	10	5.90
frb35-17-2	595	0.842	59,973	8	30.54	10	3.00	10	1.53	10	15.86	10	14.31	10	0.92
frb35-17-3	595	0.842	60,357	10	8.17	10	1.43	10	0.23	10	3.61	10	4.35	10	0.45
frb35-17-4	595	0.842	59,653	2	24.67	7	28.16	10	7.64	6	39.71	4	10.31	10	11.55
frb35-17-5	595	0.841	60,749	10	5.00	10	0.51	10	0.13	10	6.35	10	2.89	10	0.37
frb40-19-1	760	0.857	79,800	4	31.94	10	9.43	10	1.90	2	17.94	3	26.58	10	9.61
frb40-19-2	760	0.857	79,004	1	52.71	6	37.81	10	13.30	2	30.09	2	22.10	9	13.28
frb40-19-3	760	0.858	79,457	6	15.04	10	2.35	10	1.20	2	9.54	9	9.71	10	4.03
frb40-19-4	760	0.856	79,247	5	19.22	10	9.44	10	5.30	2	32.39	4	26.24	10	15.68
frb40-19-5	760	0.856	79,223	2	34.64	3	29.76	8	30.48	0	-	0	-	4	34.85
frb45-21-1	945	0.867	99,802	0	-	2	11.43	6	25.86	0	-	1	55.92	7	33.30
frb45-21-2	945	0.869	99,838	0	-	0	-	2	40.94	0	-	3	19.20	1	47.50
frb45-21-3	945	0.869	100,282	1	14.48	5	38.82	6	34.29	0	-	0	-	3	37.87
frb45-21-4	945	0.869	101,182	0	-	7	23.96	10	19.61	0	-	2	28.82	7	33.00
frb45-21-5	945	0.869	99,614	0	-	2	18.46	2	12.74	0	-	1	56.26	3	20.26
frb50-23-1	1,150	0.879	122,931	0	-	0	-	2	11.02	0	-	0	-	0	-
frb50-23-2	1,150	0.878	123,674	0	-	0	-	1	39.73	0	-	0	-	0	-
frb50-23-3	1,150	0.877	123,494	0	-	0	-	1	51.31	0	-	0	-	0	-
frb50-23-4	1,150	0.879	123,298	1	41.14	4	27.33	5	15.37	0	-	1	21.98	7	21.06
frb50-23-5	1,150	0.879	122,846	1	2.84	0	-	6	28.27	0	-	0	-	1	45.33
frb53-24-1	1,272	0.883	135,675	0	-	0	-	1	32.05	0	-	0	-	0	-
frb53-24-2	1,272	0.883	140,162	0	-	0	-	2	18.76	0	-	0	-	0	-
frb53-24-3	1,272	0.884	140,122	0	-	0	-	3	48.52	0	-	0	-	2	30.81
frb53-24-4	1,272	0.883	138,758	0	-	2	32.76	0	-	0	-	0	-	0	-
frb53-24-5	1,272	0.883	139,614	0	-	1	11.67	0	-	0	-	0	-	0	-
frb56-25-1	1,400	0.888	151,823	0	-	1	9.41	1	30.04	0	-	0	-	1	17.36
frb56-25-2	1,400	0.888	151,377	0	-	0	-	1	34.34	0	-	0	-	0	-
frb56-25-3	1,400	0.888	150,509	0	-	0	-	1	57.62	0	-	0	-	0	-
frb56-25-4	1,400	0.888	156,615	0	-	0	-	3	16.69	0	-	0	-	0	-
frb56-25-5	1,400	0.888	155,630	0	-	0	-	0	-	0	-	0	-	1	32.29
frb59-26-1	1,534	0.892	170,472	0	-	0	-	1	52.20	0	-	0	-	0	-
frb59-26-2	1,534	0.893	170,232	0	-	1	38.01	1	54.10	0	-	0	-	0	-
frb59-26-3	1,534	0.893	168,343	0	-	0	-	0	-	1	59.36	0	-	0	-
frb59-26-4	1,534	0.892	168,397	0	-	0	-	1	21.40	0	-	0	-	0	-
frb59-26-5	1,534	0.893	172,795	0	-	0	-	2	14.70	0	-	1	20.07	0	-
suc の合計回数				97		161		206		99		116		176	

表 5 higgs-twitter の実験結果
Table 5 Experimental result for higgs-twitter.

				MN/TS				PLS			
				文献 [17] の構造		提案法 L		文献 [17] の構造		提案法 L	
	$\|V\|$	$\|E\|$	W_{best}	suc	time	suc	time	suc	time	suc	time
higgs-reply	38,683	29,552	28	9	25.09	10	0.76	10	1.46	10	0.13
higgs-mention	115,684	140,421	430	10	0.08	10	0.01	10	0.13	10	$< \epsilon$
higgs-retweet	256,491	327,374	101	10	0.20	10	0.03	10	0.44	10	0.02

が suc が多いという違いがあるものの，これは DIMACS と同様の傾向である．

4.3 higgs-twitter データセット

頂点が非常に多く辺疎で巨大なグラフとして，higgs-twitter データセットを用いて実験を行った．このデータセットは，ヒッグス粒子に関して SNS の Twitter のユーザが反応したアクティビティのデータである [21]．このデータを基にして作られたグラフが文献 [22] で公開されている．

higgs-twitter データセットでは，各頂点はユーザに対応する．ユーザ v からユーザ u へのアクティビティは有向辺 (v, u) で表され，辺重みとして，アクティビティの回数が付与されている．計算機実験では，公開されている以下の 3 つのグラフを用いた．

retweet-network リツイートからなるグラフ
reply-network リプライからなるグラフ
mention-network メンションからなるグラフ

これらのグラフは，頂点が数万から数十万存在する巨大なグラフである．我々はこれらのグラフを MEWCP のベンチマークとして用いるために以下の加工を行い，辺重み付き単純無向グラフを得た．

- ループは削除する
- 二頂点 u，v 間に有向辺が存在するときは，向き/数を問わずそれらを削除し 1 つの無向辺 (u, v) に変更する
- 無向辺 (u, v) の重み $w_e(u, v)$ は，元の有向グラフ上での二頂点 u，v 間の有向辺の重みの総和とする．

いずれのアクティビティもユーザ間の交流を示すものであるので，こうして加工されたグラフから最大辺重みクリークを抽出することで結び付きの強いグループを見つけられる．そのようなグループを抽出するうえで，上記の加工は差し支えないと判断した．

4.3.1 データ構造のメモリ使用量

我々が作成した higgs-twitter グラフに対して隣接行列および隣接リストでグラフを表現した場合のメモリ使用量を以下の方法で見積もった．一語長は実験に使用した計算機に基づき，4 byte とした．隣接リストでは，格納される要素の数は $2|E|$ であり，各要素（辺重みが存在するため，要素数 2 のタプル）のサイズは 8 byte である．よって，隣接リストのメモリ使用量は $16|E|$ byte と見積もる．隣接行列では，格納される要素の数は $|V|^2$ であり，各要素のサイズは 4 byte である．よって，隣接行列のメモリ使用量は $4|V|^2$ byte と見積もる．以上の方法で計算した結果を**表 4** に示す．

表 4 higgs-twitter グラフに対するメモリ使用量の見積り
Table 4 Memory usage estimation for higgs-twitter graphs.

グラフ	$\|V\|$	$\|E\|$	隣接行列	隣接リスト
higgs-reply	38,683	29,552	5.57 GB	0.45 MB
higgs-mention	115,684	140,421	49.85 GB	2.14 MB
higgs-retweet	256,491	327,374	245.07 GB	5.00 MB

表 4 から隣接行列では非常に大きなメモリ容量が必要となることが分かる．一方，隣接リストであれば大幅に小さいメモリ容量でグラフを表現することができる．以上より，higgs-twitter に対する実験は隣接リストを用いる手法によってのみ行った．

4.3.2 実験結果

higgs-twitter データセットに対する実験結果を**表 5** に示す．実験結果から，提案法 L を用いることで文献 [17] の構造よりも短い時間で W_{best} に到達できることが確認できた．

5. あとがき

MEWCP に対する局所探索法における近傍管理のための 2 つのデータ構造を提案した．提案法には MWCP に比べて時間のかかる MEWCP の解の重みの計算を効率良く行うための工夫も含まれている．2 つの局所探索法に対して，従来データ構造と提案したデータ構造のすべての組合せで計算機実験を行った．

比較実験のベンチマークとしては DIMACS，BHOSLIB および higgs-tiwtter データセットを用いた．実験結果から，提案法は従来のデータ構造よりも性能が良いことを確認した．提案した 2 つのデータ構造については，DIMACS，BHOSLIB のような辺密度の高いインスタンスでは提案法 M が良いことを確認した．一方，higgs-twitter データセットのように辺疎で巨大なグラフに対しては，メモリ使用量の少ない隣接リストが与えられた場合に用いることができる提案法 L が適していることを確認した．以上から，グラフの頂点数，辺密度および計算機のメモリ容量によって提案法 L と提案法 M を使い分けるのがよいことを確認した．

今後の課題として，より様々な実データでの性能評価が

あげられる．また，本稿で提案した近傍管理のためのデータ構造は，文献 [23] などのヒューリスティックを用いてMEWCP や MCP，MWCP を解く際に，クリークへと追加する候補となる集合を管理するために使うことができる．そのような，局所探索以外のアルゴリズムに本稿で提案したデータ構造を用いた場合の性能評価を行うことも今後の課題としてあげられる．

参考文献

[1] Gary, M.R. and Johnson, D.S.: *Computers and Intractability - A Guide to the Theory of NP-completeness*, W H Freeman and Company (1979).

[2] Bogdanova, G.T., Brouwer, A.E., Kapralov, S.N. and Östergård, P.R.: Error-correcting codes over an alphabet of four elements, *Designs, Codes and Cryptography*, Vol.23, No.3, pp.333–342 (2001).

[3] Sorour, S. and Valaee, S.: Minimum broadcast decoding delay for generalized instantly decodable network coding, *Global Telecommunications Conference*, pp.1–5, IEEE (2010).

[4] Horaud, R. and Skordas, T.: Stereo correspondence through feature grouping and maximal cliques, *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol.11, No.11, pp.1168–1180 (1989).

[5] KC, D.B., Akutsu, T., Tomita, E., Seki, T. and Fujiyama, A.: Point matching under non-uniform distortions and protein side chain packing based on efficient maximum clique algorithms, *Genome Informatics*, Vol.13, pp.143–152 (2002).

[6] Bahadur, K., Akutsu, T., Tomita, E. and Seki, T.: Protein side-chain packing problem: A maximum edge-weight clique algorithmic approach, *Proc. 2nd Conference on Asia-Pacific Bioinformatics-Volume 29*, pp.191–200, Australian Computer Society, Inc. (2004).

[7] Brown, J., Dukka Bahadur, K., Tomita, E. and Akutsu, T.: Multiple methods for protein side chain packing using maximum weight cliques, *Genome Informatics*, Vol.17, No.1, pp.3–12 (2006).

[8] Brown, K.L.: Combinatorial Auction Test Suite (CATS) (2000), available from ⟨http://www.cs.ubc.ca/~kevinlb/CATS/⟩.

[9] Cavique, L.: A scalable algorithm for the market basket analysis, *Journal of Retailing and Consumer Services*, Vol.14, No.6, pp.400–407 (2007).

[10] Corman, S.R., Kuhn, T., McPhee, R.D. and Dooley, K.J.: Studying Complex Discursive Systems, *Human Communication Research*, Vol.28, No.2, pp.157–206 (2002).

[11] Corman, S.R. et al.: Pajek datasets: Reuters terror news network, available from ⟨http://vlado.fmf.uni-lj.si/pub/networks/data/CRA/terror.htm⟩.

[12] 清水悟司，山口一章，増田澄男：数理計画問題による最大辺重みクリーク問題の定式化，電子情報通信学会論文誌（A），Vol.J100-A, No.8, pp.313–315 (2017).

[13] Gouveia, L. and Martins, P.: Solving the maximum edge-weight clique problem in sparse graphs with compact formulations, *EURO Journal on Computational Optimization*, Vol.3, No.1, pp.1–30 (2015).

[14] Pullan, W.: Phased local search for the maximum clique problem, *Journal of Combinatorial Optimization*, Vol.12, No.3, pp.303–323 (2006).

[15] Wu, Q., Hao, J.-K. and Glover, F.: Multi-neighborhood tabu search for the maximum weight clique problem, *Annals of Operations Research*, Vol.196, No.1, pp.611–634 (2012).

[16] Wang, Y., Cai, S. and Yin, M.: Two Efficient Local Search Algorithms for Maximum Weight Clique Problem, *Proc. 30th AAAI Conference on Artificial Intelligence*, pp.805–811 (2016).

[17] Fan, Y., Li, C., Ma, Z., Wen, L., Sattar, A. and Su, K.: Local search for maximum vertex weight clique on large sparse graphs with efficient data structures, *Advances in Artificial Intelligence: 29th Australasian Joint Conference*, pp.255–267, Springer (2016).

[18] Pullan, W.: Approximating the maximum vertex/edge weighted clique using local search, *Journal of Heuristics*, Vol.14, No.2, pp.117–134 (2008).

[19] Trick, M., Chvatal, V., Cook, B., Johnson, D., McGeoch, C., Tarjan, B., et al.: DIMACS Implementation Challenges, available from ⟨http://dimacs.rutgers.edu/Challenges/⟩.

[20] Xu, K.: BHOSLIB: Benchmarks with Hidden Optimum Solutions for Graph Problems (Maximum Clique, Maximum Independent Set, Minimum Vertex Cover and Vertex Coloring), available from ⟨http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/graph-benchmarks.htm⟩.

[21] De Domenico, M., Lima, A., Mougel, P. and Musolesi, M.: The Anatomy of a Scientific Rumor, *Scientific Reports*, Vol.3, p.2980 (2013).

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

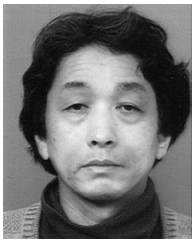
[23] Cerrone, C., Cerulli, R. and Golden, B.: Carousel greedy: A generalized greedy algorithm with applications in optimization, *Computers & Operations Research*, Vol.85, pp.97–112 (2017).

清水 悟司 （学生会員）

平成 24 年神戸大学工学部卒業．平成 26 年同大学大学院博士前期課程修了．現在，同大学院博士後期課程在学．主にアルゴリズムとデータ構造の研究に従事．電子情報通信学会会員．

石原 諒大

平成 28 年神戸大学工学部卒業．現在，同大学大学院博士前期課程在学．グラフ理論のクリークに関するアルゴリズムの研究に従事．



山口 一章 （正会員）

平成 2 年大阪大学基礎工学部情報科学科卒業．平成 7 年同大学大学院博士後期課程単位取得退学，神戸大学大学院工学研究科電気電子工学専攻助手．現在，同大学院准教授．博士（工学）．アルゴリズムの設計，グラフ理論等の研究に従事．IEEE，SIAM，電子情報通信学会，日本 OR 学会，人工知能学会各会員．

増田 澄男 （正会員）

昭和 54 年大阪大学基礎工学部情報科学科卒業．昭和 59 年同大学大学院博士後期課程修了．工学博士．同大助手，講師を経て，平成 3 年神戸大学助教授．現在，同大学大学院教授．主として，アルゴリズムとデータ構造，およびグラフ理論の応用に関する研究に従事．電子情報通信学会，電気学会，日本応用数理学会各会員．