アルゴリズム 16−10
（1990.7.17）

分枝限定法による最適系列分割問題

加地太一　　大内　東
北海道情報大学　　北海道大学

最適系列分割問題は系列グラフG（N，E）に対してノード番号を保存しながら、ブロックサイズPの大きさの部分集合に最適分割する問題である。この問題に対してKernighanによって開発されたダイナミックプログラミングによるアルゴリズムの実現と計算量の再検討を行なう。ここで計算量が下限となる洗練されたプログラムに対して理論的に計算量を推定し、同時に実験比較を行なう。あわせて分枝限定法による最適系列分割問題を検討する。

Optimal Sequential Partitions of Graphs By Brach and Bound

Taich Kaji[+]　Azuma Ohuchi[++]

[+] Department of Information Science
Hokkaido Information University
59-2, Nishinopporo, Ebetu, Hokkaido, Japan
[++] Department of Information Engineering
Hokkaido University
N-13,W-8, Sapporo, Hokkaido, Japan

Optimal sequential partitions of graphs is finding a minimum cost partition of nodes of a graph into subsets of a given size, subject to the constraint that the sequence of the nodes may not be changed. We discuss some important point's implementation of Kernighan's dynamic programing algorithm. We estimate theoretical time complexity and compare it with numerical computation. We also propose a Branch and Bound algorithm for the problem.

1．はじめに

系列グラフG（N，E）の最適系列分割問題はノード番号の系列を保存しながら、部分集合のノードウェイトの総和がブロックサイズP以下になるようにノード集合を分割し、カットされるエッジのコストの和が最小となるように分割する問題である。

この問題に対してKernighan[1]はダイナミックプログラミング（DP）を用いる効率的なアルゴリズムを開発し、その計算量がエッジ数に線形比例するものと述べている。本論文ではKernighanのアルゴリズムの実現について検討し、計算量についての再検討を試みる。ここで計算量の理論的推定を行い、計算量がエッジ数のみならず、ノード数とブロックサイズにも依存することを示す。また計算実験により理論値と実測値の比較を試みる。さらにブロックサイズの影響を抑える手法として分枝限定法（B＆B）によるアルゴリズムを同時に提案する。

2．最適系列分割問題

ノード集合N＝{1、2、・・・、n}、エッジ集合E＝{e_1、e_2、・・・、e_r}からなる有向グラフをG＝（N、E）とする。各エッジは非負のコストC＝{c_1、c_2、・・・、c_r}をもち、各ノードサイズ{w_1、w_2、・・・、w_n}を各々$0<w_i\leqq p$とする。ここでPはブロックサイズと呼ばれる正の数である。またノード集合Sのサイズは$|S|=\sum_{j\in S}w_j$とする。

最適系列分割問題とは、以下の制約のもとでGをk個の部分集合G_1、G_2、・・・、G_kに分割し、そのときカットされるエッジのコストの総和を最小とするもんだいである。

i＝1、2、・・・、kについて

（1）$|G_i|$＝部分集合のノードの重みの総和≦P、

（2）任意のG_iの各ノード番号は連続的な番号をもつ。

部分集合G_iはブロックと呼ばれG_i＝{j，j＋1，・・・，m－1，m}となる。G_iでの一番小さなノード番号をb_iとして、ブレイクポイントと呼ぶこととする。b_i＝jはノードj－1とノードjの間でカットすることを意味する。集合{b_1，b_2，・・・、b_k}は一意的に分割{G_1，G_2，・・・，G_k}を表現する。同時にグラフのノード集合に対して、ブレイクポイントを1、以外を0としたビットパターン表現{p_1，p_2，・・・，p_n}でも表現できうる。

ここで、グラフGが有向方向である場合、（i，j）と（j，i）に対して、i＜jである単一エッジ（i，j）におきかえ、コストC'ij＝Cij＋Cjiとする。

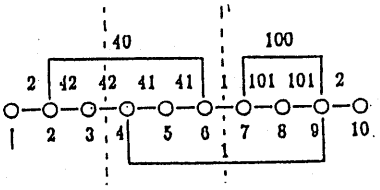

図1

図1は連続的なノードN＝{1，2，・・・，10}であり、上部の数字のコストをもつエッジからなるグラフである。また各ノードのウェイトは1とする。図1のグラフに対してブロックサイズP＝4で、この問題の最適解を求めた場合、同図の破線でカットされる。このとき、各部分集合は{1，2，3，4}，{5，6}，{7，8，9，10}となり、ブレイクポイントの集合は{1，5，7，11}となり、ビットパターン表現では{1，0，0，0，1，0，1，0，0，0，1}となる。また、最小となるコストの総和は83である。

3．KernighanのDPアルゴリズム

以上の問題に対してKernighanはダイナミックプログラミングを適用した。以下にアルゴリズムの概略を記す。

この問題に対してブレイクポイントxにおける最良な部分解をT（x）とすると、T（x）は以

下の多項式で求められる。

$T(x)=\min_y \{T(y)+C(x,y)\}$

yはxの前のブレイクポイントであり、”yからx－1の距離≦P”の範囲である。C（x，y）は前のブレイクポイントyに対して、次のブレイクポイントがxとなる時のコストの増分である。

この多項式を用い順次求まる最適部分解と前のブレイクポイントをポインターとして保存しておく。最終解に到達することによって保存してあるブレイクポイントから分割が求まる。

Kernighanはこのアルゴリズムを用いることによってランニングタイムがエッジ数に線形比例することと述べている。

定理：Procedureの実行時間はグラフのエッジ数に線形比例する。

4．アルゴリズムの具体化

4．1　定理の実現

この定理を具体化するために以下のような算法を試みた。

T（x）の計算のために使われる増分コストのデータはC（x，x－1），C（x，x－2），・・・，C（x，1）である。これらの値は、以下の漸化式を用いることによって求めることができる。

$$C(x,y)=C(x-1,y)+(C_{x-1,x}+C_{x-1,x+1}+\cdots+C_{x-1,n})-(C_{y,x-1}+C_{y+1,x-1}+\cdots+C_{x-2,x-1})$$

このことによって、一つ前のステップ（T（x－1）を計算するステップ）で使われたデータC（x－1，x－2），C（x－1，x－3），・・・，C（x－1，1）から上の漸化式を用いて計算可能である。但し、C（x，x－1）だけは新しく計算しなければならない。

C（x，x－1），C（x，x－2），・・・，C（x，1）の計算例を以下に示す。

$$C(x,x-1)=(C_{x-1,x}+C_{x-1,x+1}+\cdots+C_{x-1,n})$$

$$C(x,x-2)=C(x-1,x-2)+(C_{x-1,x}+C_{x-1,x+1}+\cdots+C_{x-1,n})-(C_{x-2,x-1})$$

$$C(x,x-3)=C(x-1,x-3)+(C_{x-1,x}+C_{x-1,x+1}+\cdots+C_{x-1,n})-(C_{x-3,x-1}+C_{x-2,x-1})$$

・　・　・

$$C(x,y)=C(x-1,y)+(C_{x-1,x}+C_{x-1,x+1}+\cdots+C_{x-1,n})-(C_{y,x-1}+C_{y+1,x-1}+\cdots+C_{x-2,x-1})$$

・　・　・

$$C(x,1)=C(x-1,1)+(C_{x-1,x}+C_{x-1,x+1}+\cdots+C_{x-1,n})-(C_{1,x-1}+C_{2,x-1}+\cdots+C_{x-2,x-1})$$

上記で求めたC（x，i）を配列に順次、更新格納していく。この操作によってTを評価する段階でおのおののエッジは2度計算され、増分コストを求める手続きにおいてはオーダーが線形比例するものと考えられる。さらに、T（x）を求めるためには少なくとも、ブロックサイズPの範囲であるC（x，x－1）、C（x，x－2）、・・・、C（x，x－p）のみ計算すればよく、効率をさらに上げることができる。このような計算法をとることによって、増分コストを求める手続きにおいてKernighanの定理に従うアルゴリズムが構築される。しかし、DPアルゴリズム構築のためには、増分コスト計算のみならず、他に最低限必要な計算部分が存在する。

4．2　アルゴリズムの実現

このDP問題に対しての下限の計算量を推定してみる。DPを動かすために最低限、以下の作業が必要である。

1）増分コストを求める。

2）各々のT［x］，L［x］を求める。

この2点が基本構造である。したがってこの2点の手続きの計算量を求めることによって真のDPの計算量が求まる。著者等が作成したプログラムでは以下の手続きによって構成される。

ｓｔａｒｔｃｃ　：　$C_{x-1,i}$（ｉ≧ｘ）を求める。
ＣｏｌＶｅｃ　：　$C_{i,y}$　（ｉ≦ｙ）の各点ｉを配列に格納する。
ｆｏｒｍｃｃ　：　前情報Ｃ（ｘ－１，ｉ）から新たにＣ（ｘ，ｉ）を作成する。
ＳｅｔＴＬ　：　ブレイクポイントｘに対するＴ［ｘ］，Ｌ［ｘ］を求める。
ＤＰ　：　ブレイクポイント２≦Ｘ≦ｎ＋１の範囲におけるＴ［ｘ］、Ｌ［ｘ］を計算する。

手続きの構成は図２になる。

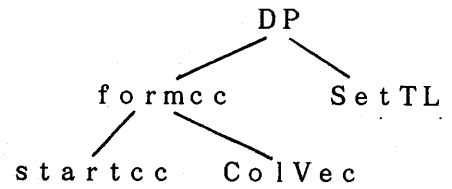

図２

また、各手続きのアルゴリズムは以下のようになる。

｛ｓｔａｒｔｃｃ手続き｝

```
W=0;
for (j=O[x-1]; j>0; j=E[j].back)
  W += E[j].cst;
return W;
```

｛ＣｏｌＶｅｃ手続き｝

```
for (j=D[y]; j>0; j=E[j].forw)
  col[E[j].org] = E[j].cst;
```

｛ｆｏｒｍａｃｃ手続き｝

```
outd = startcc(x);
cc[x-1] = outd;
ColVec(x-1,V);
ind = 0;
for (y=x-2; y>0 && y>=x-p; y--) {
  ind += V[y];
  CC[y] = CC[y] + outd - ind;
}
```

｛ＳｅｔＴＬ手続き｝

```
minval = MAXVAL;
for (y=unitf(x-p); y<x; y++) {
  val = T[y] + CC[y];
  if (val < minval) {
    minval = val;
    miny = y;
  }
}
T[x] = minval;
L[x] = minx;
```

｛ＤＰ手続き｝

```
for (x=2; x<N+2; x++) {
  formcc(x);
  SetTL(x,p);
}
```

以上のような最低構成要素から、計算量を求めることによってＤＰの計算量が求まる。しかし、各手続きにおいてより洗練したプログラムでなければ、計算量的に余計な負担を増すこととなる。

プログラムの効率化のために、グラフ構造を表現するものとして、無駄なアクセスを行なわないよう隣接行列を用いず、リスト構造をもちいてデ

ータ処理する。

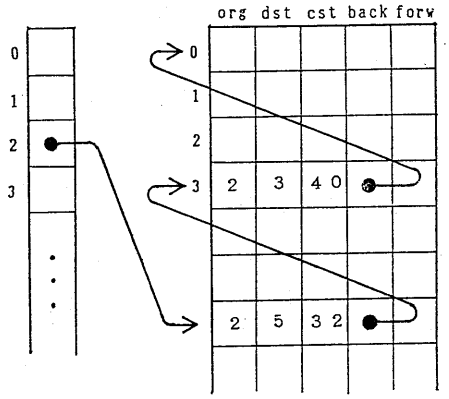

図３：先ノードの任意の同番号に対してのリスト
org:先ノード
dst:後ノード
cst:エッジコスト

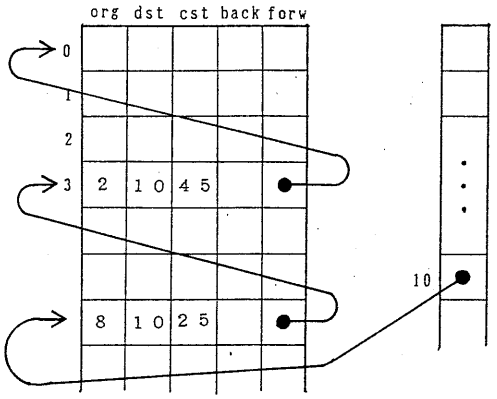

図４：後ノードの任意の同番号に対してのリスト

任意のノードを先ノードｉとして、ｉとエッジでつながる後続のノードを後ノードｊとする。（ノード番号がｉ＜ｊとなる。）　ｉとｊおよびそれを結ぶエッジのコストをグラフ構造を表現する１つの組み合せとして格納する。増分コストを計算する段階においてエッジのコスト$C_{x-1,j}$（ｘ≦ｊ）を参照するために、同じ先ノードをもつ組み合せをリスト構造でつなぎあわせる。同様に$C_{i,x-1}$（ｉ≦ｘ－２）を参照するために、同じ後ノードをもつ組み合せをリスト構造でつなぎあわせる。図３、４にそのデータ構造を示す。

以上のような工夫をほどこすことによって計算量を比較できうるものとしての洗練されたプログラムとなる。

４．３　計算量の評価

以上より基本構成部（startcc, ColVec, formcc, SetTL, DP）からなる各手続きの計算量を求める。なお　各処理の計算時間を以下の記号で表記する。

τ（＝）　：　代入処理
τ（＜）　：　判断処理
τ（＋）　：　加算処理
τ（－）　：　減算処理
τ（[]）　：　１次元配列にアクセスするのに必要な時間
τ（[].s）：　１次元配列のストラクト構造にアクセスするのに必要な時間

１）ｓｔａｒｔｃｃにおける計算量

```
W=0;
for (j=O[x-1]; j>0; j=E[j].back)
  W += E[j].cst;
return W;
```

なお、ループ構造は図５に示す流れとして考えるものとする。

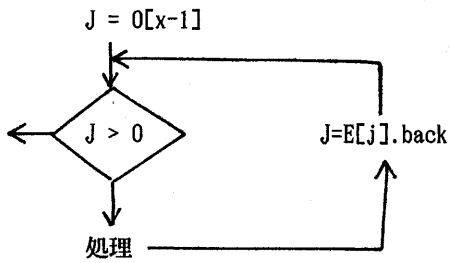

図５

ｓｔａｒｔｃｃの計算量

$=\tau(=)+\tau([])+\tau(=)$
$+\tau(<)+\tau(処理_1)+\tau([].s)+\tau(=)$
$+\tau(<)+\tau(処理_2)+\tau([].s)+\tau(=)$
　　・　・　・
$+\tau(<)+\tau(処理_{od(x-1)})+\tau([].s)$
$+\tau(=)+\tau(<)+\tau(=)$

このループの回数はｏｄ（ｘ－１）である。ただし，ｏｄ（ｖ）は頂点ｖの出次数となる。

また、処理$_i$の計算量は$\tau([].s)+\tau(=)$となるので、

$=od(x-1)\cdot(2\tau([].s)+2\tau(=)$
$+\tau(<))$
$+3\tau(=)+\tau([])+\tau(<)$

以上のような計算法をとることによって、ColVec, formmcc,SetTL,DPの計算量を求める。ここでＤＰ手続きの計算量の値が本問題における計算量の基部となり、下限となる理論的推定値と一致する。

ＤＰの計算量$=\mu|E|+\eta N+\lambda NP+\Phi$

この問題に対してKernighanはアルゴリズムの計算量はエッジ数に線形比例すると述べているが、それは増分コストの計算部においてであり、アルゴリズム全体を考えると上式のようにＮＰに大きく依存する。したがってデータが大型化するにしたがってＮはもちろんＰも大きな値を使う必要があり，ＮＰの計算量は無視できなく、Ｐとともに増加する傾向がみられる。

また隣接行列部におけるデータ構造を２次元配列で扱うことによって、ＤＰの計算量は以下の値となる。

ＤＰの計算量$=\mu|E|+\lambda N^2+\eta N+\lambda NP$
$+\Phi$　　（配列型を用いた）

このようにオーダーはN^2 に依存してしまう。これは隣接行列におけるスパース部も計算過程において参照するためである。

５．実験結果

現段階の実験では　EPSON(PC-286)を用い実測を行なった。ただし、ハード的に秒単位以上しか測定できないので秒以下の時間と、実際のプログラムにおける種種の雑損となる計算量を考慮しなければならない。

（１）ノード数による変化

ノード数を１００から１０００まで増大させたときの計算時間を図６に示す。ただしブロックサイズは４、エッジ数はノード数の約２倍の割合とする。実験結果ではほぼ理論的推定値から導かれるようにノード数に線形比例する。

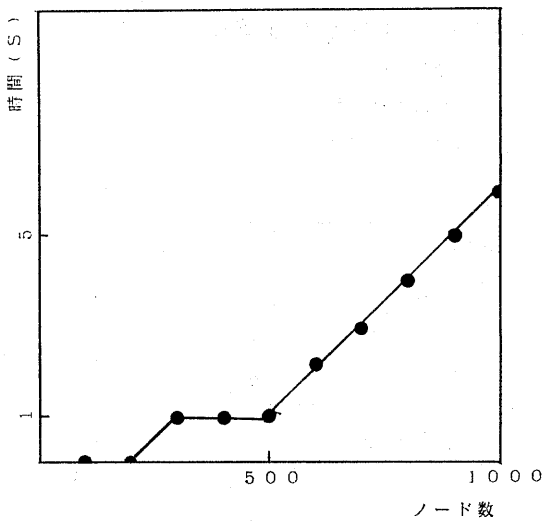

図６：ノード数による変化

（２）ブロックサイズによる変化

ノード数を一定にしてブロックサイズを増大させたときには図７のような結果が得られた。ここではエッジ数がノード数の約２倍の割合でノード数が１０００と２０００の２つの場合について調べてみる。ノード数が２０００の場合、ノード数１０００の場合より増加の傾斜が大きくなり、ノード数とブロックサイズの相乗的増加傾向を示す。

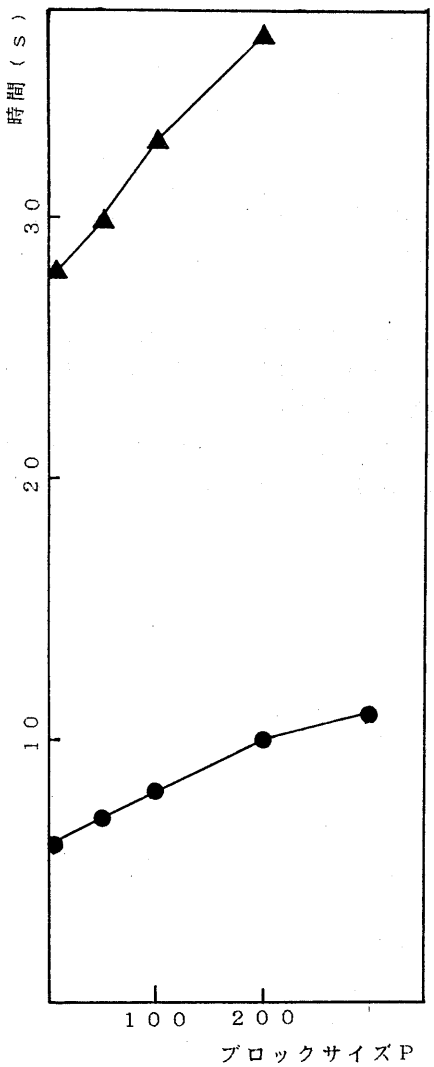

図7：ブロックサイズによる変化

（3）エッジ数による変化

現実験ではノード数５００のグラフに対してブロックサイズを２０とし、エッジ数を１０００から１００００まで変化させた結果、時間はいずれも3秒と変化を示さなかった。（メモリーサイズがヒュージ形）　これはエッジを参照する計算量が変化を示さないような値の範囲であるものと考えられる。

６．分枝限定法の提案

Kernighanによる手法はノード数とブロックサイズが相乗的に増加傾向を示すため、大型データを扱う問題には不都合な点が生じる。そこで我々は分枝限定法を用いブロックサイズの変化に対して影響が少ないアルゴリズムを構築する。以下にその概要を示す。

分枝限定法は最適化問題を分岐と限定をくりかえして解く手法である。Ｂ＆Ｂ法は以下の６つのパラメーター（Ｂ，Ｓ，Ｅ，Ｄ，Ｌ，Ｕ）によって構成される。

（1）分枝規則Ｂ

分枝規則Ｂは分割問題の派生過程を定義する。派生過程において、いかに探索空間を小さくするかが問題である。分枝規則Ｂとしてはノード系列の初期値を開始点として次の２つの性質により順次展開していく。

性質１．各ブロックの大きさがＰを越えてはならない。

性質２．あるブロックを細分化すればコストへの寄与が大きくなる。ゆえに探索木の中で、あるブロックとそのブロックを細分化した状態が生じるとき、細分化した状態をコスト最小化の立場で無視してよい。

１）と２）の規則より次の不等式が導ける。

$BP_{i-1}+P \geqq BP_i > BP_{i-2}+P$

（ノードウェイト＝１のときの不等式）

BP_i　：　ｉ番目のブレイポイント

Ｐ　：　ブロックサイズ

以上の不等式にしたがってブランチングノードを生成する。

（2）選択規則Ｓ

選択規則Ｓは現在のアクティブノードから次のブランチングノードを選択する探索戦略である。ここではこの問題に効率のよい最良下限探索を採用する。

最良下限探索は探索過程の現時点でコストの値が最も低いノードを選んで進んでいくバックトラック法である。その時点までに得られているノードの中から、最も目標に近いノードを選んで展開する。

（3）優劣関係D

部分解π_1と部分解π_2に対して、π_1、π_2を先祖とする完全解の中でコストが最小なものを比較し、2つのノードの優位性をみる。

もし　ｍｉｎ｛π_1を先祖とする完全解のコスト集合｝＜ｍｉｎ｛π_2を先祖とする完全解のコスト集合｝なら　$\pi_1 D \pi_2$となる。この問題において以下の2つの優劣関係が成立する。

1）（部分解π_k）D（部分解π_kのブロックG_tを細分化した部分解）

2）確定部の長さが等しい部分解π_a、π_bにおいて、π_aのコストがπ_bより低ければ

$\pi_a D \pi_b$

（4）下限値関数L

L（π）は各部分解πに対しての完全解のコスト集合の下限となる実数値を表わす。ここでは各部分集合の下限値は確定部においてブレイクポイントによって切られるエッジのコストと未確定部におけるブロックサイズPを越えるエッジのコストの総和とする。

（5）上限値U

Uは今現在知られている完全解のなかで最良なコストの値である。

（6）削除規則E

Eは上限値やドミナンスリレイションを用いてノードを削除する方を定義する。探索過程において、L（π_1）＞Uまたは$\pi_2 D \pi_1$の場合、π_1を削除し、探索空間を縮小させる。

7．おわりに

以上よりkernighanのＤＰアルゴリズムにおいて、計算量はエッジ数のみならずノード数およびブロックサイズに依存することが示された。またデータが大型化するにしたがい計算量はノード数とブロックサイズによって相乗的に増加する。

今後の研究課題として、ＤＰにおける大型化されたデータの計算量の変化を調べ、より洗練された分枝限定法アルゴリズムを構築し、比較実験を行ないたい。

参考文献

[1]Brian.W.Kernighan, Optimal Sequential Partitions of Graphs, J.ACM,Vol.18,No.1,(1971),pp.34-40.

[2]Walter.H.Kohler, Characterization and Theoretical Comparison of Branch-and-Bound Algorithms for Permutation Problems, J.ACM, Vol21,No.1,(1974),pp.140-156.

[3]加地、大内、加地、Branch-and-Bound法によるグラフの最適系列分割問題、ＯＲ学会春季研究発表会,(1990),pp198-199.

[4]加地、大内、加地、グラフの最適系列分割アルゴリズムの改善、情報処理学会第４０回全国大会,(1990),pp61-62.