数理工学実験～組合せ最適化～

2 年 1029313168 南里昌哉

実験日時 1 月 4,5,12,18 日　於. 自宅, 総合校舎 202
提出日時 1 月 24 日

1　はじめに

組合せ最適化は解が離散的であったり、解を要素の組合せの形で表すことができたりする類の最適化問題である。こうした問題は多種多様に存在するが、問題の構造を的確に把握しなければ効率的な解法が得られない。今回の実験では、組合せ最適化の代表的な問題である最短路問題を、汎用性の高い代表的な解法の１つである分枝限定法で効率的に解決する方法を探る。

2　原理・方法

2.1　最短路問題

節点集合 $V = \{0, 2, .., N-1\}$ と有向枝集合 $E \subseteq V \times V$ からなるグラフ $G(V, E)$ と各枝 $(u, v) \in E$ の長さ $d(u, v)$ が与えられている状況を考える。この時、$(v_i, v_{i+1}) \in E, i = 0, 1, 2, .., k-1$ を満たす節点の列 $P = \langle s = v_0, v_1, v_2.., v_k = t \rangle$ を始点 s から節点 t への路と呼び、この路に含まれる枝の長さの和を長さ $l(P)$ という。最短路問題は、この $l(P)$ を最小にする路を見つけるものである。今回の実験では、対象のグラフに負の長さを持つ枝が存在している状況で、単純な路である最短路を探索することとする。

2.2　分枝限定法

分枝限定法は、組織的な場合分けに基づく列挙法に不要な場合の排除機能を付す、組合せ最適化問題において汎用性の高い解法である。最適解を探索する過程においては、元の問題をいくつかの小規模な問題に分割し、その全てを解くことで等価的に元の問題を解くという構造をしている。この時、分枝操作と限定操作を使って最適解を見つけるグラフ探索を行う。分枝操作とは、元の問題を小問題に分割することにあたり、枝や節点を１つ決めてそれぞれの場合を個別に考察するものである。また、限定操作とは、今調べている部分問題に対して (Ⅰ) その最適値,(Ⅱ) その部分問題が実行不可能であること,(Ⅲ) その部分問題の最適解が元の問題の最適解とはならないこと, のいずれかに該当するかどうか判定し、それらのいずれかに該当すれば、これ以上は今調べている部分問題を分割して探索することを打ち止めにするというものである。

3　実験内容

3.1　課題１

節点集合 $V = \{0, 2, .., N-1\}$ と有向枝集合 $E \subseteq V \times V$ からなるグラフ $G(V, E)$ と各枝 $(u, v) \in E$ の長さ $d(u, v)$ が与えられている状況を考える。この時、点 s から点 t に至る最短路を求める分枝アルゴリズムを設計し、その擬似コードを書く。

3.2　課題２

課題１で設定したアルゴリズムを実装し、グラフの節点数及び枝数に対して実行時間及び探索したノード数を調べ、表やグラフにまとめる。

3.3　課題３

課題１で設定したアルゴリズムに関して限定操作を導入し、その妥当性を説明及び証明する。限定操作別の擬似コードを書き、最後に全体の限定操作アルゴリズムの擬似コードを書く。

3.4　課題４

限定操作を実装し、適当なインスタンスで課題２の実装と実行時間及び探索したノード数を比較する。また、結果を表やグラフにまとめる。

4　実験結果

4.1　課題１

関数 $SPath$ として Algorithm1 にグラフ G 上の点 s から点 t までの最短路を求める分枝アルゴリズムを記載する。設計したアルゴリズムの概要は、始点 s から終端点 t までの有向路が１つ求まったとき、その経路長を計算し、その経路長がそれまでの暫定の最短路の経路長よりも小さかった場合は最短路を更新する、というものである。この関数を $x \leftarrow s,\ F \leftarrow \{s\},\ tmp \leftarrow \infty,\ Path \leftarrow \{\}$ とした状態で実行すると始点から終点までの最短路が得られる。

Algorithm 1 分枝操作を含む関数 $SPath(x, F, tmp, Path)$

Require: グラフ上の節点 x,
始点 s から点 x に至る有向路に含まれる節点集合 F,
最短路の長さの暫定値 tmp,
暫定の最短路 $Path$

Ensure: 始点 s から終端点 t への最短路 $Path$

```
for (x, y) ∈ E, y ∉ F を満たす全ての有向枝について do
  if y = t then
    sum ← 0
    for 点 s から点 t へ至り,F を含む有向路の各枝 (u, v) do
      sum ← sum + d(u, v)
    end for
    if sum < tmp then
      tmp ← sum
      Path ← F ∪ {y}
    end if
  else
    SPath(y, F ∪ {y}, tmp, Path)
  end if
end for
return Path
```

4.2 課題２

課題１で設定したアルゴリズムを実装する。擬似コードにおいては関数の返り値として最短路を返すようにコードを書いていたが、本課題では最短路は必要なデータではないので $SPath$ 関数は void 型の関数として実装することにする。実装したプログラムのコードは付録に記載しておく。なお、今回は始点を 0, 終点を $N-1$ として問題を解いている。

プログラムを実行した結果、グラフの節点数及び枝数に対してかかった実行時間を図 1,2,3 に示す。また、グラフの節点数及び枝数に対して探索したノード数を図 4,5,6 に示す。このとき、実行時間や探索したノード数の軸は対数スケールとしている。なお、今回の実験では節点数を 6 から 12 までとし、各節点数 N に対して枝数 M を $2N \leq M \leq N(N-1)$ とし、それぞれの m に対して 1 つずつグラフを得て探索した。

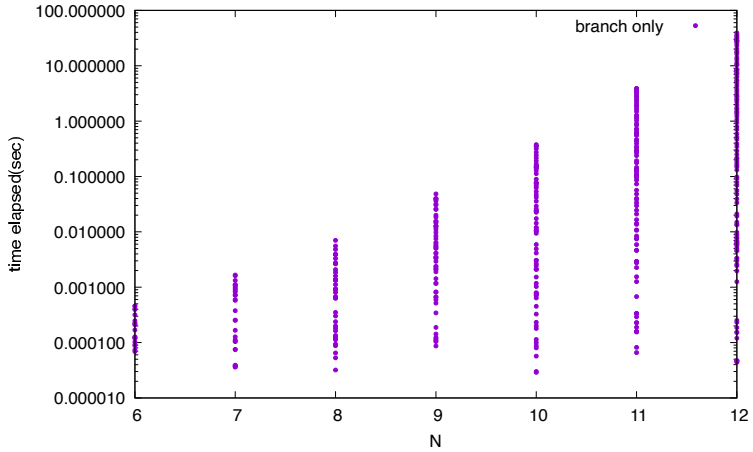

図 1: 節点数 N のグラフと実行時間 (秒) の関係

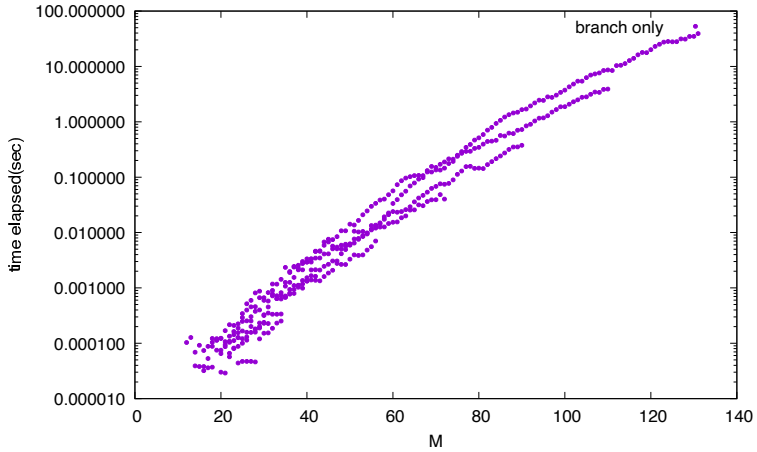

図 2: 枝数 M のグラフと実行時間 (秒) の関係

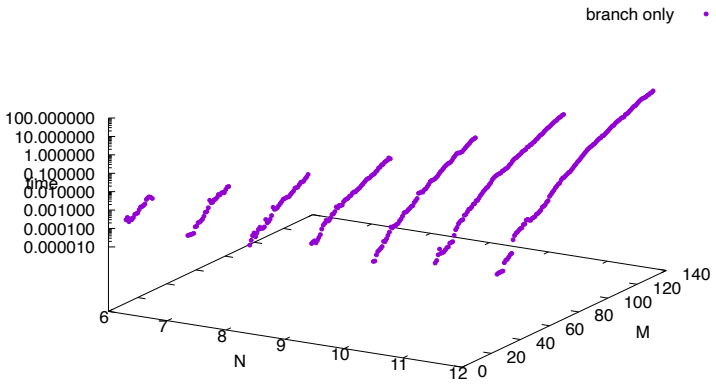

図 3: 節点数 N, 枝数 M のグラフと実行時間 (秒) の関係

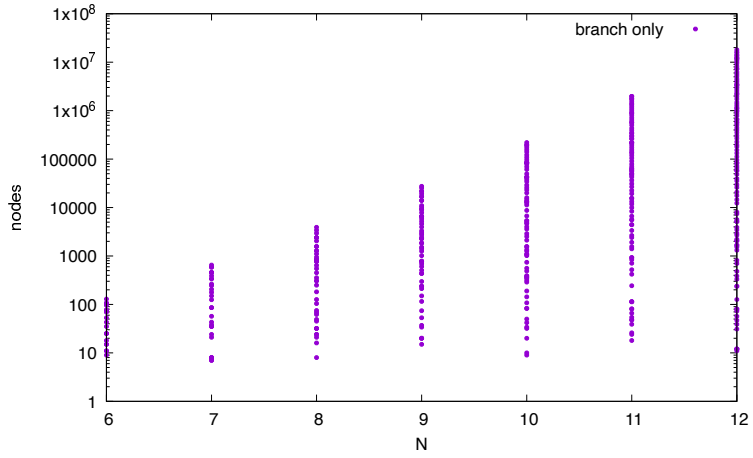

図 4: 節点数 N のグラフと探索したノード数の関係

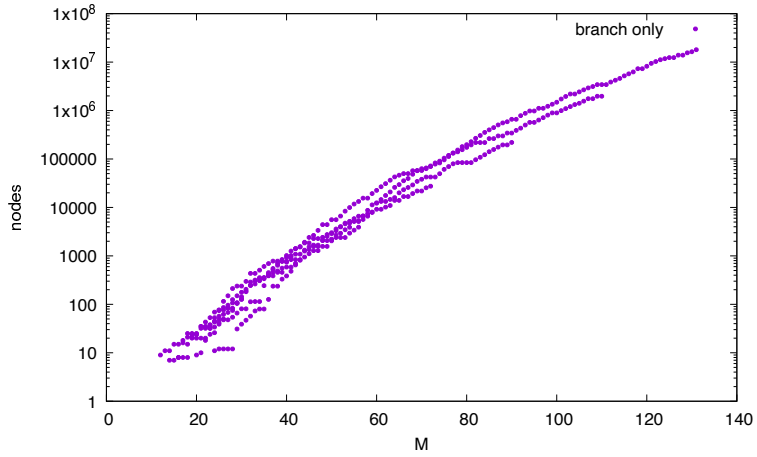

図 5: 枝数 M のグラフと探索したノード数の関係

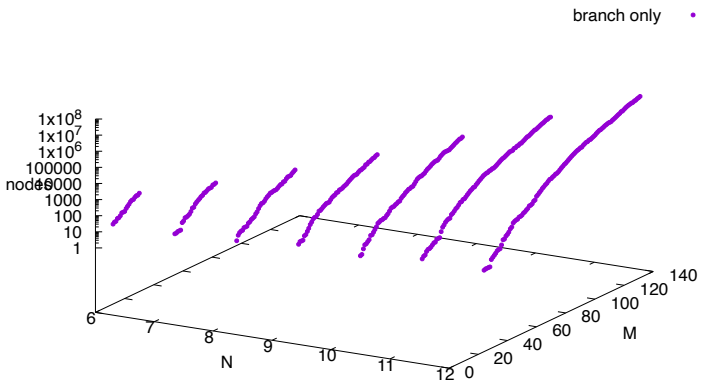

図 6: 節点数 N, 枝数 M と探索したノード数の関係

4.3 課題３

限定操作の一例を提案する。まずは Algrithm2 に従って、k 本の負の長さを持つ枝を、それぞれの枝の入る点が異なるという条件の下で選んだときに、k 本の枝の長さの総和として考えられる値について最小値を格納した配列 $MinEdges$ を用意する。ただし、k 本の枝は全て負の長さを持つものに限定しているので、場合によっては配列のサイズが $N-1$ 本未満となる可能性があることに留意する。

次に Algorithm3 により限定操作をする。ここで、この限定操作の妥当性について説明する。最短路は単純な路であることから最短路に含まれる枝の本数は $N-1$ 本以下である。また、グラフを探索するとき、始点 s からグラフ上の点 x まで l 本の枝で到達している場合は、もし x から終点 t 到達するならば最大でも $N-1-l$ の枝を伝って到達することになる。そして、$MinEdges[N-1-l]$ は $N-1-l$ 本以下の負の長さを持つ枝の組み合わせに対し、それらの枝が入る点が全て異なるという条件だけを課し、それらの枝が路を構成しているという条件を度外視した状態で、考えられる枝の長さの和における最小値であるので、連結した $N-1-l$ 本以下の負の長さを持つ枝の組み合わせを考えた場合における枝の長さの和の最小値以下の値をとっていることになる。これはすなわち、グラフに関して始点 s から終点 t までの路を一つ求めるなどして最短路の経路長の暫定値がある時に関して、始点 s から点 $x(\neq t)$ まで l 本の枝を伝って探索した状態で、それらの枝について長さの和を w とすれば、$w+MinEdges[N-1-l]$ が暫定値よりも大きい場合は、点 x から終点 t までいかなる経路を辿っても暫定値以下の経路長の路を構成できないことになるので、今調べている経路はこれ以上探索を続けても最短路が構成できないことを、この時点で判定できる。従って、この限定操作はある経路における探索の終端 (分枝木の枝刈り) に有効である。

以上のことに関して、全体の限定操作アルゴリズムを Algorithm4 に示す。この関数 $NPath$ を $x \leftarrow s,\ F \leftarrow \{s\},\ tmp \leftarrow \infty,\ Path \leftarrow \{\}$ として実行すると始点から終点までの最短路を Algorithm1 よりも少ない実行時間と探索ノード数で得られる。

4.4 課題４

課題３で述べた限定操作つきのアルゴリズムを実装する。ただし、実験データに最短路は必要ないので関数 $NPath$ を void 型で実装し、実装したプログラムのコードは付録に記載しておく。なお、今回は始点を 0, 終点を $N-1$ として問題を解いている。

そして、プログラムを実装した結果として、グラフの節点数及び枝数に対してかかった実行時間を図 7,8,9 に示す。また、グラフの節点数及び枝数に対して探索したノード数を図 10,11,12 に示す。このとき、実行時間や探索し

Algorithm 2 MinEdges

Require: グラフ G

Ensure: 負の長さを持つ枝の本数による累積和の配列 $MinEdge$

$MinDist$: 各点に入る枝のうち最小長さの枝の長さ格納
for $0 \leq i < N$ **do**
  $MinDist[i] \leftarrow \infty$
end for
for $0 \leq i < N,\ i \neq s$ **do**
  $MinDist[i] \leftarrow (min_{0 \leq j < N-1}(d(j, i))$
end for
$MinDist$ を昇べきにソート
$sum \leftarrow 0$
for $0 \leq i \leq N-1$ **do**
  $MinEdges[i] \leftarrow sum$
  if $MinDist[i] < 0$ **then**
    $sum \leftarrow sum + MinDist[i]$
  end if
end for
return $MinEdges$

Algorithm 3 限定操作

Require: 最短路の暫定値 tmp 始点 s からグラフ上の点 $x(\neq t)$ までの路の 1 つ P,
Algorithm2 で得た配列 $MinEdge$,

$w \leftarrow P$ の経路長
$l \leftarrow P$ に含まれる枝数
if $w + MinEdges[N-1-l] > tmp$ **then**
  点 x からの探索を打ち止める
end if

たノード数の軸は対数スケールとしている。ただし、探索の対象とするグラフは課題２で用いたものと同じものを用いている。

Algorithm 4 分枝限定操作を含む関数 $NPath(x, F, tmp, Path)$

Require: グラフ上の節点 x,
始点 s から点 x に至る有向路に含まれる節点集合 F,
最短路の長さの暫定値 tmp,
暫定の最短路 $Path$

Ensure: 始点 s から終端点 t への最短路 $Path$

前提として Algorithm2 の $MinEdges$ は求まっている
for $(x, y) \in E,\ y \notin F$ を満たす全ての有向枝について **do**
  if $y = t$ **then**
    $sum \leftarrow 0$
    for 点 s から点 t へ至り,F を含む有向路の各枝 (u, v) **do**
      $sum \leftarrow sum + d(u, v)$
    end for
    if $sum < tmp$ **then**
      $tmp \leftarrow sum$
      $Path \leftarrow F \cup \{y\}$
    end if
  else
    $l \leftarrow F$.length
    $w \leftarrow 0$
    for 点 s から点 y へ至り,$F \cup \{y\}$ を含む有向路の各枝 (u, v) **do**
      $w \leftarrow w + d(u, v)$
    end for
    if $w + MinEdges[N - 1 - l] > tmp$ **then**
      今調べている経路の探索を終端
    else
      $NPath(y, F \cup \{y\}, tmp, Path)$
    end if
  end if
end for
return Path

5 考察・議論

実験結果について考察する。課題１, ２に関しては図 3,6 で確認されるようにグラフの節点数や枝数が多いほど探索したノード数と実行時間が増えていた。また縦軸を対数スケールとしていることに留意しつつ、横軸がグラフの枝数である時に、直線的なプロットなっていることから、この分枝アルゴリ

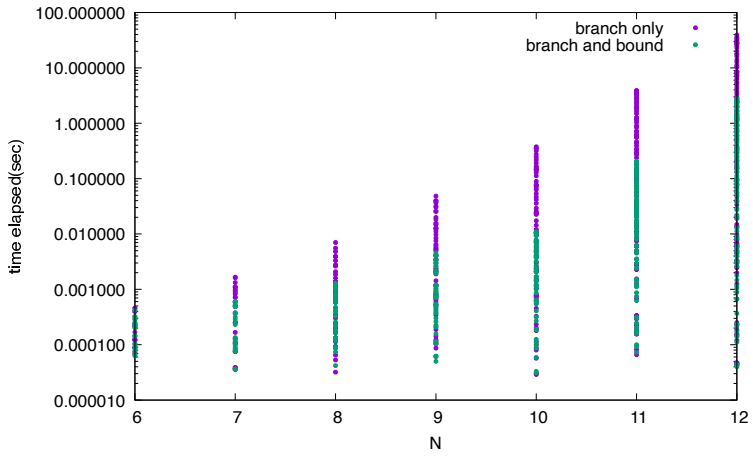

図 7: 紫:分枝操作のみで探索した場合における節点数 N のグラフと実行時間 (秒) の関係, 緑:限定操作を加えた場合における節点数 N のグラフと実行時間 (秒) の関係

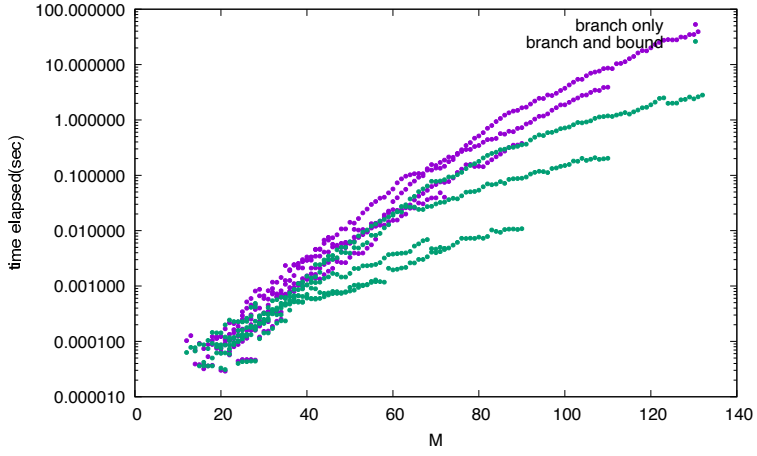

図 8: 紫:分枝操作のみで探索した場合における節点数 N 及び枝数 M のグラフと実行時間 (秒) の関係, 緑:限定操作を加えた場合における節点数 N 及び枝数 M のグラフと実行時間 (秒) の関係

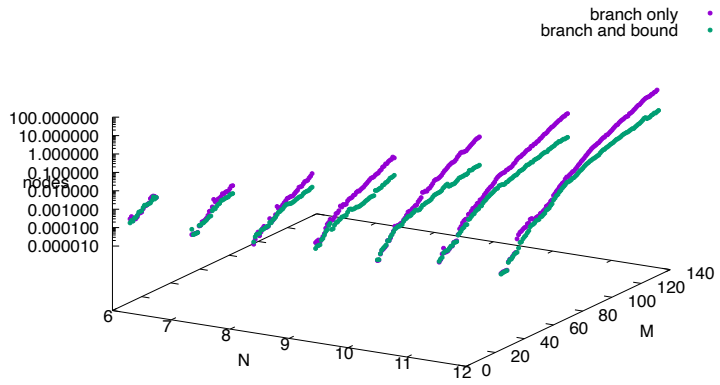

図 9: 紫:分枝操作のみで探索した場合おける節点数 N 及び枝数 M のグラフと実行時間 (秒) の関係, 緑:限定操作を加えた場合における節点数 N 及び枝数 M のグラフと実行時間 (秒) の関係

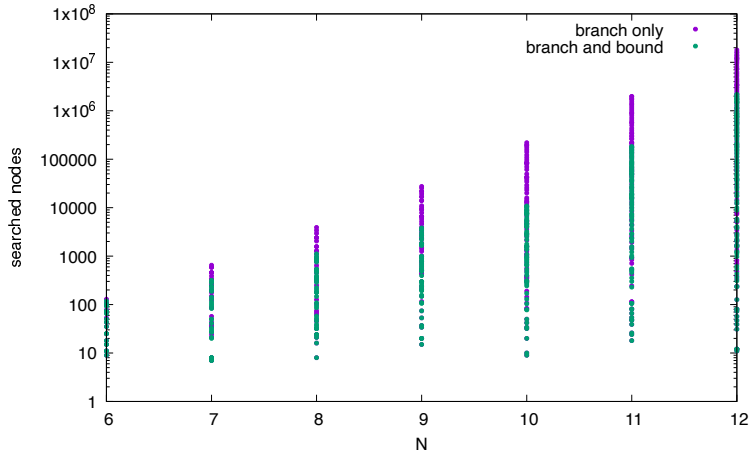

図 10: 紫:分枝操作のみで探索した場合における節点数 N のグラフと探索ノード数の関係, 緑:限定操作を加えた場合における節点数 N のグラフと探索ノード数の関係

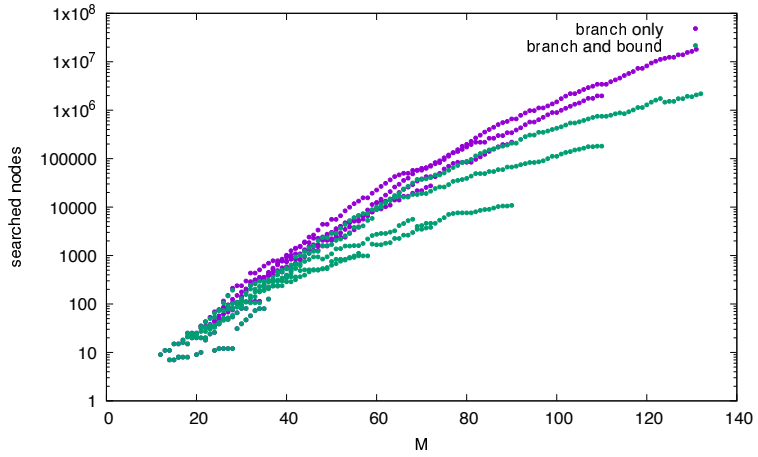

図 11: 紫:分枝操作のみで探索した場合における枝数 M のグラフと探索ノード数の関係, 緑:限定操作を加えた場合における枝数 M のグラフと探索ノード数の関係

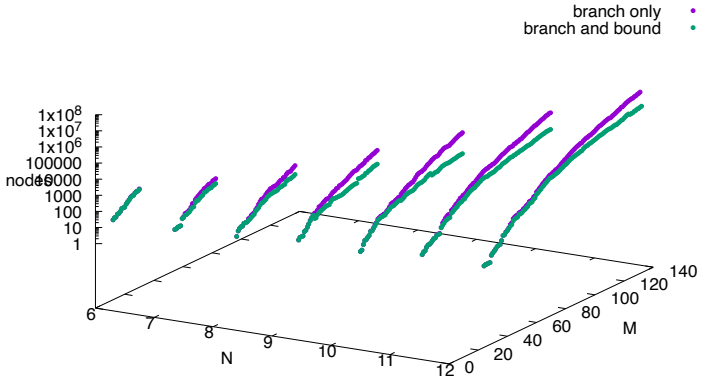

図 12: 紫:分枝操作のみで探索した場合における節点数 N 及び枝数 M のグラフと探索ノード数の関係, 緑:限定操作を加えた場合における節点数 N 及び枝数 M のグラフと探索ノード数の関係

ズムの計算量はグラフの枝数 M に対して $\mathcal{O}(2^M)$ 程度であると考えられる。

また、課題３, ４に関して図 9,12 から確かに探索ノード数や実行時間が、当該の限定操作により削減されたことが確認した。ただ、今回の限定操作において、$MinEdges$ に関してより適切な形に条件の緩和を行なった場合には、更なる探索ノード数の削減も期待できると考えられる。

6　まとめ

今回の実験を通して、NP クラスの問題に関しても問題のサイズが小さければ、分枝限定法で解決できることが確認できた。また、今回の最短路問題を解くにあたって分枝アルゴリズムのみではグラフの枝数が増えるにつれ、急激に実行時間が増加するため、限定操作をいかに問題の構造に適する形にするかが肝要であるということも見てとれた。組み合わせ最適化は多種多様に存在し、それぞれに対して適切な解法は変わってくることについては今後深く留意したい。

7　付録

7.1　課題２のソースコード

プログラムとグラフのデータをまとめたディレクトリはファイルの同じ階層に配置している。

```
#include<fstream>
#include<iostream>
#include<iomanip>
#include<string>
#include<algorithm>
#include<vector>
#include<utility>
#include<tuple>
#include<map>
#include<queue>
#include<stack>
#include<deque>
#include<bitset>
#include<math.h>
#include<numeric>
#include<time.h>
using namespace std;
```

```
#define infty 1e9

//探索済か判定する配列
vector<bool> unseen;
//Algorithm1 の関数を実装
void SPath(int x, vector<int> &F, vector<vector<int> >G,
                                int n, int &tmp, int64_t &cnt){
    //擬似コード１行目
    for(int y=0;y<n;y++){
        //グラフ上に枝 (x,y) が存在するか判定
        if(G[x][y]!=infty) {
            //y が探索済か判定
            if(unseen[y]){
                cnt++;
                //擬似コード２行目以降
                if(y==n-1){
                    int sum=0;
                    for(int i=0;i<F.size()-1;i++){
                        sum+=G[F[i]][F[i+1]];
                    }
                    sum+=G[F[F.size()-1]][n-1];
                    if(sum<tmp) tmp=sum;
                }else{
                    F.push_back(y);
                    //y を探索済にする
                    unseen[y]=false;
                    SPath(y,F,G,n,tmp,cnt);
                }
            }
        }
    }
    //探索可能点がないので、1 つ前の節点に戻る
    unseen[x]=true;
    F.pop_back();
}

int main(){
    //ファイルからの入力
    for(int n=6;n<=12;n++){
```

```
    for(int m=n*2;m<=n*(n-1);m++){
        string s = to_string(m);
        string t = to_string(n);
        ifstream fin("Topic_6_Graphs/n_"+t+"/n_"+t+"_m_"+s+".txt");
        int check_n,check_m;
        fin >> check_n >> check_m;

        vector<vector<int> > G(n,vector<int>(n,infty));
        for(int i=0;i<m;i++){
            int from,to,cost;
            fin >> from >> to >> cost;
            G[from][to]=cost;
        }

        //全ての節点を未探索にする
        unseen.assign(n,true);
        //順序付き節点集合
        vector<int> F;
        //F に始点を格納
        F.push_back(0);
        unseen[0]=false;
        //最短路の長さの暫定値を実質無限とする
        int tmp=infty;

        //cnt 変数で探索したノード数を数える
        int64_t cnt=0;
        double t_start=clock();
        SPath(0,F,G,n,tmp,cnt);
        double t_end=clock();

        cout << n << ' ' << m << ' ';
        cout << tmp << ' ';
        cout << fixed << setprecision(8) << (t_end-t_start)/CLOCKS_PER_SEC << ' ';
        cout << cnt << endl;
    }
    }
}
```

7.2 課題４のソースコード

課題２のソースコードに限定操作を加えたものであるため、一部のコメントは割愛している。

```
#include<fstream>
#include<iostream>
#include<iomanip>
#include<string>
#include<algorithm>
#include<vector>
#include<utility>
#include<tuple>
#include<map>
#include<queue>
#include<stack>
#include<deque>
#include<bitset>
#include<math.h>
#include<numeric>
#include<time.h>
using namespace std;

#define infty 1e9

vector<bool> unseen;
vector<int> MinEdges;
//Algorithm4 の関数を実装
void NPath(int x, vector<int> &F, vector<vector<int> >G,
                    int n, int &tmp, int &cnt){
    for(int y=0;y<n;y++){
        if(G[x][y]!=infty) {
            if(unseen[y]){
                cnt++;
                if(y==n-1){
                    int sum=0;
                    for(int i=0;i<F.size()-1;i++){
                        sum+=G[F[i]][F[i+1]];
                    }
                    sum+=G[F[F.size()-1]][n-1];
                    if(sum<tmp) {
```

```
                    tmp=sum;
                }
            }else{
                //限定操作 (Algorithm3 参照)
                int sum=0;
                for(int i=0;i<F.size()-1;i++){
                    sum+=G[F[i]][F[i+1]];
                }
                sum+=G[F[F.size()-1]][y];
                int edge=n-1-F.size();
                int extra=MinEdges[edge];
                //探索を終端するか判定
                if(sum+extra>tmp){
                    //打ち止め
                    continue;
                }else{
                    //探索継続
                    F.push_back(y);
                    unseen[y]=false;
                    NPath(y,F,G,n,tmp,cnt);
                }
            }
        }
    }
    unseen[x]=true;
    F.pop_back();
}

int main(){
    for(int n=6;n<=12;n++){
    for(int m=n*2;m<=n*(n-1);m++){
        string s = to_string(m);
        string t = to_string(n);
        ifstream fin("Topic_6_Graphs/n_"+t+"/n_"+t+"_m_"+s+".txt");
        int check_n,check_m;
        fin >> check_n >> check_m;

        vector<vector<int> > G(n,vector<int>(n,infty));
```

```
        for(int i=0;i<m;i++){
            int from,to,cost;
            fin >> from >> to >> cost;
            G[from][to]=cost;
        }

        //限定操作に必要な配列を用意 (Algorithm2 参照)
        vector<int> MinDist(n,infty);
        MinEdges.assign(n,0);
        for(int j=1;j<n;j++){
            for(int i=0;i<n-1;i++){
                MinDist[j]=min(MinDist[j],G[i][j]);
            }
        }
        sort(MinDist.begin(),MinDist.end());
        int sum=0;
        for(int i=0;i<n;i++){
            MinEdges[i]=sum;
            if(MinDist[i]<0) sum+=MinDist[i];
        }

        unseen.assign(n,true);
        vector<int> F;
        F.push_back(0);
        unseen[0]=false;
        int tmp=infty;

        int cnt=0;
        double t_start=clock();
        NPath(0,F,G,n,tmp,cnt);
        double t_end=clock();

        cout << n << ' ' << m << ' ';
        cout << tmp << ' ';
        cout << fixed << setprecision(8) << (t_end-t_start)/CLOCKS_PER_SEC << ' ';
        cout << cnt << endl;
    }
    }
}
```