土木学会論文報告集
第 262 号・1977 年 6 月

最適交通ネットワーク問題の厳密解法と近似解法

COMPUTATIONAL PROCEDURES FOR OPTIMAL TRANSPORTATION NETWORK PROBLEM

枝村俊郎*・森津秀夫**
By Toshiro EDAMURA and Hideo MORITSU

1. まえがき

ある地域に交通ネットワークを計画するとき，一般に使用される手法はいくつかの代替案を比較することである．しかし，この方法では代替案の作成はもっぱら計画者の主観によるので，そのため最適な計画が代替案の中に含まれないことさえ起こり得る．ネットワークが複雑になる場合には代替案の作成そのものが困難でさえある．そこでこれまでも客観的な基準によって最適なネットワーク計画案をつくるアルゴリズムを開発しようという試みがなされてきた．これがいわゆる最適交通ネットワーク問題である．

いま，ある平面上にノードの集合があり，かつこれらを結ぶリンクの集合があるとする．あらゆる可能なリンクのうち，検討対象として取り上げるリンクをこれまでの研究者の用語例にならい，建設可能リンクということにする．最適交通ネットワーク問題はこれらの建設可能リンクのなかから，制約条件を満たし目的関数を最大または最小にするリンクの組み合わせを探す問題をいうものとする．この問題は組み合わせ問題の一種である．

最適交通ネットワーク問題の解を求める厳密解法は体系的総数え上げ法の一種であり，過去に Scott[1], Ridley[2], Hoang[3] などの研究がある．厳密解法は解を系統的に並べた組み合わせトリーを探索するもので，分枝限定法と backtrack 法とに大別される．Hoang の backtrack 法は，これらのうち最新の研究であり，組み合わせトリー上の節点での目的関数の下限値を使用し，計算の効率化を行っているため，現時点では最良のアルゴリズムといえよう．われわれの研究はこれを基礎として議論を進める．

ところで厳密解法ではリンク，ノードがふえれば，目的関数の値を調べなければならない実行可能解の数が増加することと，最短路探索に要する計算量の増加のために計算時間が膨大になる．交通網計画などで実際に取り扱おうとするネットワークは，リンク数，ノード数が大きいために厳密解法を適用することはまず不可能と言ってよい．そこで得られる解は必ずしも最適解でなくとも実用性を考え計算時間が短くてすむ解法が研究されてきた．すなわち，いわゆるヒューリスティックな解法であって，通常近似解法と呼ばれているものである．近似解法として基本的なものは Scott の forward solution algorithm と backward solution algorithm[1] であり，わが国での最近の研究には西村[4]，森地[5]，飯田[6]などの研究がある．しかし，これらは必ずしも最適解がどのようなものかを十分に検討して作られたものではないように思われる．

そこで，この論文では，まず最適解を求める厳密解法の改良を行い，いくつかの改良アルゴリズムを提案する．そして，それらのアルゴリズムを用い，いくつかの例題を解くことによって，各アルゴリズムの特徴の比較・検討を行う．次に得られた計算結果をもとに，実用性を重視した新しい近似解法を提案し，かつその考察を行う．

2. 問題の定式化

最適交通ネットワーク問題では目的関数と制約条件の組み合わせによりさまざまな問題が構成されるが，ここでは基本的なアルゴリズムの比較・改良を目的とし，もっとも単純な問題のみを考えることにする．すなわち，リンクの長さの合計がある長さより短いネットワークの中で，ノード対の最短距離の合計が最小のものを選択する問題に限定して考察を行う．

さらに次の仮定を設けるが，これにより問題が一般性を失うことはない．

* 正会員　工博　神戸大学教授　工学部土木工学科
** 正会員　工修　神戸大学助手　工学部土木工学科

① リンクに向きはない．
② 同一の2ノードを結ぶ複数のリンクはない．
③ ループをなすリンクはない．
④ リンクの長さは正である．

これより，問題は以下のように定式化できる．

問題1

$$\min Z=\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} d_{ij}(x_1, x_2, \cdots, x_m) \quad \cdots\cdots(1)$$

$$\text{s.t.} \sum_{k=1}^{m} l_k x_k \leqq L^c \quad \cdots\cdots(2)$$

$$x_k=0 \text{ or } 1 \quad (k=1, 2, \cdots, m) \cdots\cdots(3)$$

ここに，Z はノード対の最短距離の合計，d_{ij} は ij ノード間の最短距離で，ij ノード間に経路がない場合は無限大とする．x_k は k リンクの状態を表わす 0-1 変数で，k リンクがネットワークに含まれないとき $x_k=0$，ネットワークに含まれるときは $x_k=1$ とする．また，l_k は k リンクの長さ，L^c はネットワークに含まれるリンクの長さの合計の上限，m はリンク数，n はノード数である．

式 (2) は解として得られるネットワークに含まれるリンクの長さの合計，つまりネットワーク長がある与えられた値以下でなければならないことを示している．以後 L^c のことを制約長と呼ぶことにする．ただし，制約長は次の条件を満足するものでなければならない．

$$L^{\min} \leqq L^c \leqq L^{\max}=\sum_{k=1}^{m} l_k \quad \cdots\cdots(4)$$

ここに，$L^{\max}$ は $x_k=1 (k=1, 2, \cdots, m)$ すなわちすべての建設可能リンクが含まれるネットワークのネットワーク長で，このネットワークを以後最大ネットワークと呼ぶことにする．$L^{\min}$ は最大ネットワークに対する経済木 (minimal spanning tree) の長さである．

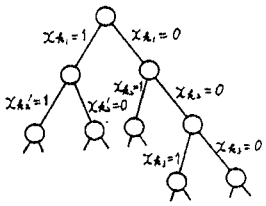
3. 厳密解法

Hoang が導入した目的関数の下限値を求める方法を受け継ぎ，組み合わせトリーを探索するときの分枝方法を改良したアルゴリズムを開発した．ここではそれらのアルゴリズムについて説明し，例題の計算を行ってそれらの特徴を調べ，優劣の比較検討を行う．

(1) アルゴリズム

比較検討を行ったアルゴリズムは，分枝方法を改良して新たにつくった4種に Hoang の branch search algorithm を加えた計5種類である．これらのアルゴリズムすべてに共通の事柄についてここで述べておく．

図—1 に示すような組み合わせトリーの各節点での目的関数の下限値は Hoang の方法によって求める．すなわち，次のナップサック問題を導入する．なお，ここで言う節点の目的関数の下限値とは，組み合わせトリーでその節点から分枝して到達する実行可能解の目的関数値の下限のことである．



図—1 組み合わせトリー

問題2

$$\min F=\sum_{k=1}^{m} f_k y_k \quad \cdots\cdots(5)$$

$$\text{s.t.} \sum_{k=1}^{m} l_k y_k \geqq L^{\max}-L^c \quad \cdots\cdots(6)$$

$$y_k=0 \text{ or } 1 \quad (k=1, 2, \cdots, m) \cdots\cdots(7)$$

ここに，f_k は k リンクをネットワークから除くことによって生じる問題1の目的関数の増加の下限値である．y_k は k リンクの状態を表わす 0-1 変数で，k リンクがネットワークに含まれないとき $y_k=1$，ネットワークに含まれるとき $y_k=0$ とする．そして，それまでに $x_k=0$ あるいは $x_k=1$ と固定されているリンクに対しては，それぞれ $y_k=1$，$y_k=0$ という制約式を追加するものとする．この問題の解を得るには，かなりの計算が必要なので，式 (7) を

$$0 \leqq y_k \leqq 1 \quad (k=1, 2, \cdots, m) \cdots\cdots(8)$$

と置き換えた線形計画問題を考える．いま，式 (5)，(6)，(8) で表わされる問題を問題3としよう．その最適解の目的関数値を F' とすると，問題1の目的関数の下限値 Z^l は，

$$Z^l=Z^0+F' \quad \cdots\cdots(9)$$

によって決めることができる．ただし，Z^0 は最大ネットワークの目的関数値である．しかし，式 (9) で下限値を定めると，組み合わせトリーの末端へいくにしたがって下限値とその節点から分枝して到達する実行可能解の目的関数の最小値との差が大きくなることがよくある．そこで組み合わせトリーの任意に決めた節点で，それまでにネットワークから除くことに決められたリンク以外からなるネットワークの目的関数値を求める．それ以下の節点ではこの目的関数値を式 (9) の Z^0 の代わりに使い下限値を強める．Hoang はさらに f_k の見直しも行っているが，必ずしも計算時間が短くなるとは限らないと思われるので，ここでは行わないことにする．

各アルゴリズムの計算を始めるときの目的関数の上限値を決めるための初期解には backward 法の結果を使用する．ただし，ここで言う backward 法とは単に最大ネットワークから順次リンクを1本ずつ除いてゆくだけの方法である．すなわち，Scott がその backward solution algorithm 中，netran と称するサブルーチンで行っているようなネットワーク変換は行わない．

a) アルゴリズム 1, 2, 3

Hoang の branch search algorithm そのものをアル

ゴリズム1と称することとし，ここでは説明を省く．

先の問題3の最適解は，一般に1変数のみが小数値で他は0か1のどちらかである．そこでこの小数値の変数の値を1にした解を考えると，それは問題2の実行可能解である．そしてまた，問題1の実行可能解で，しかも最適解に近い解であることが多い．すなわち，問題3の解を使って，

$$x_k=0 \quad \text{if} \quad y_k>0 \quad (k=1, 2, \cdots, m) \cdots\cdots\cdots\cdots (10)$$

とすれば，組み合わせトリー上で $x_k=1$ に固定されているリンクとまだ値を固定されていない自由変数のリンク（これを $x_k=-1$ で表わすことにする）からなるネットワークは問題1の実行可能解になる．この実行可能解は，問題3を解いた節点から分枝して到達する実行可能解の中で，目的関数の下限値がきわめて小さいものである．したがって問題3を使用して式（10）の分枝方法を使えば，早く最適解を得ることが予想される．

式（10）を使った方法では最大ネットワークから出発しネットワークから除くリンクを先に決め，自由変数のリンクはネットワークに含まれるものと見なしている．これとは逆にすべてのノードがばらばらの状態から出発し，ネットワークに含めるリンクを先に決め，自由変数のリンクはネットワークに含まれないと見なす方法も考えられる．問題3をこのまま使用するなら，この場合は式（10）のかわりに次の式（11）を使う．

$$x_k=1 \quad \text{if} \quad y_k=0 \quad (k=1, 2, \cdots, m) \cdots\cdots\cdots\cdots (11)$$

ただし，単に式（11）にしたがって分枝しただけでは最適解になる可能性のある実行可能解が調べられないことが起きる．そのため，分枝に際して次式を満たすようにする．

$$l_s+\sum_{k=1}^{m}\delta_k l_k>L^c \quad \text{for} \quad \text{all} \quad s: x_s=-1 \cdots\cdots (12)$$

$$\delta_k=\begin{cases}1 & \text{if } x_k=1\\ 0 & \text{if } x_k=0 \text{ or } -1\end{cases}$$

これは自由変数のどのリンクをネットワークに入れても，ネットワーク長が制約長を越えることを意味している．

式（10）の分枝方法を使うものをアルゴリズム2，式（11），（12）の分枝方法を使うものをアルゴリズム3と称する．アルゴリズム2,3の計算手順は次のようになる．

① 近似解法によって初期解を求め，その目的関数値を Z^u，ネットワーク長を L^0，解を $X^0=\{x_1^0, x_2^0, \cdots, x_m^0\}$ とする．② へ進む．

② その節点までに $x_k=0$ または $x_k=1$ と決められている変数，すなわち固定変数を制約に加えて問題3を解き，目的関数の下限値 Z^l を求める．もし $Z^l>Z^u$ なら ⑤ へ，他は ③ へ進む．

③ 問題3の解を使い，アルゴリズム2では式（10），アルゴリズム3では式（11），（12）によって分枝する．④へ進む．

④ その節点の解に対する目的関数値 Z，ネットワーク長 L を求める．(Z, L) が辞書編集的に (Z^u, L^0) より小さければ，(Z^u, L^0) を (Z, L)，X^0 を $X=\{x_1, x_2, \cdots, x_m\}$ で置き換える．⑤ へ進む．

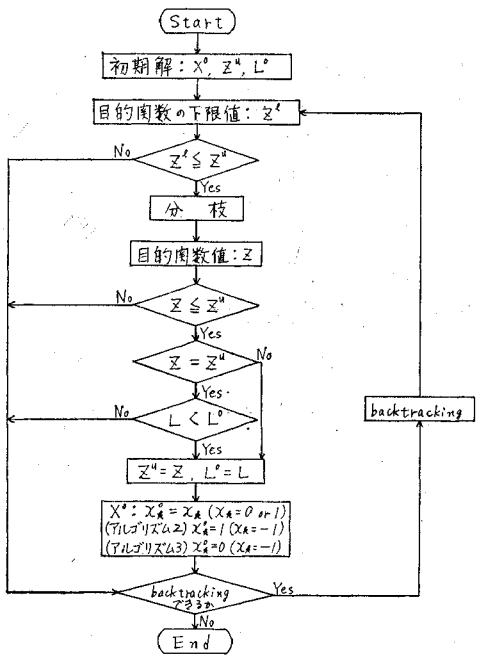
⑤ （アルゴリズム2）その節点へくるときの分枝変数の値が0ならそれを1にした節点へ進む．もし分枝変数の値が1ならそれを -1 にし，さらに手前の節点へもどり ⑤ を繰り返す（backtracking の操作）．新たな節点が決まれば ② へ進む．もどる節点がない場合は計算終了で，そのときの X^0 が最適解である．

（アルゴリズム3）その節点へくるときの分枝変数の値が1ならそれを0にした節点へ進む．もし分枝変数の値が0ならそれを -1 にし，さらに手前の節点へもどり ⑤ を繰り返す．新たな節点が決まれば ② へ，もどる節点がなければ計算終了で，そのときの X^0 が最適解である．

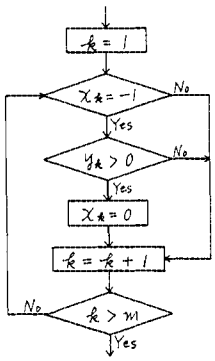
アルゴリズム 2,3 のフローチャートは，図—2 のようになる．また，分枝部分の詳細を示したものが 図—3,4 である．

b） アルゴリズム 4,5

最短距離 d_{ij} は，ij ノード間に経路がない場合無限大とした．したがって経路のないノード対が1組でもあれば，そのネットワークの目的関数値は無限大になる．すなわち，ある解の目的関数値が有限値であるための必要十分条件は，そのネットワークが連結であることであ



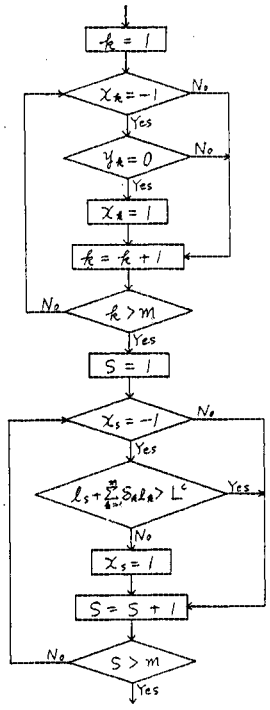
図—2　アルゴリズム 2,3 のフローチャート



図—3 アルゴリズム 2 の分枝のフローチャート

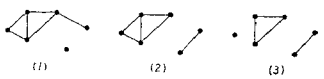
る．

Scott[1], Hoang[3] のアルゴリズムは連結網かどうかを調べる過程は含まれていなかった．ここでのアルゴリズム 1,2,3 においても，その計算過程で生じる実行可能解の中にはネットワークが非連結になるものが存在する．



図—4 アルゴリズム 3 の分枝のフローチャート

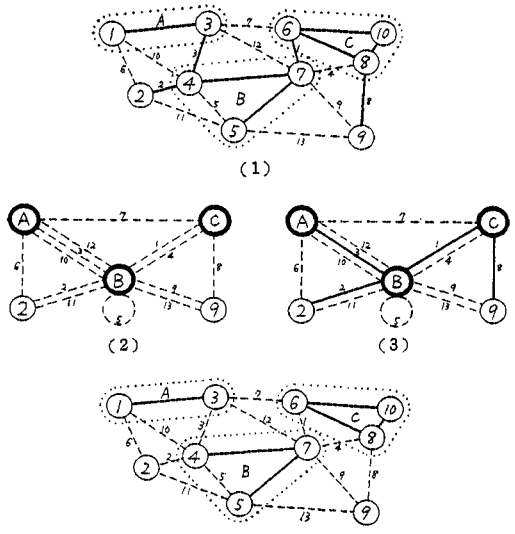
非連結網には 図—5(1) のように孤立したノードがあるもの，図—5(2) のように複数の連結成分からなるもの，図—5(3) のようにその両方の場合のものがある．孤立したノードがあるかどうかは計算過程で容易に調べることができる．アルゴリズム 1,2,3 でも孤立したノードを含むネットワークについては目的関数の計算を省略している．しかし，図—5(2) のようにネットワークが複数の連結成分で構成されているかどうかを調べるためには，それだけを目的とする計算が別に必要となる．



図—5 非連結網の例

制約長が短く，実行可能解のネットワークに含まれるリンクが少ないときには，特にそのネットワークが非連結である可能性が高いであろう．そこでネットワークを連結に保ったまま実行可能解を探索すれば計算の効率がよくなることが予想できる．ネットワークを連結に保つには次のようにすればよい．

組み合わせトリーに従って解を探索しているとき，リンクの状態は 3 種類ある．すなわち，変数で言えば $x_k=-1, 0, 1$ である．たとえば，ある計算段階でのネットワークが 図—6(1) のようになったとする．図の実線は $x_k=1$，破線は $x_k=-1$ のリンクで $x_k=0$ のリンクは省略してある．



図—6 連結網構成の例

1 と 3,4 と 5 と 7,6 と 8 と 10 のノードはそれぞれ $x_k=1$ のリンクで結ばれている．これらのノードは組み合わせトリー上の現在の節点で backtracking を行わない限り，互いに結ばれたままである．そこで，これらの連結成分をひとつのノードと考え，ノード A, B, C とする．そうすれば，図—6(1) は 図—(6)(2)のように書き換えることができる．なお，図—6 (1),(2) のリンクのそばの数字は自由変数のリンクを長さの短い順に並べたときのリンク番号である．

図—6(2) のネットワークが生成木 (spanning tree) を含むようにすれば，元のネットワークが連結になる．制約長を満たすためにはできるだけ生成木の長さが短い方がよいので，このネットワークの経済木をなすリンクをネットワークに含めることにする．図—6 (2) の経済木は 図—6(3) の実線のリンクで構成される．図—6(3) のネットワークを元のネットワークにもどせば 図—6 (4) になる．すなわち，図—6(4) のリンク 1,2,3,8 の変数の値を 1 にすれば，ネットワークは $x_k=1$ のリンクだけで連結になる．組み合わせトリー上でこの節点から分枝して到達する節点の表わすネットワークもすべて連結になる．

以上をまとめれば，固定変数のリンクだけではネットワークが連結でないような組み合わせトリーの節点で，連結網になるように分枝する計算過程は次のようになる．まず，$x_k=1$ のリンクで結ばれた連結成分をひとつのノードと考えたネットワークで，$x_k=-1$ のリンクからなる経済木を探す．そして経済木を構成するリンクの変数を 1 にするように分枝する．

これから明らかなように，連結網を構成する計算を始

めた節点から分枝した節点の表わす解は，その解のネットワークが連結ならば，ネットワーク長は上に述べた方法で連結網になるように分枝を終ったときのネットワークのネットワーク長より短くなることはない．したがって上に述べた方法で連結網にしたときのネットワークが実行可能解でなければ元の節点から分枝する必要はなく，backtracking を行えばよい．

実際のアルゴリズムではこの計算にはラベリングを用い，ある連結成分に属するノードには同一のラベルを与え，このラベルを1種類にしてゆけばよい．

アルゴリズム 4, 5 は，$x_k=1$ のリンクからなるネットワークが非連結になればネットワークを連結にする計算過程を実行し，連結の場合にはそれぞれアルゴリズム2とアルゴリズム3にしたがって解の探索を行うものである．

アルゴリズム4による計算手順は次のようになる．

① 近似解法によって初期解を求め，その目的関数値を Z^u，ネットワーク長を L^0，解を $X^0=\{x_1^0, x_2^0, \cdots, x_m^0\}$ とする．②へ進む．

② ネットワークを連結にできるかどうか調べる．自由変数のリンクでネットワークを連結にできないとき，あるいは連結にしたネットワークの長さが制約長を越えるときは⑦へ，他は③へ進む．

③ ② の結果に基づき，連結網をなすように分枝する．④ へ，進む．

④ その節点での固定変数を制約に加えて問題3を解き，目的関数の下限値 Z^l を求める．もし $Z^l>Z^u$ なら ⑦ へ，他は ⑤ へ進む．

⑤ 問題3の解を使い分枝する．⑥ へ進む．

⑥ その節点の解に対する目的関数値 Z，ネットワーク長 L を求める．(Z, L) が辞書編集的に (Z^u, L^0) より小さければ，(Z^u, L^0) を (Z, L)，X^0 を $X=\{x_1, x_2, \cdots, x_m\}$ で置き換える．⑦ へ進む．

⑦ その節点へくるときの分枝が③によるものなら⑧へ，⑤によるものなら⑨へ進む．

⑧ その節点へくるときの分枝変数の値が1ならそれを0にした節点へ進む．もし分枝変数の値が0ならそれを -1 にし，さらに手前の節点へもどり ⑧ を繰り返す．新たな節点が決まれば ② へ進む．もどる節点がない場合は計算終了で，そのときの X^0 が最適解である．

⑨ その節点へくるときの分枝変数の値が0ならそれを1に変えて ④ へ進む．分枝変数の値が1ならそれを -1 にし，さらに手前の節点へもどり ⑦ へ進む．

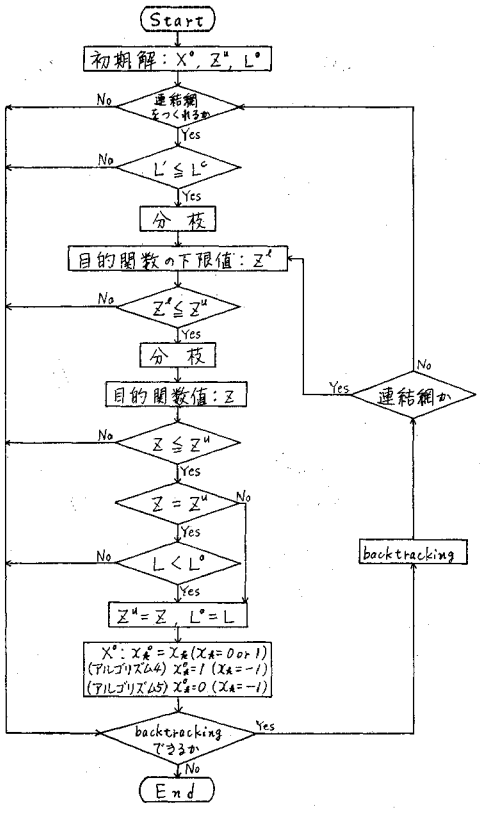
このようにアルゴリズム4では連結網になるまでは組み合わせトリーの節点から分枝した節点のうち，分枝変数を1にした場合を先に調べ，連結網になってからは分枝変数を0にした場合を先に調べることになる．アルゴリズム 1,2,3，それに次のアルゴリズム5では，分枝変数を1にした場合を先に調べるか0にした場合を先に調べるかどちらか一方で，アルゴリズム4のように途中で変わることはない．

アルゴリズム5による計算手順は ① から ⑥ まではアルゴリズム4と同じである．ただし，⑤の分枝はアルゴリズム4がアルゴリズム2と同じなのに対し，アルゴリズム5はアルゴリズム3と同じである．⑦以下は次のようになる．

⑦ その節点へくるときの分枝変数の値が1ならそれを0にした節点へ進む．もし分枝変数の値が0ならそれを -1 にし，さらに手前の節点へもどり ⑦ を繰り返す．新たな節点が決まれば ⑧ へ進む．もどる節点がない場合は計算終了で，そのときの X^0 が最適解である．

⑧ ③ での分枝変数の値を変えた場合は ② へ，そうでなければ ④ へ進む．

アルゴリズム4とアルゴリズム5はこのように分枝方法が異なるが，両者をまとめたフローチャートは 図—7 のようになる．なお，図—7 の L' はネットワークを連



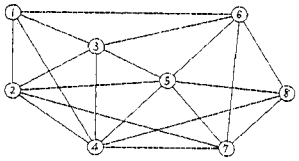
図—7　アルゴリズム 4,5 のフローチャート

結にしたときのネットワーク長である．

（2） 計算例とアルゴリズムの比較

アルゴリズム 1～5 により 3 つの例題を解いたが，ここではその結果を示しアルゴリズムの特徴を比較する．なお，計算には京都大学大型計算機センターのシステム I（FACOM 230-75）を使用した．

a） 例 題 1



図—8 例題 1 の最大ネットワーク

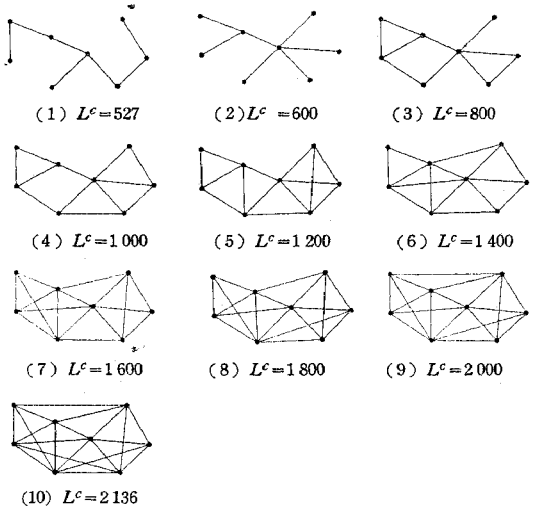
図—8 の最大ネットワークを例題 1 とする．リンク長は **表—1** のとおりで，ノード数，リンク数などは，$n=8$, $m=20$, $L^{max}=2136$, $L^{min}=527$ である．そこで，527～2136 の間に任意に 10 個の制約長を設け，それぞれの最適解を求める．

得られた最適ネットワークは **図—9** に示すとおりである．そして，各制約長の最適解の目的関数値をグラフで表わせば **図—10** のようになる．

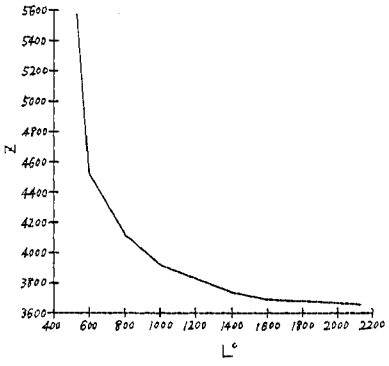
各アルゴリズムの計算時間（CPU タイム）は **表—2** のとおりで，グラフにしたものが **図—11** である．また，

表—1 例題 1 のリンク長

ノード / ノード	1	2	3	4	5	6	7	8
1	—	70	76	139	—	190	—	—
2	70	—	80	86	130	—	187	—
3	76	80	—	90	67	124	—	—
4	139	86	90	—	85	—	110	168
5	—	130	67	85	—	85	78	100
6	190	—	124	—	85	—	120	80
7	—	187	—	110	78	120	—	71
8	—	—	—	168	100	80	71	—



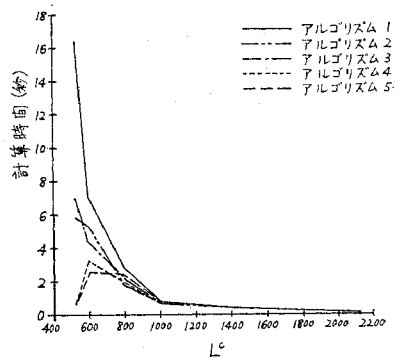
図—9 例題 1 の最適ネットワーク



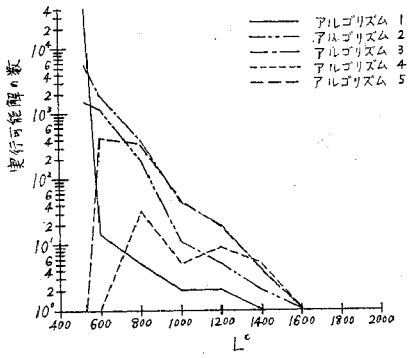
図—10 例題 1 の最適解の目的関数値

表—2 例題 1 の計算時間 （単位：秒）

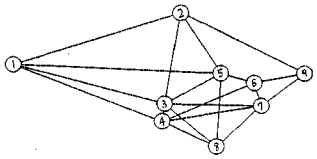
アルゴリズム / L^c	1	2	3	4	5
527	16.3	5.8	6.9	0.6	0.6
600	7.0	5.2	4.4	3.2	2.6
800	2.8	1.8	2.1	1.9	2.4
1 000	0.8	0.7	0.7	0.7	0.8
1 200	0.6	0.6	0.6	0.6	0.6
1 400	0.4	0.4	0.4	0.4	0.4
1 600	0.3	0.3	0.3	0.3	0.3
1 800	0.2	0.2	0.3	0.2	0.3
2 000	0.2	0.2	0.2	0.2	0.2
2 136	0.1	0.1	0.1	0.1	0.1
計	28.7	15.3	16.0	8.2	8.3



図—11 例題 1 の計算時間



図—12 例題 1 の計算過程で生じた実行可能解の数



図—13　例題2の最大ネットワーク

計算過程で目的関数値を調べた実行可能解の数は 図—12 のようであった．

b）例題 2

図—13 の最大ネットワークを例題2とする．リンク長は 表—3 のとおりで，$n=9$, $m=20$, $L^{max}=5610$, $L^{min}=1490$ である．1490～5610 の間で任意に与えた 10 個の制約長に対する最適解を求める．

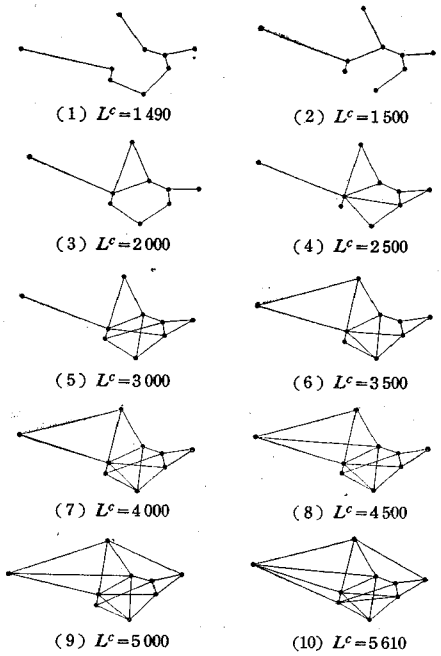
得られた最適ネットワークは 図—14，最適解の目的関数値は 図—15，計算時間は 表—4，図—16，計算過程で目的関数値を調べた実行可能解の数は 図—17 である．

c）例題 3

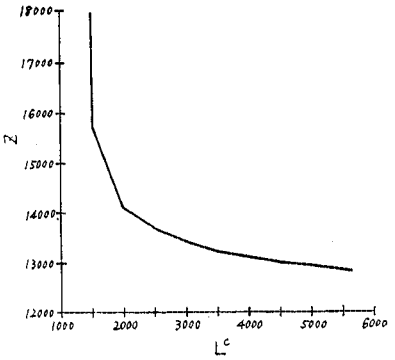
図—18 の最大ネットワークを例題3とする．リンク長は 表—5 のとおりで，$n=9$, $m=20$, $L^{max}=7080$, $L^{min}=1925$ である．1925～7080 の間で任意に与えた

表—3　例題2のリンク長

ノード＼ノード	1	2	3	4	5	6	7	8	9
1	—	540	480	490	640	—	—	—	—
2	540	—	290	—	230	—	—	—	430
3	480	290	—	60	200	—	290	210	—
4	490	—	60	—	—	310	310	190	—
5	640	230	200	—	—	105	—	240	—
6	—	—	—	310	105	—	75	—	160
7	—	—	290	310	—	75	—	190	170
8	—	—	210	190	240	—	190	—	—
9	—	430	—	—	—	160	170	—	—



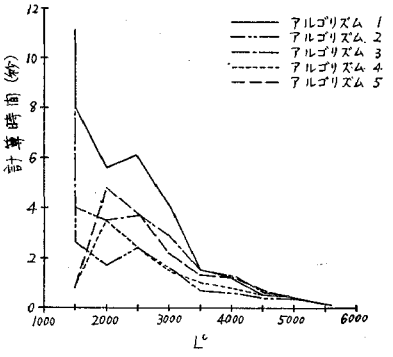
図—14　例題2の最適ネットワーク



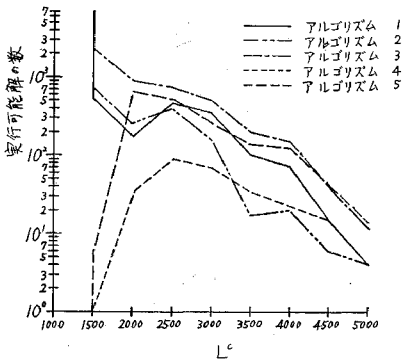
図—15　例題2の最適解の目的関数値

表—4　例題2の計算時間　（単位：秒）

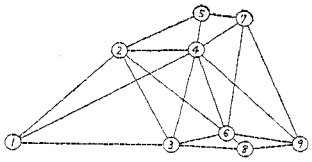
アルゴリズム＼L^c	1	2	3	4	5
1490	11.1	4.2	9.3	0.7	0.7
1500	8.0	2.6	4.0	0.8	0.7
2000	5.6	1.7	3.5	3.5	4.8
2500	6.1	2.4	3.7	2.4	3.8
3000	4.2	1.6	2.9	1.5	2.2
3500	1.5	0.7	1.5	1.0	1.3
4000	1.2	0.6	1.3	0.8	1.2
4500	0.5	0.4	0.6	0.5	0.7
5000	0.4	0.4	0.4	0.4	0.4
5610	0.1	0.1	0.1	0.1	0.1
計	38.7	14.7	27.3	11.7	15.9



図—16　例題2の計算時間



図—17　例題2の計算過程で生じた実行可能解の数



図—18 例題3の最大ネットワーク

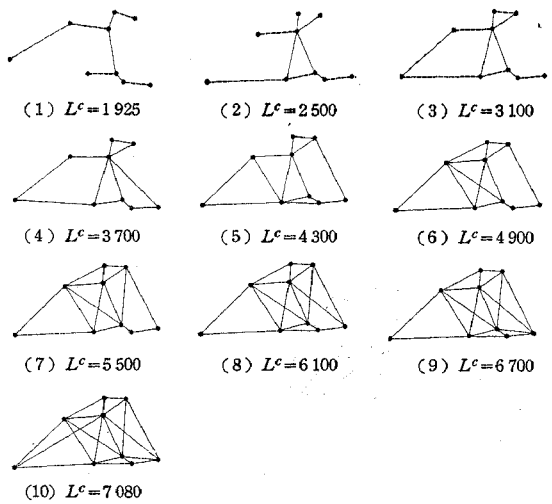
10個の制約長に対する最適解を求める．

得られた最適ネットワークは図—19，最適解の目的関数値は図—20，計算時間は表—6，図—21，計算過程で目的関数値を調べた実行可能解の数は図—22である．

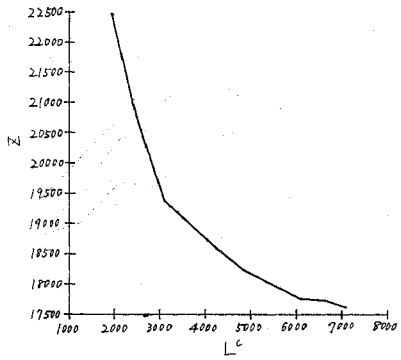
d） 解の性質についての考察

表—5 例題3のリンク長

ノード＼ノード	1	2	3	4	5	6	7	8	9
1	—	524	576	754	—	—	—	—	—
2	524	—	407	276	329	499	—	—	—
3	576	407	—	375	—	205	—	266	—
4	754	276	375	—	140	337	210	—	517
5	—	329	—	140	—	—	156	—	—
6	—	499	205	337	—	—	440	89	267
7	—	—	—	210	156	440	—	—	515
8	—	—	266	—	—	89	—	—	198
9	—	—	—	517	—	267	515	198	—



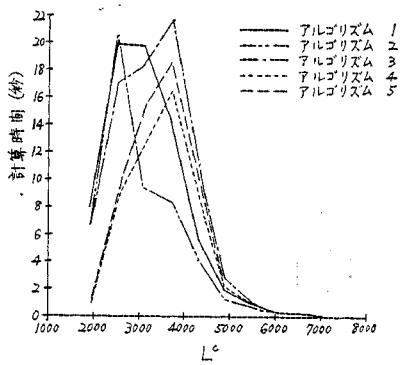
図—19 例題3の最適ネットワーク



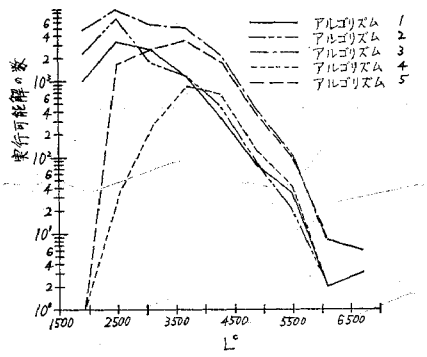
図—20 例題3の最適解の目的関数値

表—6 例題3の計算時間 （単位：秒）

アルゴリズム＼L^c	1	2	3	4	5
1925	8.1	6.8	6.7	0.7	0.7
2500	19.9	20.6	17.0	8.5	8.9
3100	19.8	9.4	18.4	12.5	15.4
3700	14.1	8.4	21.7	16.7	18.7
4300	5.7	4.3	11.4	9.1	10.3
4900	1.9	1.3	2.9	2.2	2.8
5500	0.9	0.7	1.0	0.9	1.0
6100	0.3	0.3	0.3	0.3	0.3
6700	0.3	0.3	0.3	0.3	0.3
7080	0.1	0.1	0.1	0.1	0.1
計	71.1	52.2	79.8	51.3	58.5



図—21 例題3の計算時間



図—22 例題3の計算過程で生じた実行可能解の数

例題の結果から最適解について次のことがわかる．

① 最適解の目的関数値と制約長の関係を便宜上，連続関数として表わせば，ほぼ双曲線状になっている（図—10，15，20）．

② ある最適ネットワークからリンクを除くことにより，それよりも制約長が小さい問題の最適ネットワークが得られることが多い（図—9，14，19）．

③ 制約長が大きければ最適ネットワークでリンクがノード以外のところで交差しているが，リンクがノード以外で交差する制約長の範囲では制約長が大きくなっても最適解の目的関数値はゆるやかに減少す

るだけである．一方，リンクがノード以外で交差しない制約長の範囲では，制約長が大きくなれば最適解の目的関数値は急激に減少している．

② については後でさらに検討する．③ での最適ネットワークでリンクがノード以外のところで交差していない制約長の範囲とは，例題1で $L^c<1200$，例題2で $L^c<3000$，例題3で $L^c<4900$ である．この $L^c=1200$，$L^c=3000$，$L^c=4900$ の点をそれぞれ図—10，15，20で見れば③のことがわかる．

e) アルゴリズムの比較

例題 1～3 のすべての制約長の問題に対して，他のいずれのアルゴリズムよりもつねに計算時間が短いというものはなかった．合計の計算時間で見ると，例題1ではアルゴリズム4とアルゴリズム5がよく，例題2,3ではアルゴリズム 4，アルゴリズム 2，アルゴリズム5の順によかった．

5種類のアルゴリズムはアルゴリズム1が Hoang の branch search algorithm で，その分枝方法を改良したものがアルゴリズム 2,3，さらにネットワークをつねに連結に保つようにしたものがアルゴリズム 4,5 であった．すなわち，5種類のアルゴリズムを改良の程度により次の3グループに分けることができる．

① アルゴリズム1
② アルゴリズム 2,3
③ アルゴリズム 4,5

グループ①，②，③を比較すると，計算時間と制約長との関係は①と②がほぼ同じで③のみが他と異なっている．①と②では例題1,2において制約長の値が小さくなれば計算時間が増加し，特に制約長が経済木の長さに近いところでは計算時間が急激に増加している．これに対して③では制約長が経済木の長さに近くなれば逆に計算時間が短くなり，例題3においても計算時間がもっとも長くなる制約長の値は①,②より大きい．

①，②と③の相違は，制約長が小さければ①，②では目的関数値を調べる実行可能解の中にネットワークが非連結になるものが含まれ得るが，③では非連結になるものが生じないことに起因するものであろう．制約長が経済木の長さに近づけば，計算過程で目的関数値を調べた実行可能解の数が③では少なくなっているのに対し，①,②では非常に多くなっていることもこれを裏付けている．このことはアルゴリズム 4,5 をつくるときに予期したとおりであり，制約長が小さいときの計算時間を短縮するという目的は十分に達せられた．

改良したアルゴリズムでは，組み合わせトリーの同じ節点から分枝した節点のうち分枝変数を0にした場合を先に調べ，backtracking を行ってから分枝変数を1にした場合を調べるものと，分枝変数を1にした場合を先に調べるものとの2種類があった．これによって5種類のアルゴリズムは次の2グループに分けることもできる．

ⓐ アルゴリズム 1,2,4
ⓑ アルゴリズム 3,5

アルゴリズム4はネットワークが連結になるまでは分枝変数を1にした場合を先に調べ，連結になってからは分枝変数を0にした場合を先に調べるのでⓐ,ⓑの中間とも考えられるが，計算結果を見ればⓐに含めるのが適当である．

計算結果について先の②，③のグループ内でⓐに属するものとⓑに属するものとを比べると，ⓑの方がⓐよりも計算過程で生じる実行可能の解の数が非常に多い．また，計算時間はⓐの方がⓑよりも短く，ⓑの方が短くなっているのは制約長が小さいときに数例あるのみである．ここでの結果から判断する限りでは，組み合わせトリーでの分枝方法は，先に分枝変数を0にした場合を調べるものの方が，ほとんどの場合計算時間が短いようである．

図—11，16，21 と図—12，17，22 のグラフを比べれば計算過程で目的関数を調べた実行可能解の数と計算時間は必ずしも比例していないが，それらと制約長との関係は互いによく似ていることがわかる．同じアルゴリズムであれば計算過程で目的関数値を調べた実行可能解の数が多ければ計算時間も長くなっていることが多いが，異なるアルゴリズムの間ではこのようには言えない．たとえば，アルゴリズム1は計算時間はアルゴリズム2より長いが，実行可能解の数はアルゴリズム2と同じ位である．これは，計算過程で目的関数値を調べた実行可能解の数の多少だけではアルゴリズムの効率を論じることができないということである．

アルゴリズム 1～3 では計算過程で生じた実行可能解の中にはネットワークが非連結になるものが含まれ得た．しかし，生じた実行可能解の中でネットワークが非連結になるものの割合を調べると，3種のアルゴリズムの間で大差はなかった．これは，計算過程でネットワークが非連結になる実行可能解が生じることに，アルゴリズム 1～3 の間の分枝方法の違いが影響していないということである．また，計算過程でネットワークが非連結になる実行可能解が生じるのを減らそうとすれば，アルゴリズム 4,5 のような方法をとる必要があることを表わしているとも言える．

解の性質についての考察のところで制約長の範囲を2つに分けたが，最適ネットワークでリンクがノード以外のところで交差する制約長の範囲では，5種類のアルゴリズムの計算時間の差はあまり大きくなく，制約長が変わっても計算時間は大きく変わっていない．一方，リン

クがノード以外のところで交差しない制約長の範囲では，アルゴリズム間の計算時間の差が大きく，制約長の変化によって計算時間が大きく変化している．

計算結果から見れば，アルゴリズム2とアルゴリズム4を適当に使い分けることがもっとも効率的だと言えよう．

4. 近似解法

ここでは厳密解法の計算例に表われた最適解の性質を再検討し，その考察を生かした新たな近似解法を提案する．また，それらの近似解法の特徴を計算例により検討し，実用性を調べる．

(1) 厳密解法による計算結果の再検討

a) forward 法の出発点の生成木

計算時間を短くするにはリンク数を減らし，リンクの組み合わせである解の数を少なくすることが有効である．forward 法では生成木に順次リンクを付加するという手順をとるが，ここでも forward 法の出発点とするような生成木を常に解のネットワークに含め，組み合わせを調べるリンクの数を減らすことを考える．

forward 法の出発点とする生成木は，その生成木の長さよりも制約長の大きな問題の最適ネットワークに常に含まれるものが望ましい．このような観点からどのようにして求めた生成木が forward 法の出発点の生成木にふさわしいかを調べてみる．

まず最初に考えられるのが経済木であるが，経済木が最適ネットワークに含まれない例が計算例に多く見られる．次に backward 法で得られる生成木について調べてみた．その結果，この生成木がその長さより制約長の大きい問題の最適ネットワークに含まれていないのは例題3の $L^c=2\,500$ のみであった．さらに考えられるのは，すべての生成木の中で目的関数値が最小のものである．これを最適生成木とよぶと，最適生成木は次の問題4を解くことによって得られる．

問題 4

$$\min Z=\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} d_{ij}(x_1, x_2, \cdots, x_m) \quad \cdots\cdots(13)$$

$$\text{s.t.} \sum_{k=1}^{m} x_k \leqq M^c \quad \cdots\cdots(14)$$

$$x_k=0 \text{ or } 1 \quad (k=1, 2, \cdots, m) \quad \cdots\cdots(15)$$

ここに，M^c はネットワーク中に含まれるリンク数の上限である．式 (13),(15) はそれぞれ式 (1),(3) と同じである．$M^c=n-1$ とすれば問題4の最適解は，リンク数が $n-1$ のネットワークの中で目的関数値が最小のもの，つまり最適生成木を与える．例題の中で，最適生成木がその長さよりも制約長の大きい問題の最適ネットワークに含まれていないのは，例題2の $L^c=2\,000$, $2\,500$ の場合であった．

以上の結果から，これからつくる近似解法では制約長がその長さよりも大きい場合，backward 法で得られる生成木を常に解のネットワークに含めることにし，組み合わせを調べるリンクから除くことにする．

b) backward 法による近似解と最適解との比較

ここでは基本的な近似解法である backward 法で求めた近似解と最適解とを比較することにより，どのような近似解法をつくれば近似解が最適解に近くなるかを検討する．

いま，backward 法で最大ネットワークから除かれる順にリンクに番号をつけるとしよう．ただし，生成木が残ったとき，これに含まれるリンクにはこのようにして番号をつけることができないので，これらには任意に番号をつけることにする．こうすれば backward 法の近似解と最適解との関係は 表—7～9 のように表わすことができる．これらの表は，どのリンクが最適ネットワークに含まれるかを示し，空欄はそのリンクが最適ネットワークに含まれないことを意味する．もし最適解と backward 法の近似解とが一致するなら，表のその制約長の列は空欄が上からつながっていることになる．さらにすべての制約長に対して最適解と backward 法の近似解が一致するならば，表の最適ネットワークに含まれるリンクを表わす欄は左下から右上へ階段状になる．

これらの表を見ると，例題1では backward 法の近似解と最適解はよく一致している．しかし，例題 2, 3 で

表—7 例題1の backward 法の解と最適解との関係

リンク	端点		L^c 527	600	800	1000	1200	1400	1600	1800	2000	2136	backward法の解 L	Z
1	2	7											1949	3668
2	1	6											1759	3678
3	4	8											1591	3691
4	1	4											1452	3708
5	6	7											1332	3739
6	2	5											1202	3773
7	3	4											1112	3835
8	3	6											988	3919
9	4	7											878	4005
10	6	8											798	4122
11	7	8											727	4229
12	1	2											657	4389
13	2	4											571	4533
14	2	3												
15	5	6												
16	5	8												
17	1	3												
18	3	5												
19	4	5												
20	5	7												
最適解	M		7	7	10	12	14	15	17	18	19	20		
	L		527	571	798	988	1198	1332	1591	1759	1949	2136		
	Z		5584	4533	4122	3919	3826	3739	3691	3678	3668	3659		

表—8　例題2の backward 法の解と最適解との関係

リンク	端点		L° 1490	1500	2000	2500	3000	3500	4000	4500	5000	5610	backward法の解 L	Z
1	3	8											5400	12880
2	4	7											5090	12920
3	2	9											4660	12985
4	1	4											4170	13085
5	4	6											3860	13190
6	1	5											3220	13305
7	7	9											3050	13450
8	5	8											2810	13650
9	3	7											2520	13860
10	1	2											1980	14090
11	2	3											1690	14570
12	4	8											1500	15710
13	3	5												
14	1	3												
15	3	4												
16	7	8												
17	2	5												
18	5	6												
19	6	7												
20	6	9												
最適解	M		8	8	10	12	14	15	16	17	18	20		
	L		1490	1500	1980	2460	2990	3430	3740	4380	4810	5610		
	Z		17990	15710	14090	13695	13430	13225	13120	13005	12940	12840		

表—9　例題3の backward 法の解と最適解との関係

リンク	端点		L° 1925	2500	3100	3700	4300	4900	5500	6100	6700	7080	backward法の解 L	Z
1	6	9											6813	17658
2	1	4											6059	17749
3	3	8											5793	17861
4	4	9											5276	17982
5	4	7											5066	18128
6	2	3											4659	18372
7	7	9											4144	18677
8	2	6											3645	19019
9	2	5											3316	19367
10	3	4											2941	19868
11	6	7											2501	20640
12	1	2											1977	22464
13	5	7												
14	1	3												
15	4	5												
16	4	6												
17	3	6												
18	6	8												
19	8	9												
20	2	4												
最適解	M		8	9	11	12	14	15	16	18	19	20		
	L		1925	2406	3086	3603	4274	4836	5276	6059	6326	7080		
	Z		22474	20720	19370	18942	18543	18196	17983	17749	17709	17618		

は backward 法の近似解が最適解でないことの方が多い．そこで，どのような場合に backward 法の近似解と最適解が一致しないかを考える．

backward 法では一度除かれたリンクが再び解のネットワークに含まれることはない．したがって，最大ネットワークから制約長を満たすまでリンクを除いたとき，除かれたリンクのうち長さの短いものをネットワークに含めても制約長を越えないことがある．リンクを付加すれば必ず目的関数値は小さくなるので，このようなときには近似解は最適解と一致しない．さらに backward 法の近似解について調べると次のことがわかった．

最大ネットワークから除かれるリンクが少ない場合，backward 法の近似解と問題 4，すなわち，リンク数制約の問題の最適解はよく一致する．また，backward 法の近似解はその解のネットワーク長を制約長として与えた問題に対しては比較的よい解で，最適解であることが多い．そして最適解でないとき，最適ネットワークに含まれるリンクの数は，backward 法の近似解のネットワークに含まれるリンクの数より大きい場合が非常に多い．

backward 法の近似解と最適解が一致しないのは，近似解のネットワーク長と制約長との差が大きいか，近似解のネットワーク長より短いネットワーク長でリンク数が多く目的関数値がより小さい解があるかのどちらかであることがほとんどである．

Scott の backward 法では，ここでの backward 法の近似解に netran とよぶネットワーク変換を施し，よりよい解を求めようとしている．これで得られるのはせいぜい問題 4 の最適解であるが，問題 4 の最適解が問題 1 の最適解にならないことが多いことがわかった．すなわち，backward 法の近似解のネットワーク長以下でリンク数の多い実行可能解を探索できるアルゴリズムでなければ，最適解ないしはそれに近い解を得ることができないであろう．

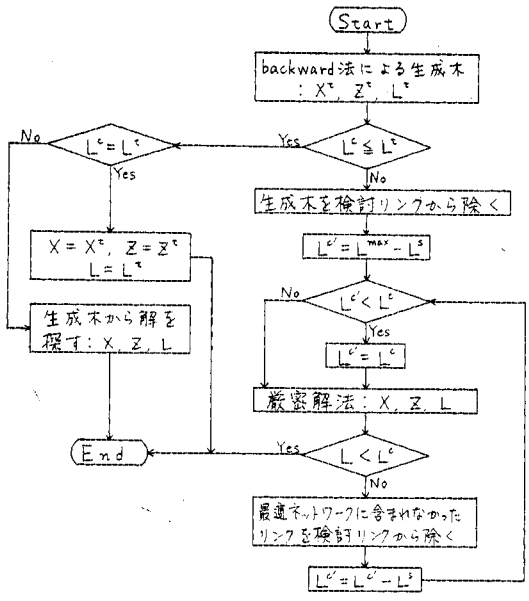
（2）　アルゴリズム

(1) の検討に基づき 2 種類の近似解法を開発した．ここではこれらのアルゴリズムについて述べる．

a)　アルゴリズム 6

ある最適ネットワークからいくつかのリンクを除くことによって，それよりも制約長の小さい問題の最適ネットワークが得られることが多いことを前に述べた．これは **表—7～9** を見ればよくわかる．これらの表でリンクが最適ネットワークに含まれることを表わす欄が横につながっている場合が，最適ネットワークからリンクを除いてそれよりも制約長が小さい問題の最適ネットワークが得られる場合に相当する．表をみると欄が途切れても間の空欄は 1 ないし 2 の場合が多い．

これらのことから，制約長が与えられればそれよりも制約長の大きな問題の最適ネットワークを最大ネットワークとし，その中から実行可能解を探せばよいのではないかと考えられる．制約長を減らしてゆくというようにこれを繰り返せば，最大ネットワークから与えられた制約長の問題の近似解を得ることができる．厳密解法のア



図—23 アルゴリズム6のフローチャート

ルゴリズムでは最大ネットワークから除くリングが少ないとき，計算時間は非常に短かった．だから制約長をある刻みで減らしてゆき，厳密解法を繰り返し使用しても近似解を求めるための計算時間はあまり長くならないであろう．このような方法であれば，最大ネットワークから複数のリンクを順次除いてゆき，除くリンクの組み合わせもリンク長を考慮して決めることになる．そこで，backward 法の短所を改善することができると考えられる．

このような考え方に基づき作成したのがアルゴリズム6である．アルゴリズム6のフローチャートは 図—23 である．図の X^t は backward 法で求めた生成木を表わし，L^t はそのネットワーク長，Z^t は目的関数値である．また，L^s は制約長を減らす刻みで，$L^{c'}$ は厳密解法による計算を繰り返すときの暫定的な制約長である．

計算手順を説明すると，まず backward 法で生成木を求める．もしその生成木の長さが制約長より大きければ，制約長以下の長さの生成木の中でもっとも目的関数値が小さいものを探し，それを近似解とする．これは制約長が backward 法で求めた生成木の長さより小さい場合，先に述べた方法では解を求めることができないからである．そしてこのように制約長が小さいとき，最適解は生成木であることがほとんどなので生成木の中からもっとも目的関数値の小さい実行可能解を探すことにした．

制約長が backward 法で求めた生成木の長さより大きいときは，その生成木を構成するリンクを検討リンク，すなわち，建設可能リンクのうち，それ以後の計算で組み合わせを調べるリンクから除き，常に解のネットワークに含める．次に制約長の値を L^{max} より L^s だけ小さい値にし，厳密解法によって検討リンクの組み合わせの中から最適なものを選ぶ．そして制約長の値を L^s ずつ減らし，前回の最適ネットワークに含まれた検討リンクを今回の検討リンクとし，与えられた制約長に対する近似解が得られるまで繰り返し計算を行う．

このフローチャートにしたがって計算を行うと，厳密解法を使う繰り返し計算の最後で与えられた制約長の近似解を求めるとき，前回の繰り返し計算の制約長と今回の制約長との差が小さくなることがある．実際にはこの差が L^s の 50～150% になるように調整している．それは刻みが小さすぎれば backward 法と同じ短所を持つことになり，大きすぎれば計算時間が急激に長くなるからである．

制約長を減少させる刻み L^s は正の値を持ち，このアルゴリズムで問題を解くときのパラメーターとなる．制約長の刻みが非常に小さいときは，リンクを1本ずつ除き，除くリンクの長さは問われないことになる．この場合アルゴリズム6は backward 法とまったく同じになるが，すでに生成木を得るまで backward 法の計算を行っているので再び同じ計算を繰り返すことになる．これとは逆に制約長の刻みを極端に大きくすると，1回の厳密解法による計算で与えられた制約長を満たす解を求めることになり，厳密解法そのものになる．ただし，backward 法で求めた生成木を検討リンクから除いていることが異なる．なお，検討リンクから除くということは，その後の計算過程で，そのリンクを $x_k=0$ あるいは $x_k=1$ に固定することである．

厳密解法による計算を行うときにはネットワークは常に連結なのでアルゴリズム4やアルゴリズム5を使う必要はなく，アルリズム6の中の厳密解法にはアルゴリズム2を使うことにする．

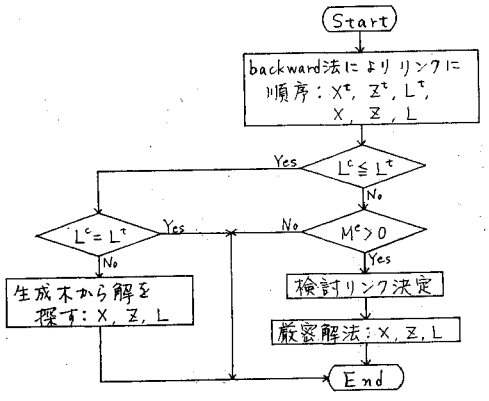
b) アルゴリズム7

表—7～9 を見るとさらに次のことがわかる．それは，最適解と backward 法の近似解が一致しないとき，近似解には含まれず最適解には含まれているリンクは，backward 法で除かれたリンクのうちリンク番号の大きなものであることが多いことである．また，backward 法の近似解には含まれ最適解には含まれていないリンクは，近似解に含まれたリンクのうちリンク番号の小さいものであることが多いこともわかる．これは backward 法で，たとえば k リンクまで除いて実行可能解になったとき，k リンクの前後のいくらかのリンクの組み合わせを考えることによって，最適解を見つけることができる可能性が大きいことを意味している．これを利用したのがアルゴリズム7である．

アルゴリズム7では backward 法で最大ネットワークから除かれる順にリンク番号をつける．そして k リンクまで除いたときに実行可能解になるなら，任意に正整数 M^e を定め 1, 2, …, $k-M^e$ のリンクは近似解のネットワークに含めないことにし，$k+M^e+1$, …, $m-n+1$ のリンクと残りの生成木をなすリンクを近似解のネットワークに含めることにする，$k-M^e+1$, …, k, …$k+M^e$ のリンクが検討リンクで，この組み合わせの中から目的関数を最小にする実行可能解を厳密解法で探す．M^e は正の整数で，検討リンクの数を決めるパラメーターである．M^e が与えられれば，もっとも多いときには $2M^e$ の検討リンクの組み合わせを調べることになる．また，backward 法で得られる生成木は，その長さより制約長が小さくない限り，つねに近似解のネットワークに含めることにする．M^e に十分大きな値を与えると，近似解のネットワークに含めないことに決めるリンクがなくなり，アルゴリズム7は厳密解法と同じになる．ただし，backward 法で得られる生成木を検討リンクにしないことが異なる．逆に M^e の値を小さくすれば検討リンク数は減り，$M^e=0$ で検討リンクがない場合を表わすものとすると，このときは backward 法そのものである．

アルゴリズム7のフローチャートは 図—24 のようになる．図のリンクに順序づけを行ったときの X, L, Z は backward 法による近似解とそのネットワーク長，目的関数値である．

計算は，まず backward 法でリンクに順序づけを行う．もし制約長が backward 法で得られた生成木の長さより小さいときは，アルゴリズム6と同じように生成木の中から近似解を求める．制約長の長さが backward 法で得られた生成木の長さより大きいときは，パラメーター M^e の値によってリンクを近似解のネットワークに含めるものと含めないもの，それに検討リンクの3種に分ける．実際にはたいていの場合 backward 法で生成木が得られるまで計算する必要はなく，近似解のネットワークに含めないことにするリンクと検討リンクが決まるまで計算すればよい．検討リンクが決まれば，その組み合わせを厳密解法で調べて近似解を得る．ただし，$M^e=0$ のときは backward 法の近似解がそのままアルゴリズム7による近似解となる．なお，厳密解法にはアルゴリズム2を使用する．



図—24　アルゴリズム7のフローチャート

（3）計　算　例

アルゴリズム6は L^s，アルゴリズム7は M^e の計算時間と得られる解の良し悪しを左右するパラメーターを持っている．まず，これらの値をどのようにすればよいかを検討する．

先の例題の中から任意に問題を選び，パラメーターの値を変えて解いてみた．その結果，アルゴリズム6の L^s については，値を大きくして厳密解法を使う繰り返し計算の回数が少ない方が一般に得られる近似解の目的関数値は小さかった．また，計算時間は L^s のある値で最短になり，L^s がそれより大きくても小さくても長くなる傾向が見られた．しかし，問題が簡単な場合には，L^s が大きいほど計算時間が短くなることもわかった．一方，アルゴリズム7の M^e は値が大きいほど計算時間は長くなるが，得られる近似解の目的関数値は小さかった．これは，M^e の値が大きければ検討リンク数が大きくなり，より多くのリンクの組み合わせが調べられるからである．

これらの結果をもとに L^s は平均リンク長の5倍，$M^e=5$ として例題1〜3を解いてみた．パラメーターの値をもっと大きくしても計算時間は問題にならないが，近似解法としての特徴が薄れて厳密解法に近くなるのでこのような値にした．

例題の計算結果は 表—10〜12 に示すとおりである．比較のために厳密解法のアルゴリズム4とともに計算時間をグラフにしたのが 図—25〜27 である．これを見ればアルゴリズム 6, 7 による計算時間は厳密解法のアル

表—10　近似解法による例題1の計算結果

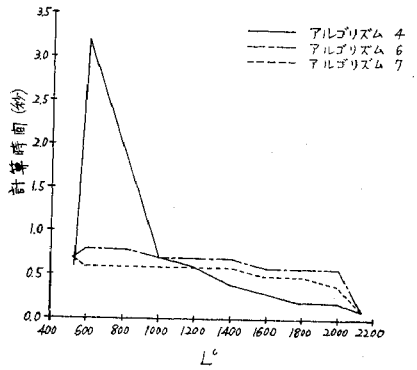
L^c	アルゴリズム6		アルゴリズム7		最適解
	計算時間 (sec)	Z	計算時間 (sec)	Z	Z
527	0.7	5584	0.7	5584	5584
600	0.8	4533	0.6	4533	4533
800	0.8	4122	0.6	4122	4122
1000	0.7	3919	0.6	3919	3919
1200	0.7	3826	0.6	3826	3826
1400	0.7	3739	0.6	3739	3739
1600	0.6	3691	0.5	3691	3691
1800	0.6	3678	0.5	3678	3678
2000	0.6	3668	0.4	3668	3668
2136	0.1	3659	0.1	3659	3659
計	6.3		5.2		

表—11 近似解法による例題2の計算結果

L^c	アルゴリズム6		アルゴリズム7		最適解
	計算時間(sec)	Z	計算時間(sec)	Z	Z
1490	1.2	17990	1.2	17990	17990
1500	0.7	15710	0.7	15710	15710
2000	1.1	14090	0.7	14090	14090
2500	1.0	13695	0.7	13735	13695
3000	1.0	13430	0.8	13430	13430
3500	0.8	13225	0.8	13305	13225
4000	0.8	13120	0.7	13120	13120
4500	0.7	13005	0.7	13005	13005
5000	0.7	12940	0.6	12940	12940
5610	0.1	12840	0.1	12840	12840
計	8.1		7.0		

表—12 近似解法による例題3の計算結果

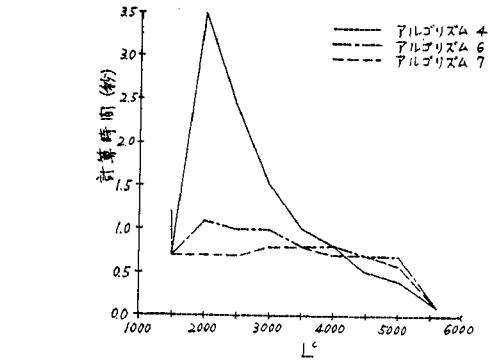
L^c	アルゴリズム6		アルゴリズム7		最適解
	計算時間(sec)	Z	計算時間(sec)	Z	Z
1925	0.8	22474	0.8	22474	22474
2500	1.2	21128	0.7	21128	20720
3100	1.1	19370	0.8	19868	19370
3700	1.1	18958	0.9	18942	18942
4300	1.1	18616	1.3	18543	18543
4900	1.2	18196	0.9	18196	18196
5500	1.0	17982	0.7	17982	17982
6100	0.7	17749	0.6	17749	17749
6700	0.7	17709	0.6	17709	17709
7080	0.1	17618	0.1	17618	17618
計	9.0		7.4		



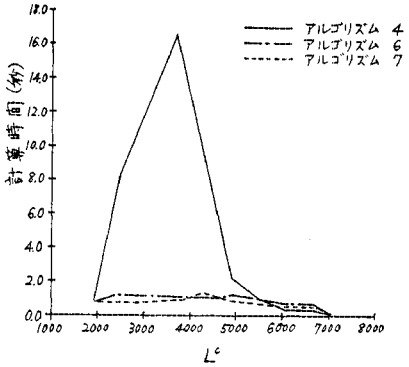
図—25 近似解法による例題1の計算時間

ゴリズムと比べるときわめて短く，制約長が変わっても計算時間は大きく変わっていない．制約長が大きいときには厳密解法よりも計算時間が長くなっているが，これは，たとえば，アルゴリズム6の場合，どのような制約長の問題に対しても backward 法によって生成木が得られるまで計算を行っているからだと推察できる．また，求めた近似解は最適解であることが多く，backward 法よりもはるかによい結果であった．

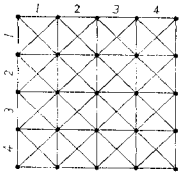
次にネットワークを構成するノード，リンクが多くなったときに計算時間がどのように変化するかを調べてみる．そのため，例題4として図—28のネットワークを使用する．このネットワークは全体が大きな四辺形になるが，この四辺形の一辺に含まれるリンク数を1～4にして最大ネットワークの大きさを変える．すなわち，一辺のリンク数が1のときは左上の4ノード，6リンクを使用し，一辺のリンク数がふえれば最大ネットワークを右および下へ拡大するものとする．



図—26 近似解法による例題2の計算時間

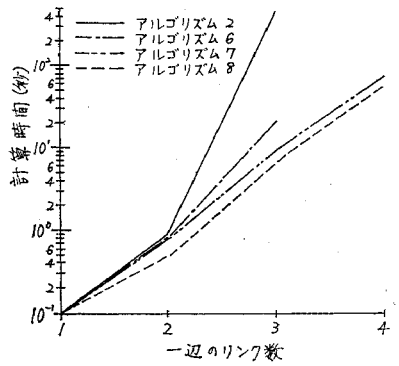


図—27 近似解法による例題3の計算時間



図—28 例題4の最大ネットワーク

L^s は平均リンク長の5倍，$M^e=5$ とし，制約長は



図—29 例題4の計算時間

$L^c=(L^{max}+L^{min})/2$ と決め，比較のために厳密解法のアルゴリズム2と backward 法（これをアルゴリズム8とする）でも解くことにする．

計算の結果，各アルゴリズムで計算に要した時間は**図—29**のようになった．ただし，アルゴリズム 2,6 では一辺のリンク数が4のとき，計算時間が非常に長くなるので解いていない．**図—29** を見れば，アルゴリズム 2,6 では計算時間の軸を対数でとったグラフでも，下に凸な曲線を描いて計算時間が増加しているが，アルゴリズム 7,8 ではほぼ直線状であり，アルゴリズム 2,6 ほど急激に計算時間が増加していないことがわかる．なお，アルゴリズム 6～8 の近似解はほとんど最適解であった．

以上のことからアルゴリズム 6,7 の特徴をまとめると次のようになる．

① 計算時間と近似解が最適解である可能性を左右するパラメーターを持ち，自由にその値を選択できる．

② 最大ネットワークが与えられれば，どのような制約長の問題もほぼ同じ位の計算時間で解くことができる．

③ 最大ネットワークに含まれるノード，リンクが多くなったときの計算時間の増加の傾向は，アルゴリズム7では backward 法と同じである．アルゴリズム6ではそれよりも急激に増加するが厳密解法ほどではない．

① がこれらのアルゴリズムの最大の利点である．パラメーターの決め方によって単純な backward 法のようにもなり，厳密解法のようにもなる．そこで，実際の問題を解くときに，必要な解の精度と許される計算時間に応じてパラメーターの値を決定することができる．③についてはパラメーターの値によっては変わることが予想され，これはここで採用したパラメーターの値について言えるものである．ネットワークが大きくなることによるアルゴリズム 6,7 の計算時間増加の傾向は，パラメーターの値により backward 法と厳密解法の間で変化すると考えられる．

5. あとがき

最適交通ネットワーク問題の解を求める厳密解法は，計算時間が膨大になり実用的でないことから最近ではあまり研究されていなかった．しかし，実用性を重視した近似解法をつくろうとするとき，最適ネットワークがどのようなものかを知っていなければならない．そうでなければ非効率的で，求めた近似解が最適解よりかなり悪いような近似解法をつくってしまうかも知れない．近似解法の良否を判断するためには，近似解と対比するための最適解を求めておくことが必要であろう．また，厳密解法を近似解法に応用できるのではないかということから，ここでは最初に厳密解法のアルゴリズムの改良をいくつか試みた．その結果，とくに経済木構成手法を導入し，ネットワークを連結にしてから実行可能解を調べてゆく方法は十分満足できるものであった．

厳密解法によって求めた種々の例題の最適解を検討すると，ある最適ネットワークがすでに得られているとき，その最適ネットワークからそれを構成しているいくらかのリンクを除けば，ネットワークに含まれるリンクの合計長がより短く制約された場合の最適ネットワークが得られる可能性が高いことが知られた．また，backward 法による近似解の近傍に最適解がある可能性が高いことにも経験的に気づいた．すなわち，これが近似解法をつくる際の指針となった．

次に提案した近似解法アルゴリズム 6,7 は，これまでにあった近似解法の考え方に厳密解法を持ち込んだものと言える．パラメーターの値によっては，もっとも単純な backward 法のようにもなり，厳密解法のようにもなるという特徴を持っている．言い換えれば，計算時間あるいは解の精度を自由に制御できるのであり，厳密解に対する近似度の保証が与えられるともいえる．

なお，近似解法については，ある地域の大量輸送機関網計画に応用して，十分実用に耐えるものであることを確認している．

参考文献

1) Scott, A.J. : The Optimal Network Problem; Some Computational Procedures, Transportation Research, Vol. 3, pp. 201～210, 1969.
2) Ridley, T.M. : An Investment Policy to Reduce the Travel Time in a Transportation Network, Transportation Research, Vol. 2, pp. 409～424, 1968.
3) Hoang Hai Hoc : A Computational Approach to the Selection of an Optimal Network, Management Science, Vol. 19, No. 5, January, 1973.
4) 西村 昂・日野泰雄：最適ネットワーク構成に関する一考察，土木学会論文報告集，第 250 号，pp. 85～97, 1976 年6月.
5) 運輸経済研究センター：神奈川県内新交通輸送体系確立に関する調査報告書，昭和 48 年3月.
6) 飯田恭敬：最適道路ネットワークの構成手法，土木学会論文報告集，第241号，pp. 135～144, 1975 年9月.

(1976.8.17・受付)