B-04

最大重みクリーク問題に対する分枝限定法に基づく近似解法に関する研究

A study on branch-and-bound heuristics for the maximum weight clique problem

芝野 悟 † 山口 一章 † 斎藤 寿樹 † 増田 澄男 †
Satoru Shibano Kazuaki Yamaguchi Toshiki Saitoh Sumio Masuda

1 はじめに

無向グラフ $G = (V, E)$ に対し，V の部分集合 C の任意の 2 頂点が隣接しているとき C をクリークと呼ぶ．各頂点に重みが与えられているとき，クリークに属する頂点の重みの和をクリークの重みとする．無向グラフ $G = (V, E)$ と各頂点の重みが与えられたとき，そのグラフ中から最大の重みをもつクリークを抽出する問題を最大重みクリーク問題と呼ぶ．最大重みクリーク問題は最大クリーク問題を一般化した問題であり，NP 困難であり，多項式時間で大域的最適解を求めるアルゴリズムは存在しないであろうと考えられている．以下，特に断らない限り最適解と記したときは大域的最適解を意味するものとする．また，通常，理論計算機科学においては近似解とは何らかの近似精度の保証のある解法によって得られた解を指すが，本稿では発見的手法により得られるような精度保証のない解についても近似解と記すことにする．

最大重みクリーク問題の解法としては，局所探索法に基づく解法がいくつか提案されている（例えば [3]）．また，高速な厳密解法も提案されている [2]．これらのうちどの手法が用いられるかは状況によって異なる．

本稿では以下のような特徴を持つ手法について考える．

- 必ず有限時間内に最適解が求まる．
- 計算を途中で打ち切った場合でもそれなりに良い近似解が得られる．

上記の二つの要求を満足するような方法として Limited Discrepancy Search[1] という探索法がある（以下，LDS と略す）．なお，局所探索法などのメタヒューリスティックに基づく解法は（局所最適解に陥るなどして）最適解が得られない可能性があるため上記の一つ目の特徴を持たない．

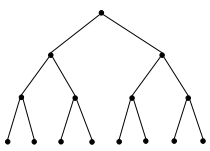
LDS の基本的なアイデアは，解空間の探索において，最適解が含まれる可能性が高い（と思われる）部分空間から順に探索を行うという方法である．LDS は上に示した特徴に加え，必要な記憶領域が極めて小さいという特徴を持つが，解空間の同じ個所を何度も探索するため計算時間については改良の余地がある．

本稿では，LDS における探索法において，空きメモリを用いて計算の重複を省くことにより，より高速に良い近似解を得る方法を提案する．計算機実験によりその有効性を検証する．LDS は解空間を探索する一般的な手法であるが，本稿では分枝限定法による探索で用いるため，まず，分枝限定法の説明を行う．次に LDS の説明を行う．その後，提案法の説明と計算機実験による評価を行い，最後にまとめと今後の課題を示す．

2 分枝限定法

分枝限定法は分枝操作と限定操作の 2 つの操作からなる．分枝操作とは問題をより小さな部分問題に分割する操作であり，本研究では場合分けを用いておこなう．限定操作とは分枝操作によって作られた部分問題のうち，求める解が存在しないと判定できる部分問題を解かずに捨てる操作（枝刈り）である．この判断は部分問題の解の上界を計算をすることによって行う．上界がそれまでに求まった暫定解の評価値以下であればその部分問題を探索する必要がなくなる．分枝限定法においては，より良い上界を得ることと，より良い解が得られる部分問題に早い段階でたどり着くことで，多くの部分問題を枝刈りすることができる．

与えられた問題を分枝操作によって小さな部分問題に分割していく様子は探索木という形で表すことが出来る．探索木の各 node は部分問題を表し，根は与えられた問題を表す．葉は実行可能解を表す．以降では，部分問題の探索順序の説明において探索木を用いる．



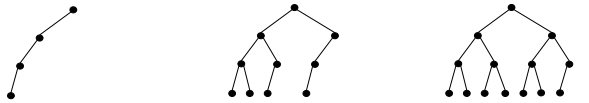
探索木

†神戸大学大学院工学研究科，Graduate school of engineering, Kobe university

3 LDS

探索木において，左の部分木の方が右の部分木に比べて最適解が含まれる可能性が高くなるように部分問題を作成されているものとする．探索木の探索を行う際，右部分木をたどる回数を Discrepancy と呼ぶ．

LDS は，右部分木をたどる回数の上限 D を定めて探索を行う．まず $D = 0$ として気を探索する（葉の中では左端の葉のみが訪問される）．その後，$D = 1$，$D = 2$ と，順に D を増やしながら探索範囲を広げる．左の部分木に最適解が含まれる確率が高くすることで，小さな D で最適解を見つけることができる．



[1]D=0 での探索範囲　[2]D=1 での探索範囲　[3]D=2 での探索範囲

LDS による探索木の探索

以下では LDS の概略を説明する．まず，アルゴリズム中で使われている記号は以下の通り．

$w_t(v)$	頂点 v の重み
$N(v)$	頂点 v に隣接する頂点の集合
P	4 項組 (D_P, w_P, V_P, C_P) からなる部分問題
D_P	探索木の根から P までで右部分木をたどった回数
V_P	使用可能な頂点の集合
C_P	現時点のクリーク
w_P	C_P に含まれる頂点の重みの和
C_{max}	暫定最適解
max	C_{max} に含まれる頂点の重みの和

LDS の手順を記す．LDS は再帰呼び出しを用いて書くことができる．2 行目にて部分問題 P が探索木の葉（再帰の終端）か否かを判定する．終端の場合，C_P が暫定最適解（その時点で求まっている最良の解）よりも良い解ならば暫定最適解を更新する．

P が探索木の内部節点ならば（7 行目以降），まず分枝変数となる頂点 v を選ぶ（8 行目）．v を選ぶ際は，何らかの方法によってできるだけ最適解に含まれる可能性の高い頂点を選ぶ（後に示す実験では重みが最大の頂点を選ぶことにした）．以降，v を使わない部分問題 SP_2（右の部分問題）と v を使う部分問題 SP_1（左の部分問題）について調べる．SP_2 は右側の部分木に対応するので，まず Discrepancy の上限 D を超えないかどうかチェックする（9 行目）．その後，Discrepancy の値を 1 増やして再帰呼び出しをする．SP_1 は使用できる頂点とクリークを更新して再帰呼び出しを行う（13〜15 行目）．Discrepancy の値は変わらないので SP_2 の場合と異なり無条件に実行される．

ここでは省略したが，通常の分枝限定法と同じように枝刈りを行うことができる．SP_2 を作成した際（10 行目）にその上界が max 以下であれば SP_2 を枝刈りできる．SP_1 についても同様．

LDS のアルゴリズム

```
1:  Procedure LDS(P,D)
2:      if V_P = ∅ then
3:          if w > max then
4:              max ← w
5:              C_max ← C
6:          endIf
7:      else
8:          V から頂点 v をひとつ選ぶ.
9:          if D_P < D then
10:             SP_2 ← (D_P + 1, w_P, V_P \ {v}, C_P)
11:             LDS(SP_2,D)
12:         endIf
13:         w_1 ← w_P + w_t(v)
14:         SP_1 ← (D_P, w_1, V_P ∩ N(v), C_P ∪ {v})
15:         LDS(SP_1,D)
16:     endIf
17: endProcedure
```

4 提案手法

LDS は $D = k$ のとき Discrepancy が k 以下の全ての node を探索する．LDS は $D = 0, 1, 2, \ldots$ と順に探索範囲を広げるため，Discrepancy i の node は $D = k$ の探索開始前に既に $k - i$ 回訪問済となっている．

提案手法では，訪問済の node を再び探索しないようにするために，メモリの許す限り探索木を記憶しておく．LDS では D の値が 1 増える度に根から探索をし直すが，探索木を部分的に記憶しておくことで未訪問の node から探索を開始できる．提案手法ではこのようにして暫定解をより速く改良する．

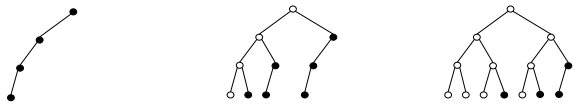
以下ではそのアイデアを忠実に実行する提案手法 1 と，部分問題の探索順序に工夫を加えた提案手法 2 について説明する．

4.1 提案手法 1

探索時に現れる部分問題を二つのリスト Q_1,Q_2 で管

理し，同じ node を探索しないようにする．D の値を増やすときにその時点でのメモリの空き状態を調べ，十分な空きがあればサブルーチン Search1 で新たに部分問題を作りリスト Q_1 に加える．

以下に探索の様子を示す．$D = 0$ の探索においては左側の部分問題のみが調べられる（下図 (a)）．これらの部分問題は Q_1 に保存される．$D = 1$ の探索においては Q_1 から部分問題を順に取り出し（下図 (b) の白丸），そこから新たな部分問題（下図 (b) の黒丸）を作り Q_1 に保存する．同様に，$D = 2$ の探索においては Q_1 から部分問題を順に取り出し（下図 (c) の白丸），そこから新たな部分問題（下図 (c) の黒丸）を作り Q_1 に保存する．なお，通常の LDS では D の値が変わる度に探索木の根から探索が行われるため，白丸に対応する部分問題が何度も調べられるが，提案手法ではその無駄が省かれる．



(a)$D = 0$ での探索範囲　(b)$D = 1$ での探索範囲　(c)$D = 2$ での探索範囲

提案手法 1 による探索木の探索

部分問題を保持するのに十分なメモリがなくなれば以降は Q_1 で保持している部分問題を順番に LDS を用いて調べる．以降は白丸に対応する部分問題は新たに増えることはない．ただし，LDS は D が増える度に白丸に対応する部分問題を新たに訪問することになるが，提案手法ではその手間が省かれる．

提案手法 1

```
    Procedure Propose1 (P)
1:      Q_1 を空にした後，P を入れる．
2:      while Q_1 に十分な空きがある
3:          Search1()
4:      endWhile
5:      t ← 0
6:      While (Q_1 ≠ ∅)
7:          Q_1 の各問題 P' に対し
            LDS(P',t) を実行する．
8:          t ← t + 1
9:      endWile
10: endProcedure
```

提案手法 1 のサブルーチン Search1 は，Q_1 に格納された部分問題から二つの部分問題を作り，左の部分問題は Q_1 の先頭に，右の部分問題は Q_2 の末尾に入れる．これにより，左の部分問題からは通常の深さ優先探索のように探索が行われ，右の部分問題は D の値が 1 増えた状態において LDS と同じ順序で探索が行われるように Q_2 に格納されていく．なお，SP_1, SP_2 を Q_1, Q_2 に入れる前にそれぞれの最適解の上界を求めて枝刈りを行うことが可能である．

提案手法 1 のサブルーチン

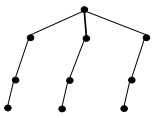
```
1:  Procedure Search1()
2:    While (Q_1 ≠ ∅)
3:      Q_1 の先頭から部分問題 P を取り出す
4:      if w_P > max then
5:        max ← w_P
6:        C_max ← C_P
7:      endIf
8:      if V_P ≠ ∅ then
9:        V_P から頂点 v をひとつ選ぶ．
10:       w_1 ← w_P + w_t(v)
11:       SP_1 ← (D_P, w_1, V_P ∩ N(v), C_P ∪ {v})
12:       SP_2 ← (D_P + 1, w_P, V_P \ {v}, C_P)
13:       if SP_1 の上界 > max then
14:         Q_1 の先頭に SP_1 を加える
15:       endIf
16:       if SP_2 の上界 > max then
17:         Q_2 の最後尾に SP_2 を加える
18:       endIf
19:     endIf
20:   endWhile
21:   Q_2 の部分問題を Q_1 に移す
22: endProcedure
```

4.2 提案手法 2

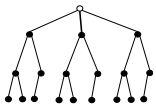
LDS や提案手法 1 を含むその改良版では，分枝変数の選び方が性能を大きく左右する．もし同じぐらいの価値があると思われる頂点がいくつか存在した場合，LDS や提案手法 1 ではランダムに選ぶしかないが，その選択の良否が結果に大きな影響を与えることになる．

提案手法 2 では，最初の分枝変数を選択する際に同じぐらいの価値の頂点が複数ある場合，それらを全て同等に扱う．すなわち，$v_1, v_2, \ldots, v_m$ が同等な価値を持つと判断された場合，$v_1, v_2, \ldots$ を順に分枝変数として選ぶが，右側の部分木を選ぶ際に Discrepancy をカウントしないことにする．これにより，m 個の部分問題がほぼ同等に扱われることになり，分枝変数の選択の良否の影響を小さくすることができる．なお，手順 7: において部分問題を Q_1 に戻す際，上界計算を行って枝刈りを行う

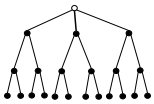
ことが可能である．



[1]D=0 での探索範囲



[2]D=1 での探索範囲



[3]D=2 での探索範囲

提案手法 2 による探索木の探索

提案手法 2

```
1:  Procedure Propose2 (P)
2:      max = 0, C_max = ∅
3:      MakeSubProblem(P)
4:      t ← 0
5:      while  Q_1 ≠ ∅
6:      LDS(Q_1 の先頭の部分問題,t)
7:      Q_1 の最後尾に部分問題を戻す
8:      一巡したら t の値を 1 増やす
9:      endWhile
10: endProcedure
```

提案手法 2 のサブルーチン

```
1:  Procedure MakeSubProblem (P)
2:      最も価値の高い頂点が m 個あるとする．
        それらを v_1, v_2, ..., v_m とする．
3:      for i = 1 to m
4:          V_i ← (V \ {v_1, v_2, ..., v_m}) ∩ N(v_i)
5:          SP_i ← (0, w_t(v_i), V_i, {v_i})
6:          Q_1 の最後尾に SP_i を加える
7:      endfor
8:      SP = (0, 0, V \ {v_1, v_2, ..., v_m}, ∅)
        を Q_1 の最後尾に加える
9:  endProcedure
```

5 計算機実験

実験方法は頂点数 400 辺密度 0.9 のランダムグラフ 10 通りを入力として各段階での保持している暫定解の値の平均値をとった．頂点の重みは 1〜10 の自然数とした．実験環境は OS は Windows 7 professional 64bit，メモリは 8GB，言語は Java である．なお，分枝変数の選択では重みの大きい頂点を優先し，限定操作における上界計算では Optimal Table[2] を用いた．

実験結果

経過時間 [ms]	深さ優先	LDS	提案手法 1	提案手法 2
10	245.8	288.0	297.2	305.3
20	294.8	312.9	312.7	313.1
30	294.8	312.9	312.7	313.1
40	297.3	315.7	314.2	315.9
50	297.3	316.2	315.7	316.4
60	297.3	316.2	315.7	316.4
70	298.3	316.7	317.1	317.8
80	298.3	318.9	318.9	318.9
90	298.3	318.9	318.9	318.9
100	298.3	318.9	320.0	318.9
1000	302.2	327.5	327.7	327.5
10000	306.8	333.9	334.2	333.6
100000	311.9	339.3	348.5	339.2
1000000	317.6	342.8	352.8	342.9
1800000	320.0	345.1	353.5	344.6

6 まとめ

実験結果より提案手法はともに従来法と比較して，早い段階から良い暫定解を得ることが出来ることがわかる．また提案手法 1 は時間がたつにつれて従来法よりも暫定解の改良が進みやすいこともわかった．

今後の課題としては，より多くの部分問題を保存することや，分枝変数の選択法に関する改良が挙げられる．

参考文献

[1] William D. Harvey, Matthew L.Ginsberg, “Limited discrepancy search,” in Proceedings of the International Joint Conference on Artificial Intelligence, pp.607–615, Montreal, 1995.

[2] Satoshi Shimizu, Kazuki Yamaguchi, Toshiki Saitoh and Sumio Masuda, “Optimal Table Method for Finding Maximum Weight Clique,” in Proceedings of the 13th International Conference on Applied Computer Science, pp.84–90, Morioka, 2013.

[3] Wayne Pullan, “Phased local search for the maximum clique problem,” Journal of Combinatorial Optimization, vol.12, pp.303–323, 2006.