最大クリーク発見問題の省メモリアルゴリズム

A Space-Efficient Algorithm for the Maximum Clique Problem

○小畠教寛(1)，喜田拓也(2)
(1)北海道大学　大学院情報科学院
(2)北海学園大学，工学部

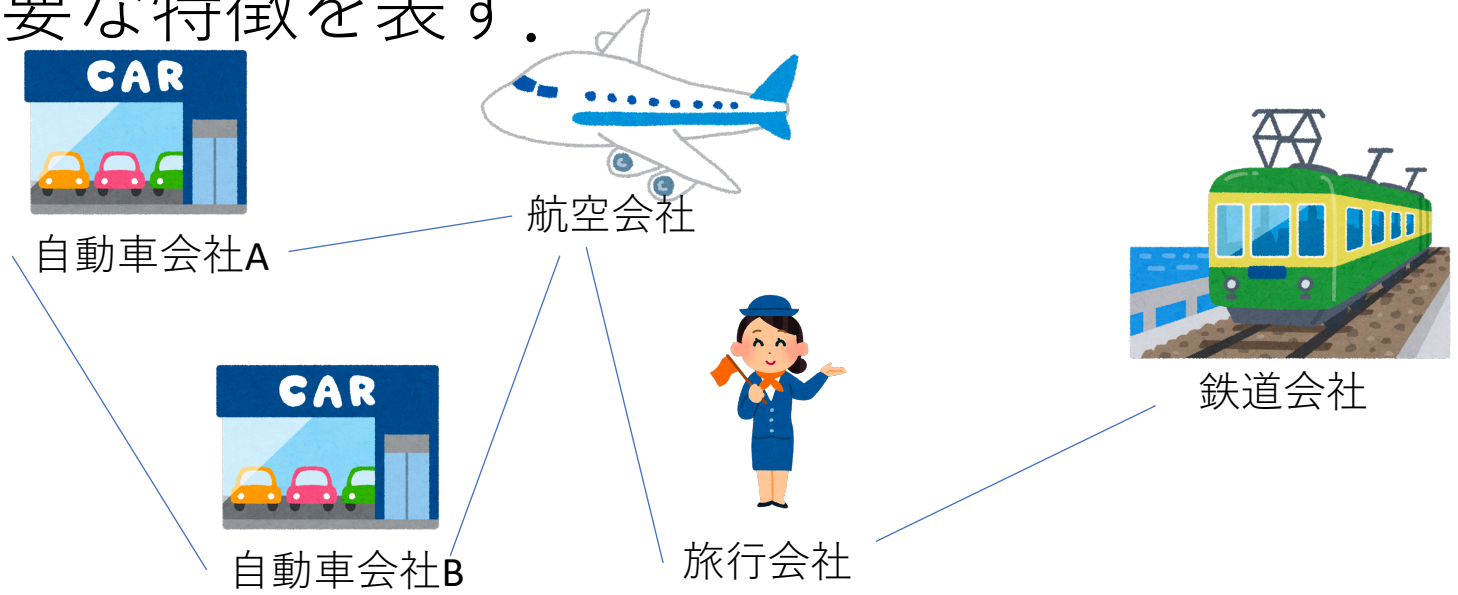
HOKKAIDO UNIVERSITY

IKN Lab

このスライドや原稿は，githubのobatakyoukan/paperに挙げています.

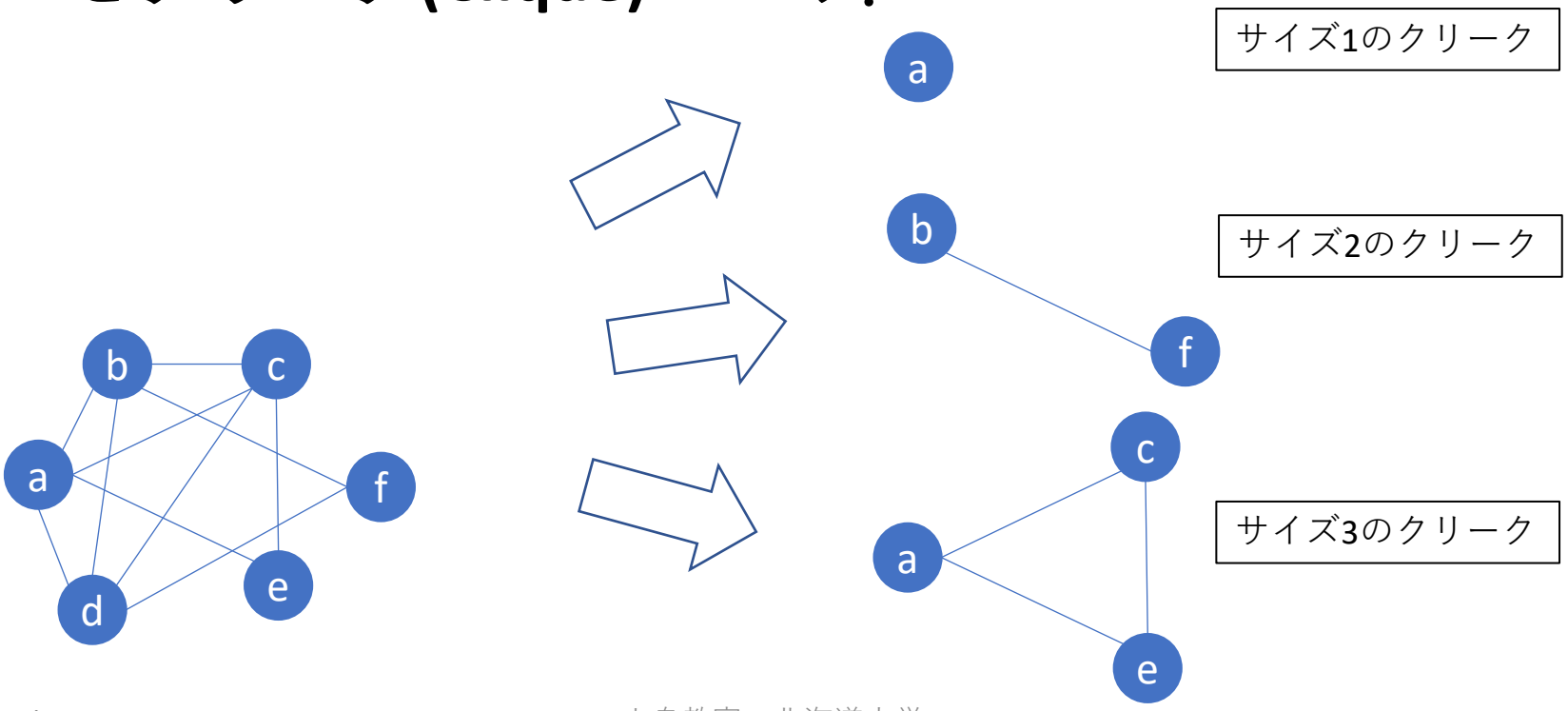
研究背景

最大クリーク問題(Maximum Clique Problem)は，幅広くの応用分野に利用でき，例えば，株式市場におけるマーケットグラフの最大クリークは，似た振る舞いをする最大のグループという重要な特徴を表す．



クリーク

無向グラフ$G = (V, E)$が与えられたとき，任意の2頂点が隣接している頂点集合Vの部分集合Qのことを**クリーク(Clique)**という．



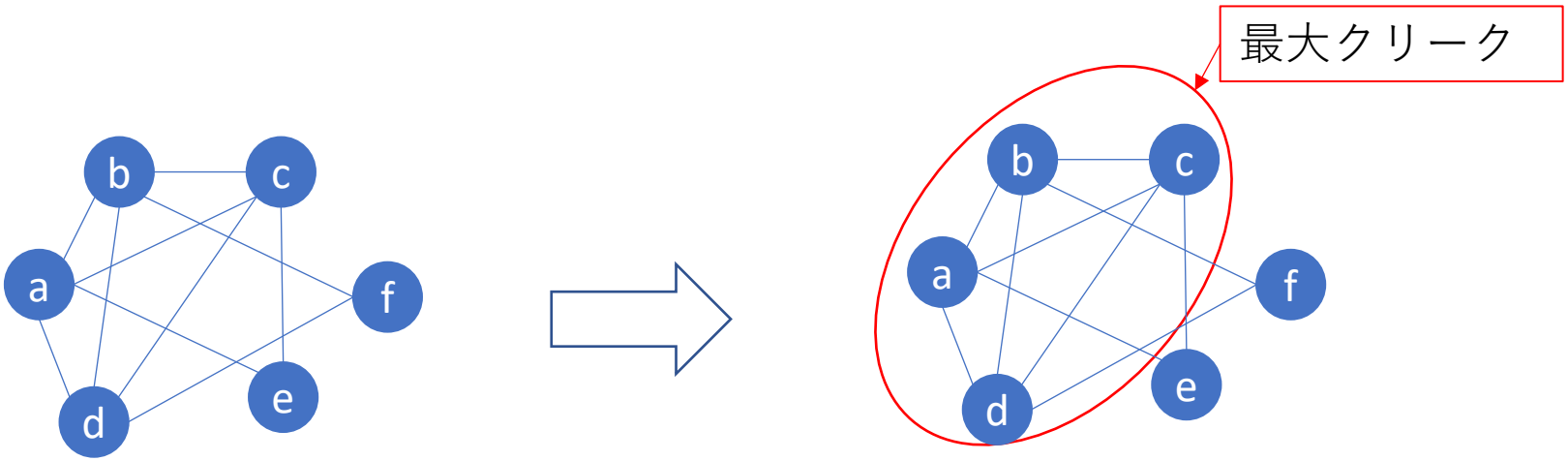
最大クリーク問題

- 入力

  無向グラフ$G = (V, E)$

- 出力

  任意の2頂点が隣接しているVの部分集合の中で，サイズが最大のものQ_{max}を**最大クリーク**といいう．



既存研究とその問題点

最大クリーク発見に関する高速なアルゴリズムは，多く研究されている．その１つとして，2010年にTomitaらによってMCSアルゴリズムが提案されている[1].

しかし，MCS は，大きなクリークサイズ ωをもつ入力グラフに対しては，その使用領域量 $O(n\omega + m)$は無視できないものになる.

ここに，グラフのサイズをn，辺の数をmとする.

[1] Etsuji Tomita,Yoichi Sutani,Takanori Higashi,Shinya Takahashi,and Mitsuo Wakatsuki. A simple and faster branch-and-bound algorithm for finding a maximum clique. In WALCOM 2010. LNCS 5942. Springer, 2010.

研究目的

本研究では，MCS アルゴリズムを，その枝刈り等の効果をできるだけ損わずに，そのメモリ量削減の改良を行うことを目指す．

主結果

- メモリ使用量を改良した**MCS-SEアルゴリズム**を提案した.
- 隣接リスト表現のとき，MCSの空間計算量$O(n\omega + m)$に対して，MCS-SEの空間計算量$O(n + m)$が，抑えられていることを示した.
  ここに，グラフのサイズをn，辺の数をm，最大クリークのサイズをωとする.
- 実験では，最大クリークが大きないくつかのグラフに対して，メモリ使用量が実際に抑えられていることを確認した.

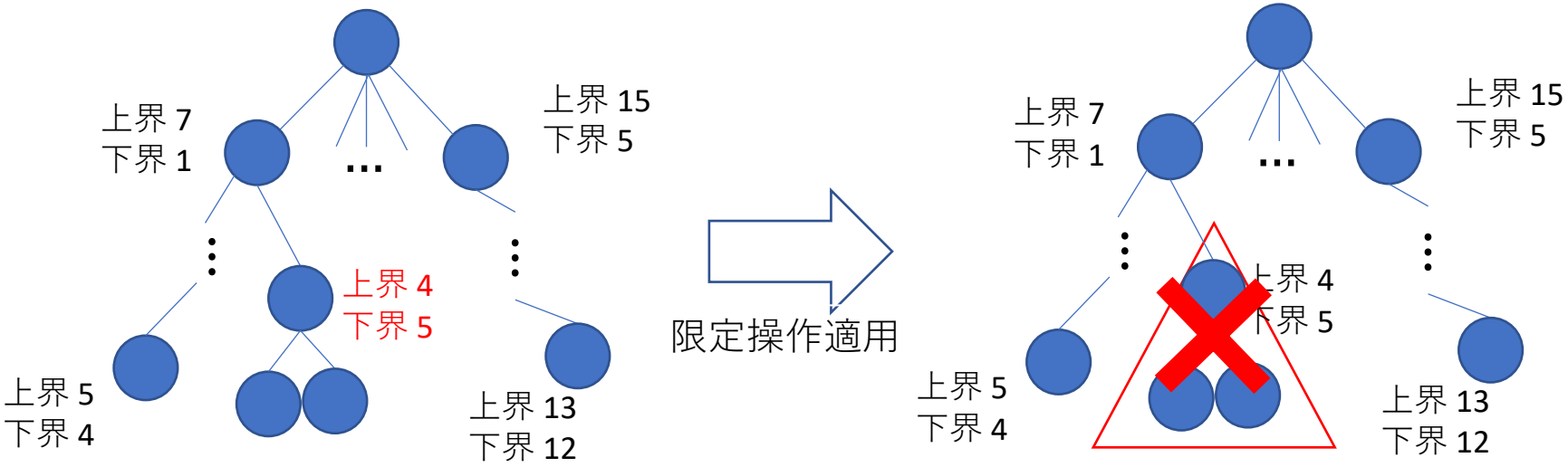
MCS

- MCS [Tomita et al.,2010]は，**分枝限定法(B&B)**によって，最大クリークを発見するアルゴリズムである.
- 以下の工夫を用いることで，探索を効率的に行っている.
  - 彩色によるクリークの上界の概算.
  - 近似解によるクリークの下界の概算.

[Tomita el al. ,2010] Etsuji Tomita,Yoichi Sutani,Takanori Higashi,Shinya Takahashi,and Mitsuo Wakatsuki. A simple and faster branch-and-bound algorithm for finding a maximum clique. In WALCOM 2010. LNCS 5942. Springer, 2010.

分枝限定法(B&B)

- 分枝操作(branching)
  部分問題に分割する手続き．これを再帰的に行う．
- 限定操作(bounding)
  部分問題の上界と下界を概算して，最適解の候補でないものの分枝操作を打ち切る(pruning).

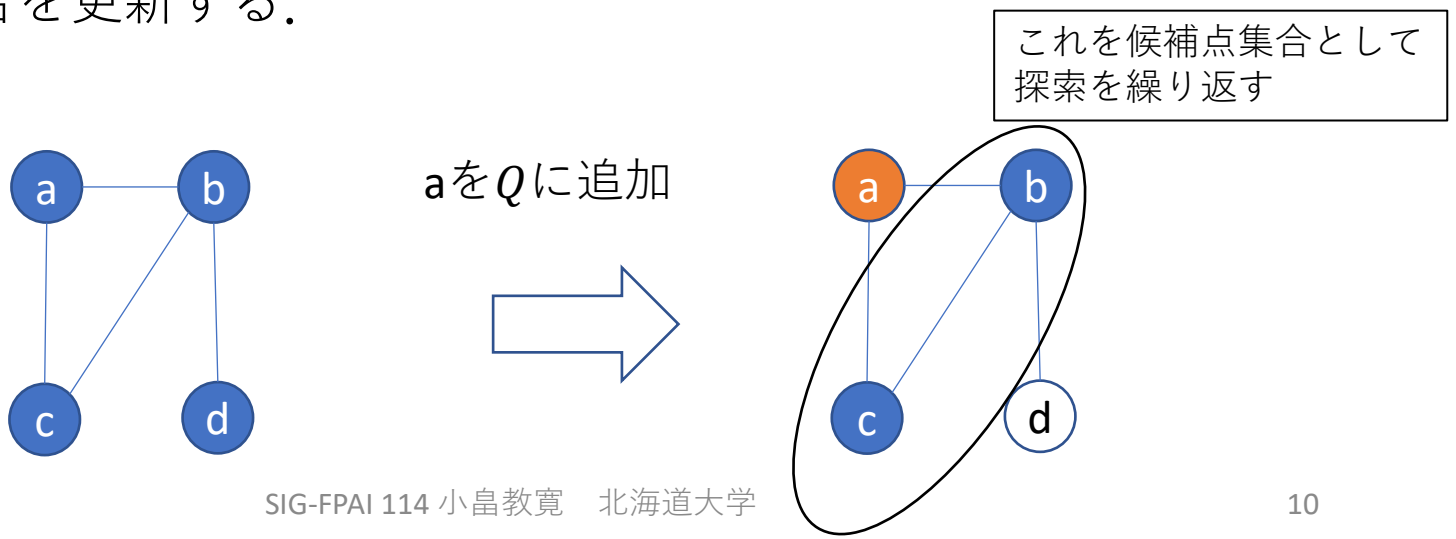


分枝操作: MCSの分枝操作手順

探索時に保持しているクリークをQとし，Qの頂点すべてに隣接している頂点集合を候補点集合とする．

分枝操作は，深さ優先探索によって，以下の操作を繰り返す．

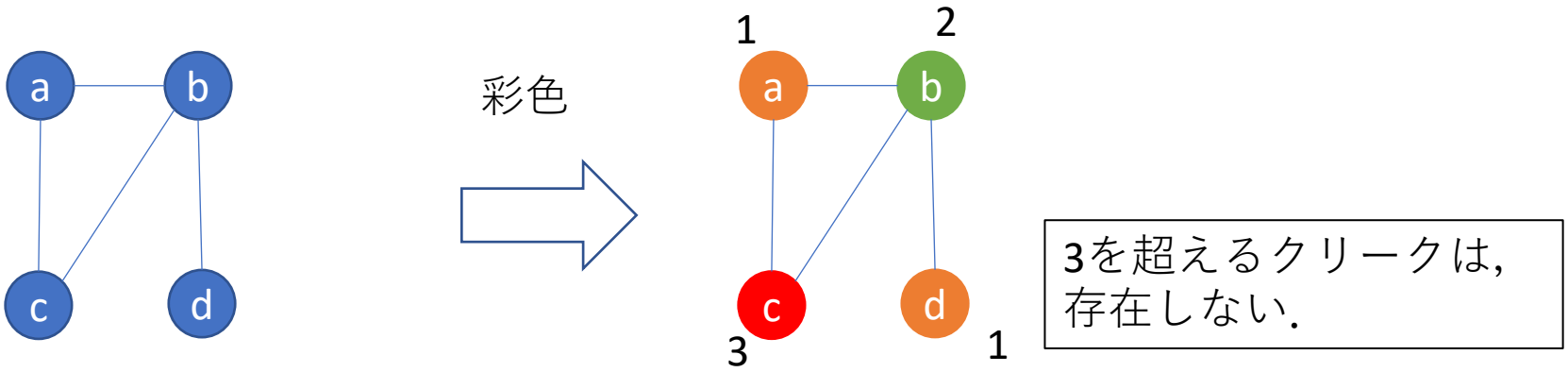
- 候補点集合の中から，1つ頂点選び，Qに追加する．
- Qが，現在までに見つかった中で最大のサイズのとき，暫定解Q_{max}とする．
- 候補点集合を更新する．



限定操作1: 彩色による上界の概算

彩色(coloring)とは，隣接する頂点に異なる正整数を振ること．

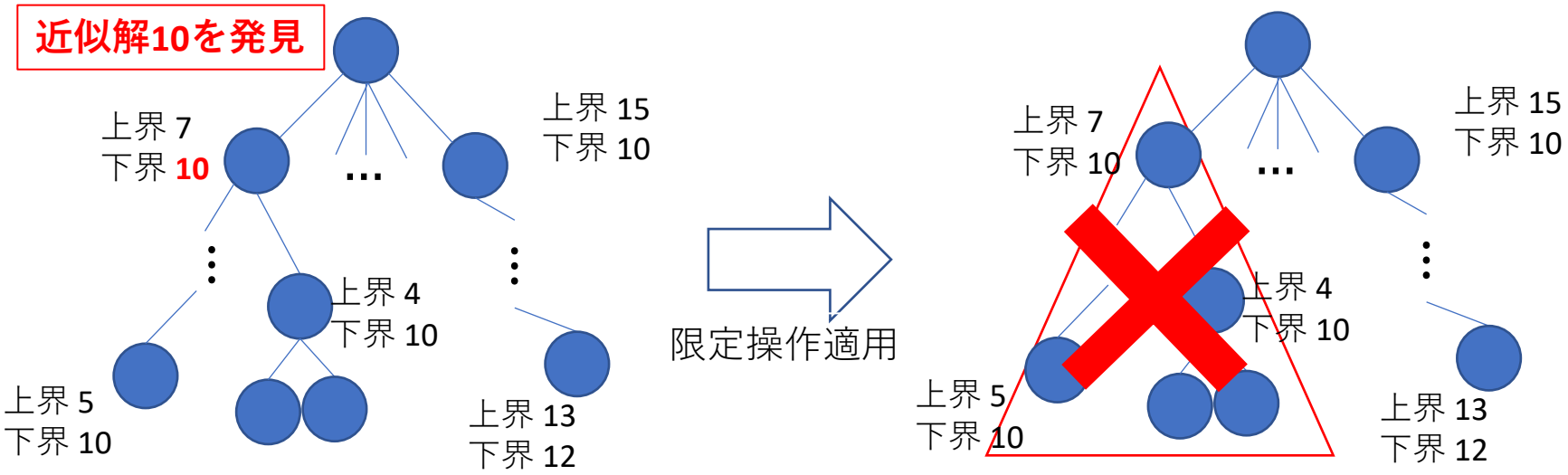
彩色に使われた最大の整数がkのとき，サイズkを超えるクリークは存在しない．



限定操作2: 近似解による探索の効率化

今回の問題では，下界は暫定解となる．何もしなければ，暫定解の初期値は0となる．

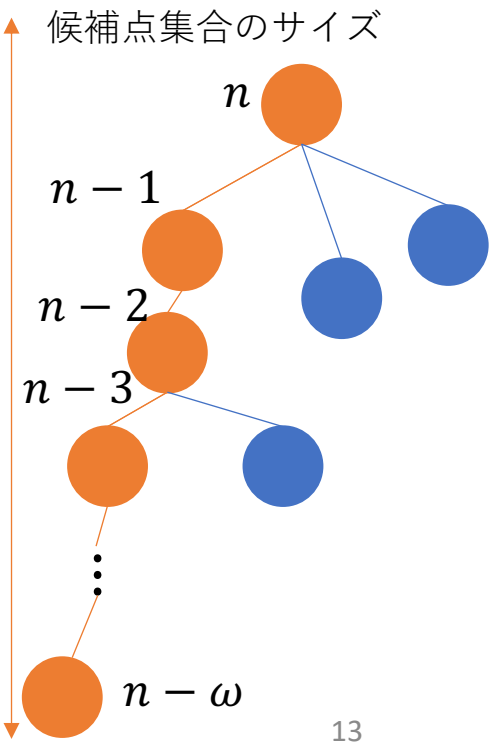
最初に近似アルゴリズムで求めた近似解を暫定解とすることで，より効果的な探索を行える．



理論解析従来MCSの空間計算量

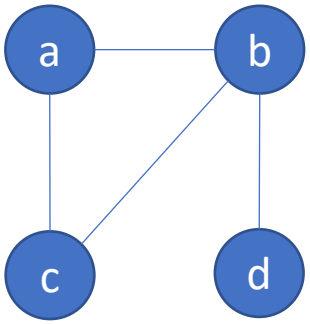
探索時に，各ノードは頂点集合とその彩色情報を保持する．
最大メモリを使用するのは，
探索の根から葉までのパスを考えればよい．
各ノードの空間計算量は，$O(n)$である．
探索の最大の深さは，
最大クリークのサイズωとなる．

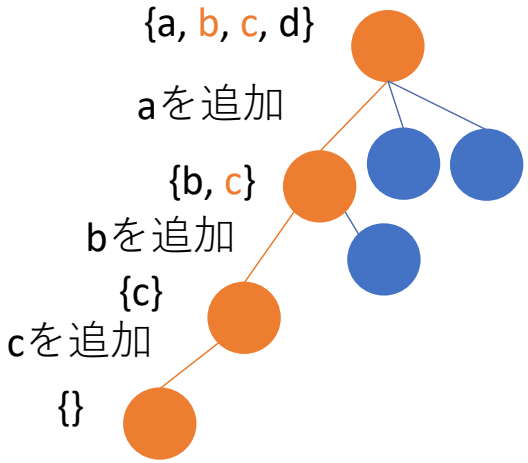
従って，隣接リストのときの空間計算量は，
$\underline{O(n+m)+O(n\omega)=O(n\omega+m)}$となる．



提案1: 改善アイデア

探索木の根から現在の繰り返しまでのパス上で,
候補点集合が集合の包含関係の下降列にという
性質を利用する.

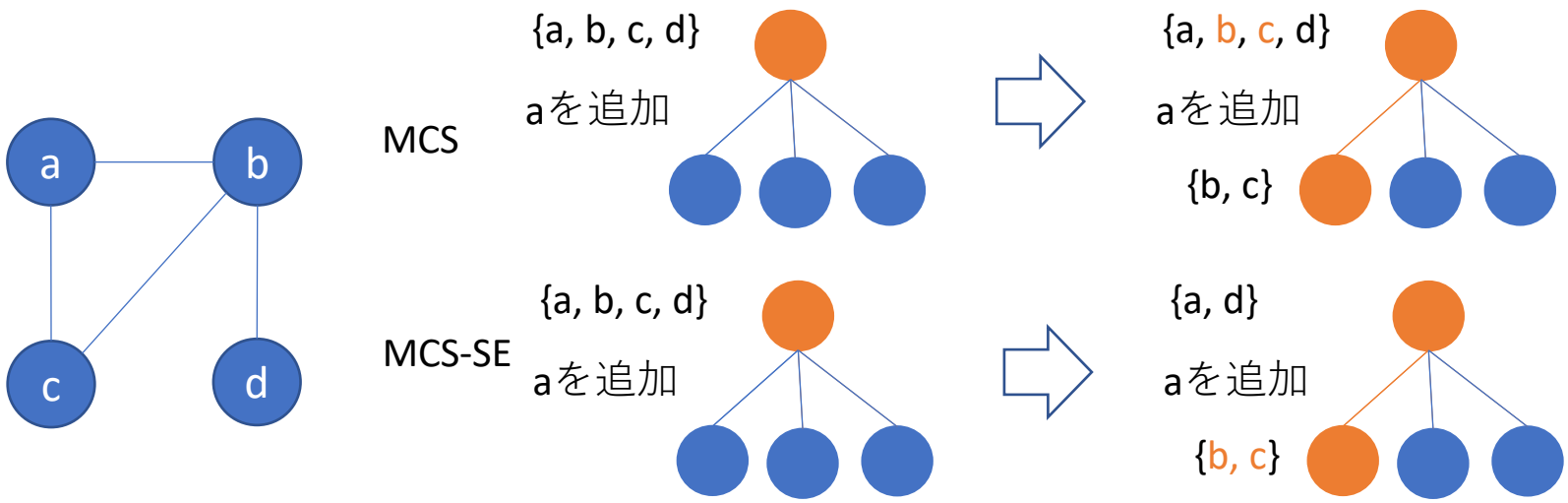




提案2: 差分表現

子に進むとき，親から子が持つ要素を削除することで，重複する情報を持たないようにする．

親は，彩色情報を保持せず，バックトラック時に，彩色情報を動的に復元する．

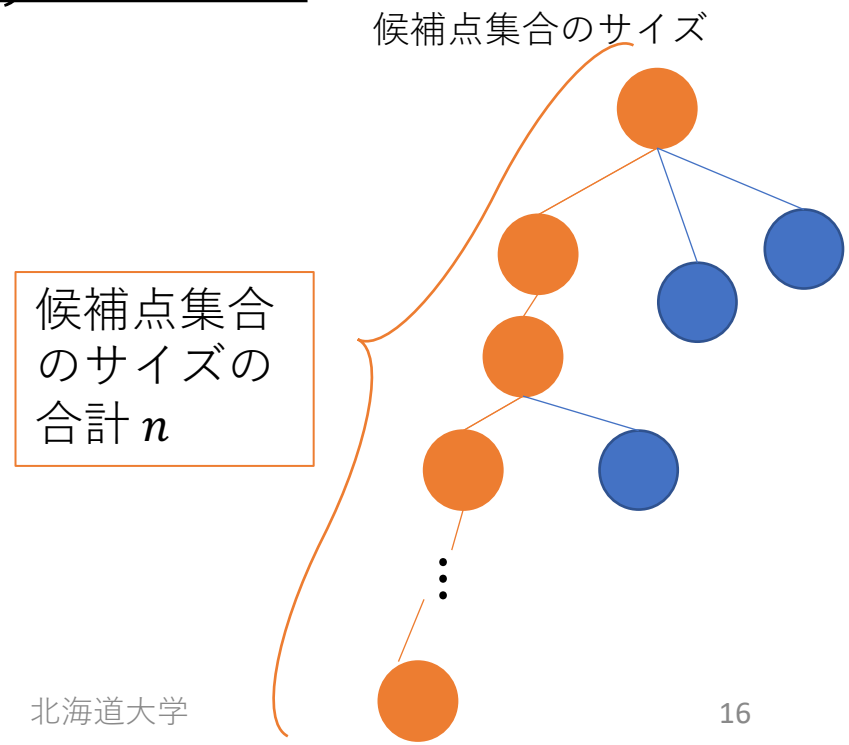


理論解析: 提案手法の空間計算量

頂点集合が，重複する頂点の情報を保持しない．

提案手法MCS-SEの隣接リストでの空間計算量は，

$O(n+m)+O(n)=O(n+m)$となる．



実験

- アルゴリズム: MCSとMCS-SEを隣接リストで実装し，近似解アルゴリズムでは，2005年にKatayamaらが提案したKLSアルゴリズムを使用した[2].
- 実験データ: (1) DIMACS ベンチマークセットのグラフ[1] と，(2) ランダムに生成したグラフ，(3) DIMACS のグラフを合体させたグラフの3種類の計35個のグラフを用意した.
- 環境: PC(CPU Intel Core i5-7360U，2.30GHz，8 GB memory，MacOSX.10.14.6) 上で，全てのプロ グラムは，gcc 8.3.0 -O3 -lm でコンパイルを行った.

[2] Kengo Katayama, Akihiro Hamamoto, and Hiroyuki Narihisa. An effective local search for the maximum clique problem. *Information Processing Letters*, Vol. 95, No. 5, pp. 503–511, 2005.

[1]http://iridia.ulb.ac.be/~fmascia/maximum_clique/DIMACS-benchmark

実験結果(最大メモリ使用量)

最大クリークの大きな一部のグラフに対して，最大メモリ使用量の減少を確認できた.

最大クリークと近似解が一致しているとき，探索が深くならない可能性がありので，最大メモリ使用量の差が小さくなる傾向がある.

入力グラフ				メモリ使用量(KB)	
グラフの名前	グラフサイズ	解	近似解	MCS	提案手法
MANN-a45	1035	345	344	1628	936
hamming10-2	1024	512	512	928	928
3000-2000	3000	2050	2050	1352	1076
union	1235	347	345	2272	1440
union2	2259	522	355	5536	4584

実験結果(実行時間)

実行時間は，ほとんどのデータで悪化した.

最大メモリ使用量に関係なく，実行時間は，増加した.

入力グラフ				実行時間(秒)	
グラフの名前	グラフサイズ	解	近似解	MCS	提案手法
MANN-a45	1035	345	344	242.98	347.05
hamming10-2	1024	512	512	26.80	27.21
3000-2000	3000	2050	2050	4786.92	5114.48
union	1235	347	345	519.86	759.34
union2	2259	522	355	402.88	463.39

まとめ

- 最大クリーク発見する省メモリなアルゴリズムを提案した.

- 実験において，MCS-SEはMCSに比べて，実行時間は増加したが，最大メモリ使用量は最大クリークの大きいグラフに対して減少することが確認できた.

- 今後の課題
  - MCSとMCS-SEを切り替えることで，メモリ消費量を抑えつつ，十分に高速化を実現すること.
  - 他のアルゴリズムに提案手法を組み込むことで，メモリ消費量を改善すること.
  - 並列化を取り入れるなどして，実行速度を改善する.

ご静聴ありがとうございました.