B-101

ある最大重みクリーク抽出法における頂点集合の効率的な実装方法の提案

An Efficient Implementation of Vertex Sets in an Algorithm for the Maximum Weight Clique Problem

清水 悟司† 森中 諒太 † 山口 一章 † 増田 澄男 †
Satoshi Shimizu Ryota Morinaka Kazuaki Yamaguchi Sumio Masuda

1. まえがき

無向グラフにおいて，任意の二頂点間に辺が存在する頂点誘導部分グラフをクリークという．頂点に重みが付与された無向グラフが与えられたとき，頂点の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という．最大重みクリーク問題の厳密解を求める手法として，分枝限定法に基づくアルゴリズムがいくつか提案されている．

本稿では，最大重みクリーク問題を解く Kumlander の手法 [2] について，頂点集合の実装方法の検討を行う．集合の表現方法としては，配列などに要素を列挙する方法（以降，リスト表現と呼ぶ）と，要素が集合に含まれるか否かを示す真偽値を 2 進数の各桁で表現する方法（以降，bit 列表現と呼ぶ）が考えられる．

bit 列表現において，台集合の要素数が整数型で扱うことのできる 2 進数の桁数 R[bit]（以降，1 語長と呼ぶ）を超える場合は整数の配列などを用いる必要があるが，台集合の要素数が 1 語長以内であれば単一の整数変数で表現することができる（以降，1 語長 bit 列表現と呼ぶ）．

本稿では，Kumlander の手法を変更し，1 語長 bit 列表現で実装する方法を提案する．計算機実験により，1 語長 bit 列表現による実装は，リスト表現や通常の bit 列表現による実装よりも高速なことを確認した．

本稿の構成は以下のとおりである．まず **2.** において，最大重みクリーク問題に関連する用語や記号の定義を示す．**3.** では Kumlander の手法の概略を示す．**4.** では提案法について述べる．**5.** では計算機実験の結果を示す．最後に **6.** で本研究の結果をまとめる．

2. 諸定義

本論文で用いる記号，語句の定義を示す．入力として与えられるグラフ $G = (V, E)$ の頂点数を n とする．頂点 v $(\in V)$ の重みを $w[v]$ と記す．頂点 v に隣接している頂点の集合を $N(v)$ と記す．グラフ中の 2 頂点をランダムに選んだとき，その頂点間に辺が存在する確率を辺密度と呼ぶ．V を独立頂点集合 $S_1, S_2, \cdots, S_k$ に分割することを彩色と呼ぶ．このとき各 S_i を色組と呼ぶ．bit 列の中で最上位の 1 の bit を MSB と呼ぶ．

3. 従来法

Kumlander の手法は，Östergård による最大重みクリーク問題の厳密解法 [1] を改良したものである．まず最初にÖstergård の手法の概略を示し，次に Kumlander による手法の概略を示す．

3.1 Östergård の手法

$V = \{v_1, v_2, \cdots, v_n\}$，$V_i = \{v_i, v_{i+1}, \cdots, v_n\}$ とする（つまり $V_1 = V$ である）．Östergård のアルゴリズムでは，$i = n, n-1, \cdots, 2, 1$ の順に，V_i による G の頂点誘導部分グラフの最大重みクリークを求め，それぞれの重みを $c[i]$ に保存する．各 i における探索は分枝限定法で行うが，部分問題の上界として $c[\cdot]$ を用いることができる．すなわち，ある部分集合 S に対し，$c[\min\{j|v_j \in S\}]$ が最大重みクリークの重み上界となる．この上界計算は S の要素の最小の添字を調べるだけで良く，$O(1)$ 時間で実行可能である．

以上の探索処理を backtrack search と呼ぶ．backtrack search の計算時間は，V をどのような順序で処理するかによって大きく変わることが分かっている．重みの降順や彩色した色毎に並べるなどが良いことが経験的に知られている．

3.2 Kumlander の手法

$S = (S_1, S_2, \cdots, S_k)$ をグラフ G の彩色とする．このとき，$\sum_{i=1}^{k} \max\{w[u]|u \in S_i\}$ は最大重みクリークの重みの上界となる（以降，彩色による上界と呼ぶ）．Kumlander の手法ではまずグラフ G の彩色を行い，色組ごとに重みの降順の頂点系列を作成する．その色組ごとの系列を一つに連接し，前から順に $v_1, v_2, \cdots, v_n$ とする．以降の処理では，V の部分集合は色組ごとに保持する．彩色の上界は各色の系列の先頭の頂点の重みを合計すれば良いので，色組の数に比例した時間で計算できる．Kumlander は彩色による上界を backtrack search に組み合わせることで Östergård の手法を高速化することに成功した．以下にアルゴリズムを示す．なお，*expand*

†神戸大学，Kobe University

の第一引数 S は V のある部分集合 V' の彩色の色組を系列として保持したものである．

Kumlander のアルゴリズム

```
入力: 頂点に重みが与えられた無向グラフ
1: function main
2:     彩色 S_1, S_2, ..., S_k を求め，頂点系列を作る
3:     record ← 0
4:     for i from n to 1 do
5:         expand(V_i ⋂ N(v_i), w[v_i])
6:         c[i] ← record
7:     end for
8:     最大重みクリークを出力
9:     return

10: function expand(S, weight)
11:     if |S| = 0 then
12:         if weight > record then
13:             record ← weight
14:             最大重みクリークを更新
15:         end if
16:         return
17:     end if
18:     上界 ← 0
19:     for i from 1 to N(S) do
20:         j ← min{k | v_k ∈ S_i}
21:         上界 ← 上界 + w[v_j]
22:     end for
23:     for i from 1 to N(S) do
24:         l ← min{k | v_k ∈ S_i}
25:         while |S_i| ≠ 0 do
26:             if weight + 上界 ≤ record then
27:                 return
28:             end if
29:             j ← min{k | v_k ∈ S_i}
30:             if weight + c[j] ≤ record then
31:                 return
32:             end if
33:             S_i ← S_i \ v_j
34:             expand(S ⋂ N(v_j), weight + w[v_j])
35:         end while
36:         上界 ← 上界 − w[l]
37:     end for
38:     return record
```

4. 提案法

1 語長 bit 列表現での実装方法を以下に示す．ただし以下の説明の大半は，計算量の評価を除けば，1 語長という制限のない通常の bit 列表現でも成立する．

Kumlander のアルゴリズムでは，$V_i = \{v_i, v_{i+1}, \cdots, v_n\}$ とし，$i = n, n-1, \cdots, 2, 1$ の順に，V_i による G の頂点誘導部分グラフの最大重みクリークを順に求めていくが，この部分を以下のように変更する．

- $V_i = \{v_1, v_2, \cdots, v_i\}$ とする．
- $i = 1, 2, \cdots, n-1, n$ の順に，V_i による G の頂点誘導部分グラフの最大重みクリークを求めていく．

v_i を i 桁目に対応させるような bit 列表現を用いる場合，Östergård で使われていた V_i の定義に従って処理を行うと，最初に扱う小さな部分問題に対して大きな桁の bit 列が必要になってしまう．上記の変更により，分枝変数は常に最上位桁が選ばれるようになり，小さな部分問題を表現する bit 列の桁が小さくなる．

4.1 集合の演算

一般に，bit 列による集合表現は，リスト表現に比べ，メンバ判定，要素の追加・削除，積集合の計算などが高速に行えるが，空集合か否かを判定する処理や，要素を一つずつ取り出す処理の手間が大きい．1 語長 bit 列表現においては，空集合か否かを判定する処理は $O(1)$ で実行できるので，以下では，要素を一つずつ取り出す処理について説明する．

Kumlander のアルゴリズムでは，要素を順に取り出す処理では上位ビットから順に取り出す．よって，その処理は以下の二つの処理を用いることで実現可能である．

- MSB の桁を計算する処理
- MSB を 0 にする処理

bit 列表現において上記二つの処理を行う高速な方法はいくつか提案されている [3][4]．1 語長 bit 列表現の $R = 32$ の場合においてそれらの中で最も高速だったアルゴリズムを以下に示す．ただし，$Lshift(x, k)$ は x を左に $k$bit シフトした値を，$Rshift(x, k)$ は x を右に $k$bit シフトした値を返す．

MSB の位置を取得する処理の実装 ($R = 32$)

入力: 整数値 x
$MSB \leftarrow 1$
if ($Rshift(x, 16) \ = \ 0$) then
  $MSB \leftarrow MSB \ + \ 16$
  $x \ \leftarrow \ Lshift(x, 16)$
fi
if ($Rshift(x, 24) \ = \ 0$) then
  $MSB \leftarrow MSB \ + \ 8$
  $x \ \leftarrow \ Lshift(x, 8)$
fi
if ($Rshift(x, 28) \ = \ 0$) then
  $MSB \leftarrow MSB \ + \ 4$
  $x \ \leftarrow \ Lshift(x, 4)$
fi
if ($Rshift(x, 30) \ = \ 0$) then
  $MSB \leftarrow MSB \ + \ 2$
  $x \ \leftarrow \ Lshift(x, 2)$
fi
$MSB \ \leftarrow \ MSB \ - (Rshift(x, 31))$
出力: MSB

上記アルゴリズムは与えられた整数の MSB の位置を二分探索で求める方法である．まず上位 16bit に 1 が立っているか確認し，立っていれば上位 16bit を，立っていなければ下位 16bit を調べる．次にその 16bit のうち上位 8bit に 1 が立っているか確認し，という操作を繰り返して MSB の位置を調べている．

4.2 分枝操作と限定操作

Kumlander のアルゴリズムにおいては，部分集合 S に対して

操作 1: 新しい部分問題 $S \bigcap N[v_i]$ を求める．

操作 2: 部分問題から頂点を順に取り出して調べる．

操作 3: 彩色の上界を計算する．

の操作を行う．**3.** のアルゴリズムにおいて，操作 1 は 34 行目で行われる．操作 2 は 29 行目の系列の先頭の頂点を取り出す処理と 33 行目のその頂点を削除する処理によって行われる．操作 3 は 20 行目と 24 行目の S_i から先頭の頂点を取り出す処理によって行われる．

操作 3 を高速に実行するために，S は色組ごとに 1 語長 bit 列で表現するものとする．1 語長 bit 列表現を用いるためには，各色組の要素数（以下，色サイズと呼ぶ）は R 以下でなければいけない．Kumlander のアルゴリズムは，まず，頂点を重みの降順に Greedy 彩色するが，そのままでは要素数が R より大きな色組ができる可能性がある．提案法では，Greedy 彩色において，ある色組の要素数が R になったときに，それ以上追加しないという方法を用いる．

各操作の実装方法を説明する．操作 1 については，グラフの隣接行列を色組ごとの bit 列で表現しておくことにより，色組ごとの bit 積の計算で実行できる．操作 2 については 4.1 で既に解説した．操作 3 は，各色組の最上位ビットを求め，その頂点の重みの合計を求める．各色組の最上位ビットの計算法については **4.1** で解説したものを用いれば良い．

以上の変更により，1 語長 bit 列表現の実装ができた．1 語長 bit 列表現においては，集合を例えば C 言語などにおける int 型一つで表現できるので，通常のビット列表現において必要なループや条件分岐が省略され，より高速に処理できるようになる．

5. 計算機実験

提案法は各色組を 1 語長 bit 列で表現するものであるが，集合に関する処理が通常の bit 列表現に比べ高速化される代わりに，彩色で必要な色数が増え，上界が悪くなり分枝数が増える可能性がある．処理の高速化と分枝数の増加のそれぞれの効果を検証するため，以下のような計算機実験を行った．

1 語長 bit 列で表現する提案法，語長に制限のない通常の bit 列表現，リスト表現に加え，色サイズを R 以下に制限したリスト表現も比較対象とした．リスト表現において色サイズの制限のあるものとないものを比べることにより，分枝数の増加の度合いを調べることができる．bit 列のサイズ R は 32bit とした．入力として頂点数 100 から 2000 のランダムグラフを辺密度を変えながら与えた．頂点に与える重みは 1 から 10 のランダムな値とした．各頂点数，辺密度に対して 100 個ずつグラフを作成し，計算時間の平均値を求めた．使用した計算機の CPU は Intel(R) Core(TM) i7-2600 3.40GHz，メモリは 8GB，OS は Linux，プログラミング言語は Java(OpenJDK 1.6.0_22) である．表 1 に実験結果を示す．

色サイズに制限のないリスト表現と通常の bit 列表現の計算時間を比較すると，bit 列表現のほうが高速であることが確認できた．頂点数，辺密度が大きくなればなるほど bit 列表現のほうが相対的に速くなっていくことがわかる．これは，頂点数，辺密度が大きければその分，集合演算の速度の影響が大きくなるからであると考えられる．

次に，リスト表現の色サイズ制限ありと制限なしを比べる．これらに計算時間に差は見られず，制限を加えてもあまり悪くなっていないことが確認できる．おそらく，要素数が 32 を超える色組はあまり現れないか，あっても影響は少ないものと思われる．

表 1: リスト表現と bit 列表現の計算時間 (秒)

頂点数	辺密度	リスト表現	リスト表現 (制限あり)	bit 列表現	1 語長 bit 列表現
100	0.7	0.0198	0.02	0.0295	0.0201
100	0.8	0.034	0.0345	0.0358	0.0296
100	0.9	0.1385	0.1391	0.0968	0.0655
200	0.6	0.0779	0.0774	0.0696	0.0531
200	0.7	0.4481	0.4483	0.3175	0.2094
200	0.8	7.5961	7.5999	4.9862	2.9527
300	0.5	0.1148	0.1154	0.0885	0.0629
300	0.6	0.6708	0.6702	0.4713	0.3084
300	0.7	9.7222	9.7419	6.9127	4.1818
400	0.5	0.4699	0.4718	0.3232	0.2149
400	0.6	4.6336	4.6241	3.2505	2.0071
400	0.7	> 120	> 120	> 120	52.7103
500	0.4	0.2213	0.2205	0.1496	0.1046
500	0.5	1.6137	1.613	1.0716	0.6729
500	0.6	21.9159	21.885	15.3116	9.2162
600	0.4	0.509	0.5112	0.3288	0.2155
600	0.5	4.7127	4.7122	3.1267	1.9059
600	0.6	> 120	> 120	63.9979	37.8319
700	0.3	0.1725	0.1723	0.114	0.0812
700	0.4	1.0634	1.0631	0.6713	0.4211
700	0.5	12.0875	12.0887	8.1598	4.8289
800	0.3	0.2838	0.282	0.1752	0.1201
800	0.4	2.1645	2.1592	1.3705	0.8282
800	0.5	28.6267	28.6758	19.4657	11.3041
900	0.3	0.4623	0.4619	0.2788	0.1839
900	0.4	3.9447	3.9314	2.497	1.475
900	0.5	62.6622	62.73	43.3771	24.8047
1000	0.3	0.7182	0.7175	0.4179	0.2649
1000	0.4	7.1042	7.1041	4.5464	2.6425
1000	0.5	> 120	> 120	> 120	51.7193
1500	0.2	0.4253	0.4266	0.2337	0.1587
1500	0.3	4.4584	4.4554	2.5853	1.4973
1500	0.4	73.5465	73.4578	48.514	26.5243
2000	0.1	0.1355	0.1318	0.1356	0.0757
2000	0.2	1.3196	1.3185	0.697	0.4261
2000	0.3	18.2505	18.2084	10.9029	5.971

通常の bit 列表現と 1 語長 bit 列表現を比べると，1 語長の制限があるほうが計算時間が大きく減少していることが確認できた．制限により上界はあまり悪くなっておらず，通常のビット列表現において必要なループや条件分岐が省略されたことによる高速化の効果が大きかったと思われる．

通常のリスト表現と 1 語長 bit 列表現を比べると，最大で約 3 倍の高速化が確認できた．頂点数，辺密度が大きくなればなるほど 1 語長 bit 列表現は高速になっているのでより大きいグラフで比較すればより大きい性能差が確認できると思われる．

以上の結果より，Kumlander の手法において bit 列表現が有効であること，および，1 語長 bit 列表現がより有効であることが確認できた．

6. あとがき

頂点に重みが付いた無向グラフが与えられたときに最大重みクリークを求める Kumlander の手法において，1 語長 bit 列表現を用いる方法を提案した．Kumlander の手法はもともと bit 列表現の方が有利であったようだが，色サイズを制限することによりさらに高速化されたことを実験により確認した．

提案法は，最大重みクリーク問題以外のグラフの組合せ最適化問題に応用できる可能性がある．今後は，その可能性について検討していきたい．

参考文献

[1] P.R.J. Östergård, “A New algorithm for the maximum-weight clique problem,” Nordic Journal of Computing,vol. 8, pp.424-436, 2001

[2] Deniss Kumlander “Some practical algorithms to solve the maximum clique problem,” Tallinn University of Technology Press, Tallinn 2005

[3] Leiserson C, Prokop H, Randall K. “Using de Bruijn sequences to index a 1 in a computer word,” http://supertech.csail.mit.edu/papers/debruijn.pdf., 1998.

[4] Warren Jr HS. “ハッカーのたのしみ,” エスアイビーアクセス, 2004.

[5] Pablo SanSegndo, Diego Rodriguez-Losada, Agustin Jimunez, “An exact bit-parallel algorithm for the maximum clique problem,” Computers & Operations Research 38 pp571-581, 2011,