主専攻実験 最終発表

情報科学類3年

今から話すこと

- 1-Treeを緩和問題とした分枝限定法と、その実装の工夫
- 分枝限定法と動的計画法の比較

今から話さないこと

- クリストフィード(Christofides)のアルゴリズム(奇数次の頂点の完全マッチングを2-optの近似解にしたもの)
- 以下の構築法、改善法を組み合わせた16パターンの比較

  (見やすい形でまとめていません... ログ -> http://ur2.link/SbBK)

  - 構築法: 最近隣接法、最遠挿入法、最近挿入法、貪欲法

  - 改善法: 2-opt法、comb-opt法(2-optと1.5-optをまぜたもの) 、焼きなまし法(SA)、禁断探索法(TL=20)

やっていないこと

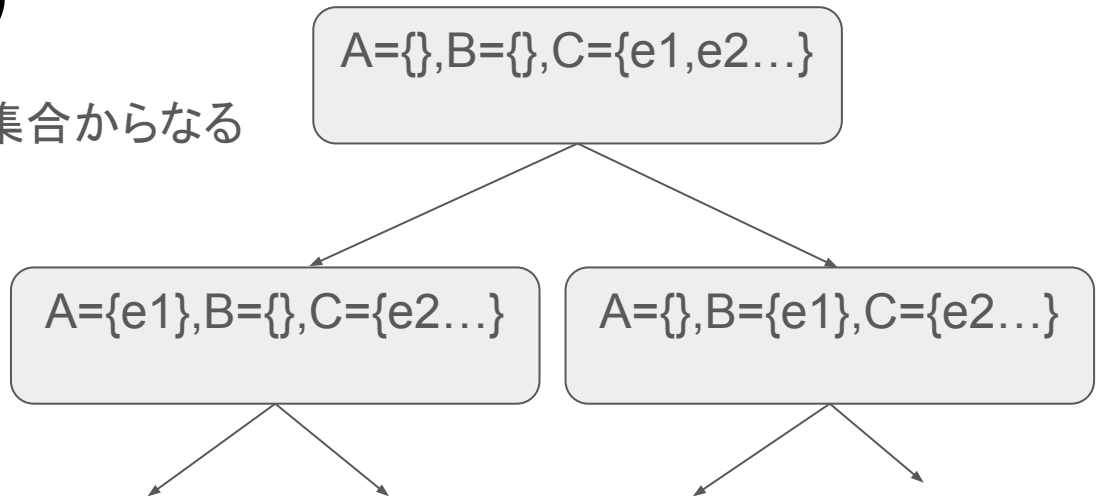
- 禁断探索法の上手なパラメータ調整
- 焼きなまし法の上手なパラメータ調整
- 保存しておく近傍の頂点の個数の上手な調整
- etc

1-Treeを緩和問題とした分枝限定法

アルゴリズムの概要(1)

探索中の状態は主に以下の3つの集合からなる

- 必ず使用する辺の集合A
- 使用を禁止する辺の集合B
- それ以外の集合C



初期状態: A={},B={},C={e1,e2…}

状態遷移:集合Cに属するある辺eを集合AまたはBにうつす

どの辺を選ぶかは後述

状態数は頂点数をNとして、枝刈りしなければ3^{(N-1)*N/2} となる

アルゴリズムの概要(2)

状態(A,B,C)から巡回路を構築不可能な例

- 集合Aに含まれる辺のうち、

  ある頂点xに接続されているものの数が3以上

- 集合AまたはCに含まれる辺のうち

  ある頂点xに接続されているものの数が2未満

ある状態から構築不可能だとわかれば、それ以降の探索を打ち切る(枝刈り)

しかしこれだけではあまり状態数を減らせない

アルゴリズムの概要(3)

この状態から最小一木(後述)をつくることで、

この状態から構築できる巡回路のコストの下界(lowerbound)を求める

これが現在の最適解(上界,upperbound)のコストより大きければ、探索を打ち切る

最適解は予め、近似解を求めて設定しておく

(無限を初期値にしてしまうと解がまだ見つかっていない時に枝刈りできない)

(今回は最遠挿入法で構築して2-opt法で改善した)

1-Tree(一木)とは

1-Tree
Definition: For a given vertex, say vertex 1, a 1-Tree is a tree of {2,3,…n} +2 distinct edges connected to vertex 1.
1-Tree has precisely one cycle.

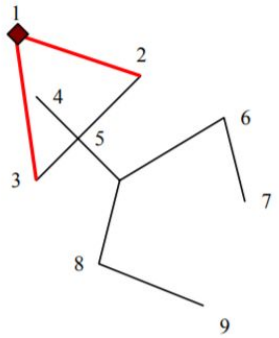


Fig. 1 Example of 1-Tree

- 頂点2...Nからなる部分誘導グラフは木
- 頂点1の次数は2

https://kaminsky.ieor.berkeley.edu/ieor251/notes/3-16-05.pdf

Minimum Weight 1-Tree(最小一木) とは

構築可能な1-Treeのうち、辺のコストの和が最小のもの

求め方

最小全域木(MST)を求めてから

頂点1を端点として持つ未使用の辺のうちコストが最小のものを追加するだけ

(プリム法やクラスカル法で簡単に求まる)

緩和問題として一木を使う

頂点1から全ての都市をちょうど一度ずつ巡り出発地に戻る巡回路は一木

順回路の辺のコストの和は、

最小一木の辺のコストの和より小さくなることはない

ことを利用して枝刈りをする

状態(A,B,C)から最小一木を求める(1)

クラスカル法を元にしたアルゴリズムで、

- 集合Aに含まれる全ての辺と、
- 集合Cに含まれる辺の一部

を使った一木のうちの、最小一木を求める

x := Aに含まれる辺のうち、頂点1に接続するものの個数

とすると、xによって場合分けされる

前準備として、Cに含まれる辺はコストの昇順にソートしておく

(頂点2...Nからなる部分誘導グラフは木になる必要があるので、頂点1が関節点にならないように注意する)

状態(A,B,C)から最小一木を求める(2)

(i) x = 2 のとき

まず、集合Aに含まれる、

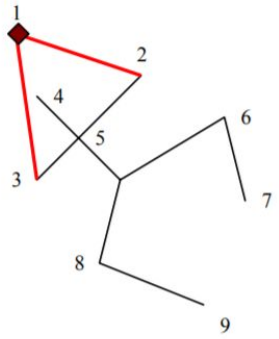
端点に頂点1を含まない辺を使う

次に

集合Cの辺で頂点2..Nの最小全域木を求める

その後集合Aに含まれる、

端点に頂点1を含む辺(2つ)を追加する



状態(A,B,C)から最小一木を求める(3)

(ii) x < 2 のとき

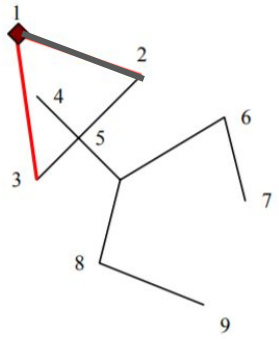
まず頂点1の次数が2以上にならないように

辺をとばしつつクラスカル法をやる

使用していない、端点に頂点1を含むCの

辺のうち、最もコストが小さいものを

追加する



状態遷移に使う辺の選び方(1)

未使用辺の集合Cを

- 最小一木の構築に使用された辺の集合C1
- それ以外の辺の集合C2

に分割して考える

(AとBが決まればC1,C2も一意に決まる)

今回は深さ優先ベースの探索にしたが、状態遷移に使う辺は(主に)

C2のうち最もコストが大きいものを禁止/使用にした(右上疑似コード)

```
void dfs(State (A,B,C)) {
  {C1,C2} := mst(A,C);
  e := max(C2)// C2のうち最もコストが大きい辺
  dfs(A,B \cup e, C \e);
  dfs(A \cup e,B, C \e);
}
```

状態遷移に使う辺の選び方(2)

なぜC2のコストが最大の辺を選んだか

- もしC1から選ぶとすると、C1に含まれる辺はコストが小さい方に偏在しているので、それを初期の分枝に使っても下界にあまり変化は起きず枝刈りできなさそうと考え、C1から選ぶ案は見送った。

状態遷移に使う辺の選び方(3)

なぜC2のコストが最大の辺を選んだか(2)

- 禁止遷移をしているうちは下界の変化がないので高速に回せる

  その時点のA,Cの辺で巡回路構築不可能になるかだけ気をつけていれば良い

  Aが更新されたときだけC1,C2を更新すれば良い

- コストが最大の辺を選んだのは、その方が使う可能性が低く、使用したとき下界に大きな変化があるので枝刈りできそうだからである。

  (頂点数7のTSPでC2の最小の辺/最大の辺を選ぶようにしたとき、状態数はそれぞれ196,36となった。C2の最小の辺を選ぶよりはC2の最大の辺を選ぶほうがいいと考えるのは正しそう)

実装の工夫(1)

各状態でなるべく計算をせずに次の状態に遷移したい

時間がかかるが必要な計算

- 状態(A,B)の更新
- 下界とC1,C2(最小一木に使用したか否か)の更新
- 次の遷移で使う辺の選択(ここは多少時間がかかっても、いいものを選ぶ)

工夫した計算

- Cに含まれる辺の走査(最小全域木で使用。コストの昇順)
- 各頂点に接続している/接続可能な辺の数の最小値など(枝刈りで使用)

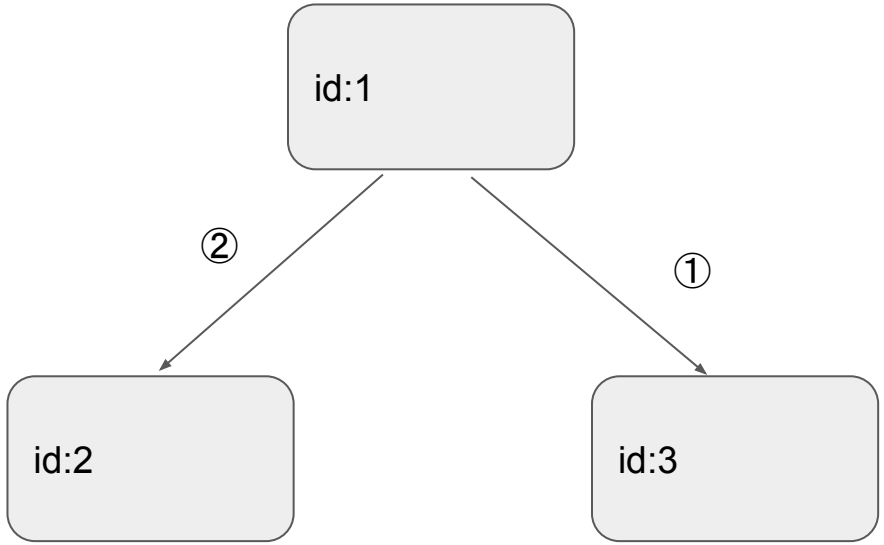
実装の工夫(2)

状態を変えたらそのログをとっておき、rollbackする

- 状態遷移とdiff管理はそれぞれstackで行う
- 各状態にidを振る(id:1の次はid:3が実行)
- 状態を変えたらそのログをstackにpush
- stackにある自分のidより

  大きいidの変更をrollbackしてから

  自身の状態に遷移する



実装の工夫(3)

状態をいくつかのデータ構造で扱う

- availabledims (RMQ)

  各頂点に隣接している辺のうち、禁止されていない本数(集合A,C)

  最小値が2未満のとき、巡回路を構築するのは不可能

- requireddims (RMQ)

  各頂点に隣接している辺のうち、必要な本数(集合A)

  最大値が3以上のとき、巡回路を構築するのは不可能

まだ巡回路を構築可能か高速に求める

実装の工夫(4)

状態をいくつかのデータ構造で扱う

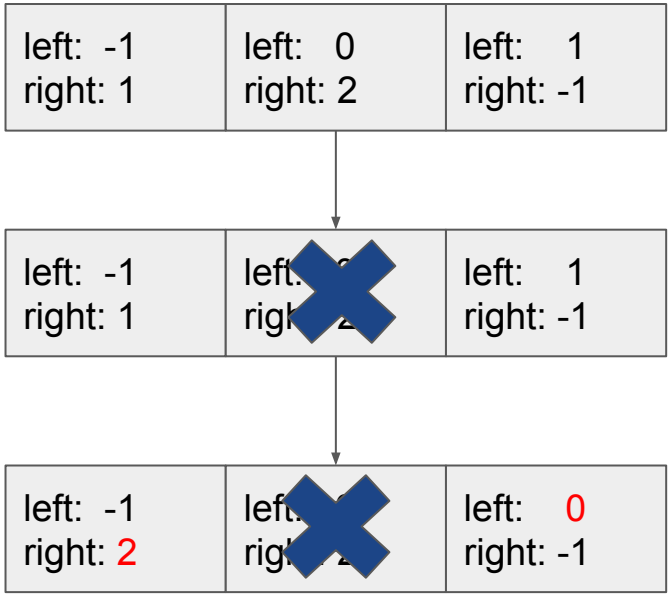
- trackneighbor(Doubly Linked Listっぽいもの)

自身の左右の辺の位置を覚えておく

AまたはBに入れられた辺は飛ばす

辺は左にあるほどコスト昇順にソートしておき、

最小全域木を構築する時は左から辿る



実装の工夫(5)

プロファイラにかけた結果

lowerboundv2()は集合Aに頂点0を端点にもつ辺が二つあるときの下界を更新する処理

selectedge()は次の遷移で使う辺を選んでいて、時間がかかる理由は、

1.C2の辺のうち、禁止or使用すると巡回路を構築できない辺を優先して選ぶ (C2の要素数だけかかる)

2. C2のコストが最も大きい辺を選ぶ

をしているから

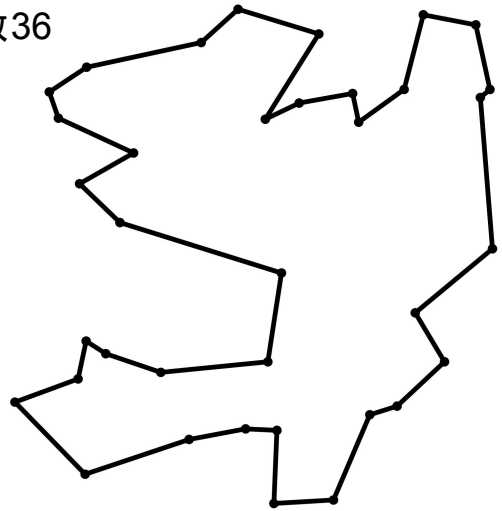
```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 19.32      0.91     0.91  1078739     0.00     0.00  State::lowerbound(StateLog&)
 14.65      1.60     0.69 54693032     0.00     0.00  RMQUndo::inc(int)
 13.17      2.22     0.62  1237755     0.00     0.00  State::selectedge()
 13.17      2.84     0.62   480204     0.00     0.00  State::lowerboundv2(StateLog&, int)
```

分枝限定法と動的計画法の比較

比較方法

- x,y座標を0~999でランダムに生成
- 頂点数を10...36にして実行時間(time コマンド)を比較した
- 疑似乱数列生成にはxor64を使用(https://ja.wikipedia.org/wiki/Xorshift)

例)頂点数36

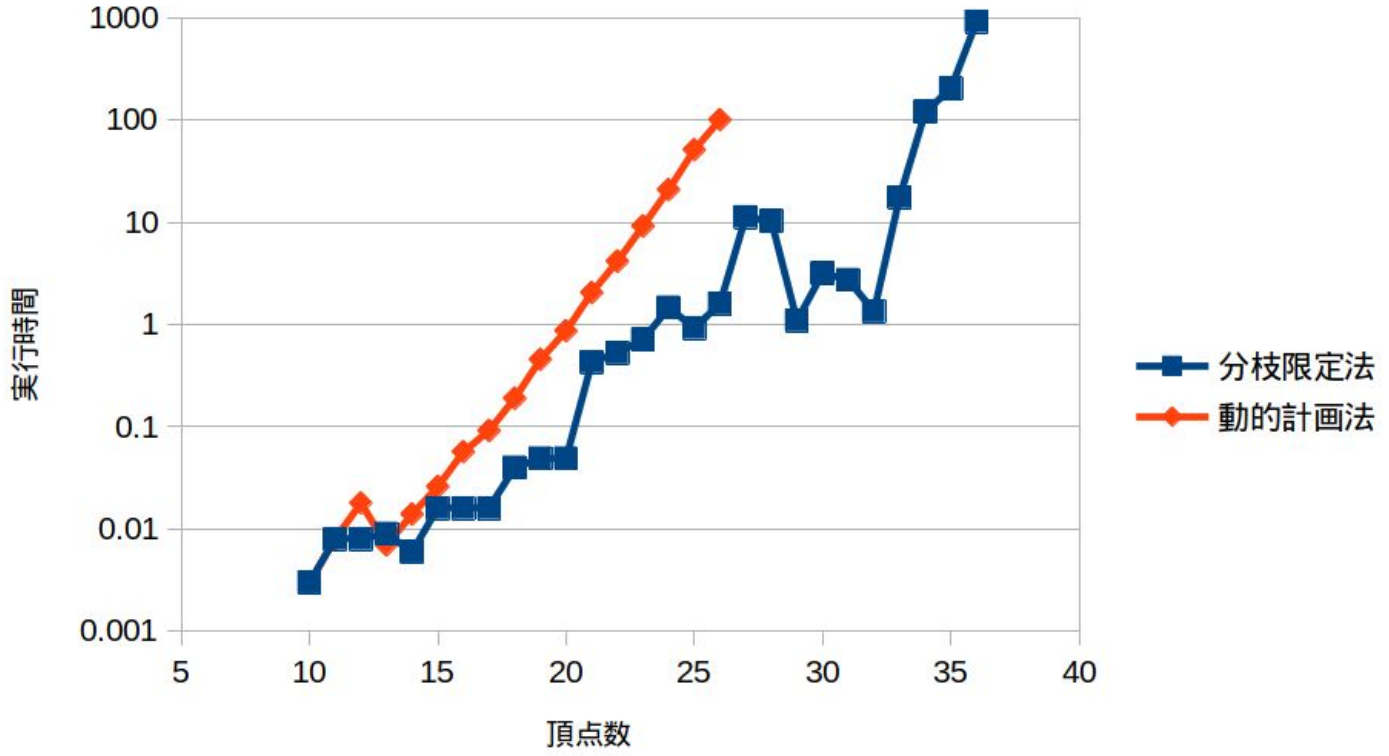


比較結果

頂点数	10	11	12	13	14	15	16	17	18	19	20	21	22	
分枝限定法	0.003	0.008	0.008	0.009	0.006	0.016	0.016	0.016	0.04	0.049	0.049	0.43	0.528	
動的計画法	0.003	0.008	0.018	0.007	0.014	0.026	0.057	0.092	0.19	0.457	0.869	2.053	4.16	
頂点数	23	24	25	26	27	28	29	30	31	32	33	34	35	36
分枝限定法	0.712	1.466	0.925	1.602	11.246	10.304	1.092	3.18	2.732	1.347	17.518	120.05	203.447	914.324
動的計画法	9.208	20.87	51.185	100.74										

比較結果

縦軸は対数


1000
100
10
1
0.1
0.01
0.001
実行時間
5
10
15
20
25
30
35
40
頂点数
分枝限定法
動的計画法

参考資料

- 情報システム評価学 ー整数計画法ー

  http://www.dais.is.tohoku.ac.jp/~shioura/teaching/dais08/dais06.pdf