第 3 回 最適化ゼミ

ゼミ担当者　：　佐藤　史隆，平井　聡，林　俊行
指導院生　　：　花田　良子，勝崎　俊樹
開催日　　：　2003 年 5 月 13 日

1　準ニュートン法

多変数の非線形目的関数を制約なしで最小化する問題に対しては，最急降下法，ニュートン法などさまざまな方法がある．ここで説明するのは準ニュートン法で，ニュートン法ではなかった「大域的収束性」を持った解法である．以下に準ニュートン法の特徴であるヘッセ行列の近似法について説明する．

目的関数 $f(x)$ において $y^k = \nabla f(x^{k+1}) - \nabla f(x^k)$, $s^k = x^{k+1} - x^k$ として次の 2 つの条件を要請する．

- セカント条件
  ヘッセ行列を H としたとき
  $$Hf(x^k)s^k = y^k$$
  を得ることができ，これと類似で新しい近似行列 B^{k+1} に
  $$B^{k+1}s^k = y^k \tag{1}$$
  が成り立っていることを要請する．(1) をセカント条件という．
- 正定値条件
  良好な局所的収束のためにも，降下方向を得るためにも，B^k は正定値であることが望ましい．よって B^k が正定値のとき B^{k+1} も正定値になるような更新を行う．

1.1　ＢＦＧＳ方式

準ニュートン法における B^k の更新公式として，様々な方法が提案されているが，なかでも次に紹介するＢＦＧＳ方式は，その理論および実用の両面における有効性から最もよく使われているものである．その公式は次の通りである．

$$B^{k+1} = B^k - \frac{B^k s^k (B^k s^k)^\top}{(s^k)^\top B^k s^k} + \frac{y^k (y^k)^\top}{(s^k)^\top y^k} \tag{2}$$

2　離散最適化問題の概要

2.1　組合せ問題とは

離散最適化問題は，一部の変数あるいはすべての変数に整数条件が付加された最適化問題である．特に組合せ的な性質を持つ場合には，組合せ最適化問題という．

組合せ最適化問題とは，いろいろな組合せの中で，与えられた制約条件を満たし，かつ，目的関数を最小化（最大化）するものを見つける問題である．組合せ最適化問題には TSP，スケジューリング問題，割当問題，クラス編成問題，およびナップザック問題があり，その解法には解の効率的列挙などがある．ここでは，TSP，スケジューリング問題について紹介し，それらを解くためのアルゴリズムにはどのようなものがあるか簡単に紹介する．

2.2　TSP

TSP とは，Fig. 1 のように，都市数とそれぞれの都市間の道のりがわかっているとき，セールスマンがＮ個ある都市を必ず一度だけ通って巡回したとき，総経路長を最短にするような都市の巡回経路を見つける問題である．都市数がＮ個のとき，考えられる巡回経路の種類は，Ｎ個の種類は，Ｎ個の要素の最適な並び替え（数珠　順列）であるので，$\frac{(N-1)!}{2}$ である．例えば，Ｎ＝ 10 の場合，約 18 万だが，Ｎ＝ 20 となると約 6×10^{16} となってしまう．このことからわかるように，Ｎが大きくなると，この組合せは爆発的に多くなる．このような現象を組合せ爆発という．この場合も，すべての巡回経路を調べることは実際的には不可能で，都市の配置に法則性がなければ厳密解を求めることはできない．

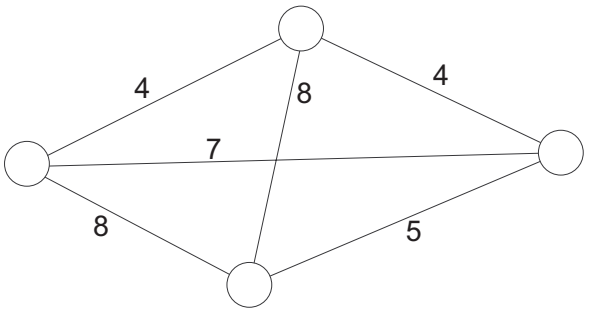

Fig. 1　TSP

TSP のバリエーションとして，非対称 TSP，球面 TSP，S 人 TSP がある．

- 非対称 TSP
  都市 a から都市 b に行く時と，都市 b から都市 a

に行く時で巡回経路が異なる問題．一般的には，都市 a と都市 b の直線距離に重みをかけた値をその都市間のコストとし，巡回の方向によって異なる重みを設定する．一方通行用の道路でつながれた都市間の道で，異なる道路を使うと考えるとイメージしやすい．

- 球面 TSP
  都市が球面上に配置されていると考える問題．平面上では，もっとも右上（左下）に位置する都市ともっとも左下（右上）に位置する都市は，直線距離のコストが大きいが，球面では逆に小さい場合もある．

- S 人 TSP
  一人で巡回するのではなく，S 人（複数）で都市を巡回する問題．各都市を一回だけ通過する．

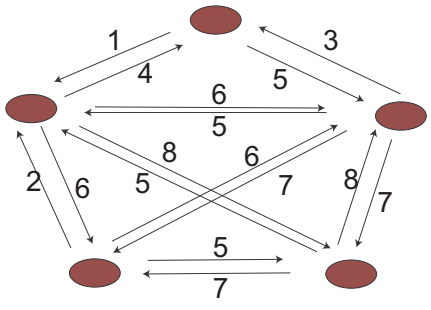

Fig. 2　非対称 TSP

2.3　生産スケジューリング問題

2.3.1　ジョブショップスケジューリング問題

ジョブショップスケジューリング問題（JSP）は以下のように定義される．

- N 個の仕事を M 台の機械で処理する．
- 各仕事を処理する機械の順序，すなわち技術的順序は任意に与えられ，各仕事の各機械上での処理時間は与えられているとする．
- すべての仕事を処理し終えるまで総所要時間を最小にする各機械上での各仕事 (作業) の処理順序の決定することが目的である．ただし二つの作業を同一機械で同時に実行することはできない．

以上をふまえたスケジューリング問題は Fig. 3 のように最短で処理し終えるような仕事の順序を決める問題である．

またジョブショップスケジューリング問題のうち，Fig. 4 に示すようにすべてのジョブが同じ順に機械を占有するような問題をフローショップスケジューリング問題と呼ぶ．この問題は，現実の工場では，もっともよく存在するものであり，コンベアを用いた流れ作業などがこれに相当する．

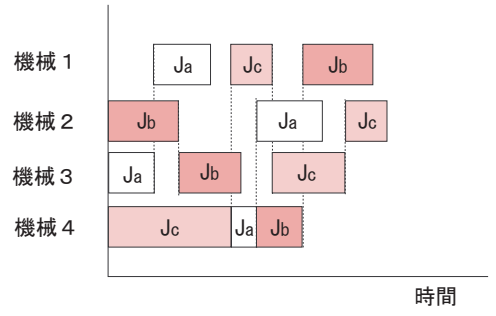

Fig. 3　ジョブショップスケジューリング問題

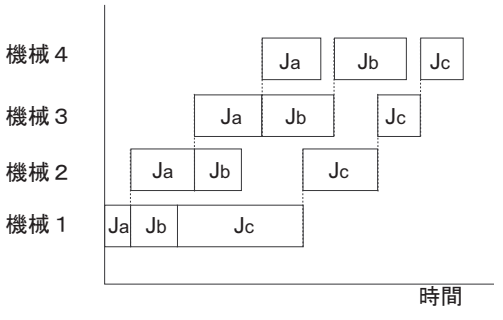

Fig. 4　フローショップスケジューリング問題

2.3.2　プロジェクトスケジューリング問題

プロジェクトスケジューリング問題は，ジョブショップ・スケジューリング問題をさらに拡張した問題で，工場におけるスケジューリング問題のみではなく，建築工事や製品開発などより広い適用範囲をあげることができる．この問題では，工程に属する作業が直列である必要がなく，分岐や合流があっても構わない．また，利用する資源は，同時に複数の作業で共有することが可能であるものとする．プロジェクト・スケジューリング問題は，PERT（Problem Evaluation and Review Technique）によって一般的に扱われる．

2.4　ナーススケジューリング問題のアルゴリズム

ナーススケジューリング問題は，看護婦 i の j 日の勤務 x_{ij} を要素とする n × m 行列を決定する問題であると言える．ここでは n は看護婦数，m は当該月の日数，x_{ij} は基本的には深夜勤，準夜勤，日勤，休日等が入る．(Fig. 5 参照)

そして例えば，看護婦の数は 15 名，スケジュールの数は 30 日，1 日に必要な深夜勤は 2 人，1 日に必要な準夜勤は 2 人，などという各種制約条件を設定し，各看護婦と全体の評価項目を与える．

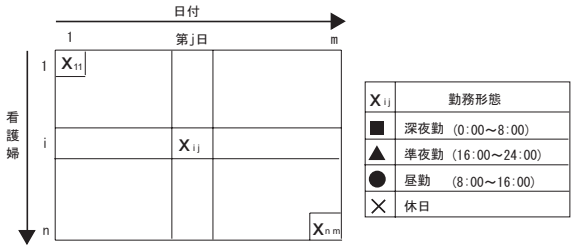

Fig. 5　NSP のスケジュール表現と勤務形態

2.5　組み合わせ最適化に用いられるアルゴリズム

組み合わせ問題において，それぞれの組み合わせに対する計算が，問題の規模（例えば問題の要素数）に関する多項式時間で実行できるアルゴリズム[1]があるならば，その問題はクラス NP[2]に属するという．ただし，組み合わせの総数が爆発的に多くなる問題については，仮に多項式で表せるアルゴリズムが存在したとしても，すべてに対する解を多項式時間で調べ尽くすことは不可能である．このように，クラス NP に含まれるどの問題と比べても難しさが同等か，もしくはそれ以上であり，しかも，多項式アルゴリズムがまだ見つかっていない問題のクラスを NP 困難と呼ぶ．NP 困難である問題の計算量は，一般に問題の規模に対して指数関数的に増加する．TSP，スケジューリングの問題は，NP 困難かつ組み合わせが爆発的に増える非常に難しい問題である．

これらの組み合わせ最適化では，連続最適化問題のように微分の概念に基づく古典的な最適化手法を直接利用することはできない．したがって，組み合わせ最適化問題の解放は連続最適化問題のアルゴリズムと本質的に異なるものとなり，基本的には，解となり得るすべての場合について調べ上げるという列挙的手法を取らざるを得ない．しかし，NP 困難で，しかも組み合わせの総数が有限であっても膨大な数にのぼる問題は多い．

これらの問題に関しては，厳密な最適解を求めることはあきらめ，良い近似最適解が比較的短時間で得られるなら，それで満足しようという考え方が一般的には受け入れられている．その方法はヒューリスティック解法，あるいは発見的解法といわれており，その有力な手法として第一回・第二回で説明した欲張り法がある．欲張り法は，有限である実行可能解に対する目的関数から値を計算して，求まった値がその段階で最適であればその値を解として取り入れるという方法を繰り返し行い，最適解を求めていくという動作である．この方法では，解となり得る点についてすべて調べるわけではないので，問題の（大域的）最適解が得られる保証はないが，組み合わせが多い問題に対しては実際に解を効率的に得ることができる．

ヒューリスティック解法は，一度局所的解に陥るとそこから抜け出すことができなくなる．そこで，改悪となるような解への移動を許すことによって，好ましくない局所解に補足されることを避ける考え方がある．その手法としてメタヒューリスティック解法がある．

メタヒューリスティック解法の基本型は局所的探索法であるが，得られた近似最適解に対して，さらに近傍で部分的な修正を繰り返し加えることにより，より良い近似最適解を得る方法である．ただし，最終的に得られる解のよしあしは近傍の定め方による．一般に，近傍を大きく取れば，小さい近傍を用いた場合に比べ，現時点における解よりもより良い解が見つかる可能性が大きい．そのため，最終的に得られる局所最適解の質は良くなる．しかし，近傍を大きく取れば計算量が多くなってしまうので，近傍の取り方を調節する必要がある．

局所探索法を基にした代表的な手法としては，以下のようなものがある．

- タブー探索法

  近傍内での最良の解を求め，これが改悪であっても，現時点での解を更新することを基本とする．しかし，この操作をそのまま実行すると循環が生じやすいので，タブーリスト T を用意しておき（例えば，過去何回かの繰り返しで，訪れた解のリスト），T 内への遷移を禁止することで循環を抑制している．

- SA（Simulated Annealing）

  EC ゼミで説明済みである．

- GA（遺伝的アルゴリズム，Genetic Algorithm）

  EC ゼミで説明済みである．

2.6　TSP に用いられるアルゴリズム

2.6.1　TSP と列挙的手法

TSP に用いられる列挙法には動的計画法などがある．

動的計画法は，大きな問題を小さな問題に分けて，それをひとつひとつ解いていくことで大きな問題を解く方法である．ここでは，動的計画法について簡単に説明する．

あらかじめ出発点（＝終点）を定めておき，巡回路の最終点に戻る直前に訪問する可能性のあるすべての点 x について，出発点から点 x までの最短な部分巡回路を考える．そして，その部分巡回路を求めるには，同様の考え方を用いて，点 x にたどり着く直前の点として考えられるすべての点 y までの最短な部分巡回路を調べればよい．この考え方を用いて，出発点までさかのぼっていくと，最終的には解きたい問題の解を得ることができるという手法である．

[1]アルゴリズムの計算量が，問題の規模，例えば要素数 N 個を考えたとき，$N^5 + N^2$ のような，N のべき乗（$logN$ を含む）の式で表されるならば，そのアルゴリズムは多項式アルゴリズムという

[2]Nondeterministic Polynomial time，クラスとは集まり

TSP は組み合わせ爆発を起こしてしまう問題なので，都市数に対して指数関数的に計算量が増加するということは前節で述べた．動的計画法は，ある程度の規模の問題なら適用可能であるが，同じ列挙法で解くならば，より効率の良い分枝限定法を用いたほうが良い．しかし，分枝限定法でも解けないような大規模な問題になると，列挙法ではなく，次に説明するヒューリスティック解法を用いると良い．

2.6.2 TSP とヒューリスティック解法

ヒューリスティック解法の 1 つである欲張り法に基づく手法に最近近傍法という方法がある．この方法では，ある都市から出発して，まだ訪問していない隣接した都市の中で，現在の都市に最も近い接点を次々と移動していく．Fig. 6 の例の最適解は 23 である．この例に対して，まず，都市 1 を出発点として最近近傍法を適用すると，巡回路 1→3→2→4→1，巡回路長が 23 が得られる．これはこの問題の最適解になっている．次に都市 3 を出発点とする場合を考える．都市 2 からの最近近傍法を適用すると，巡回路は Fig. 7 のように 3→1→2→4→3 となる．しかし，この巡回路は長さ 24 であるから最適解ではない．最近近傍法は禁止的に巡回路を構成していくので，得られる巡回路は一般に最適化解と比べてかなり悪くなることが多いが，次に説明するメタヒューリスティック解法で，さらに部分的に修正することができる．

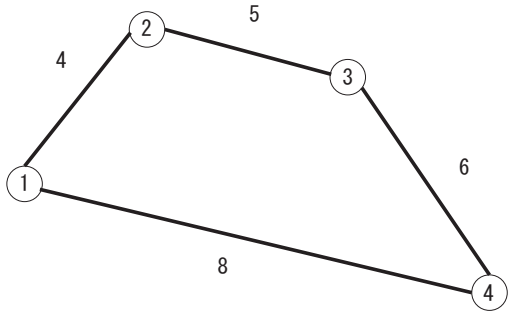

Fig. 6　巡回路 1→3→2→4→1

2.6.3 TSP とメタヒューリスティック解法

メタヒューリスティック法は，得られた局所解を部分的に修正を加えるものである．メタヒューリスティック解法の種類については，前節で述べた．これは TSP についてのメタヒューリスティック解法における近傍の取り方について述べる．巡回路が Fig. 8 のように得られているとする．そのとき隣り合わない任意の 2 本の枝を巡回路 Fig. 8 から取り去り，別の 2 本の枝を付け加えて得られる巡回路を考える．ここで述べている 2 本というのは，TSP における近傍のことであり，2 本の場合は 2-opt 近傍[3]という．2-opt の場合，都市数が N 個の場合

[3]連続最適化問題の場合は，ある点を中心とした十分小さな半径 d の円の内部を近傍と呼ぶが，離散的な問題については，ある点 x に隣接した点のうちいくつの点を x の近傍とするという定義になっている

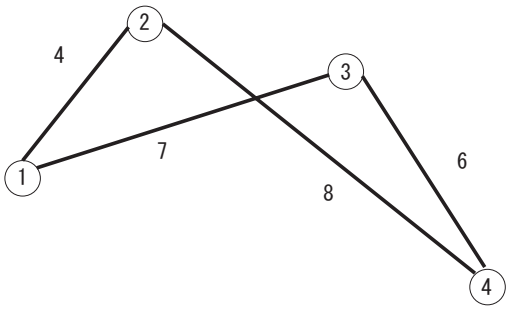

Fig. 7　巡回路 3→1→2→4→3

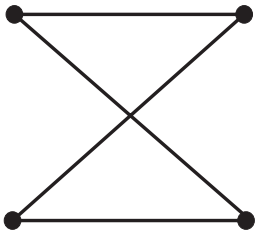

Fig. 8　巡回路

新たに考えられる巡回路の数は $N(N-3)/2$ 個である．この操作を何度繰り返していると，徐々に解が良くなる．

この作業を行うと，Fig. 9 のように最適解ではない巡回路が求められる場合もあるが，作業を繰り返し行うと Fig. 10 のように，この問題の最適解が求めることができる．

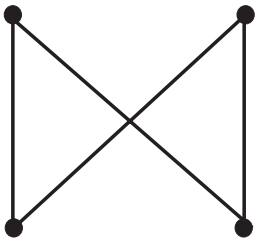

Fig. 9　新たに生じた局所解の巡回路

先ほどの例の解である Fig. 7 の巡回路 3→2→4→1→3 についてこの方法を用いる．都市数が 4 個であるので新たにできる巡回路は $4(4-3)/2 = 2$ 個で Fig. 11，Fig. 12 である．このうち，Fig. 11 は Fig. 6 と同じ最適解である．

ただし，2-opt 近傍の場合，近傍がとても小さいので，

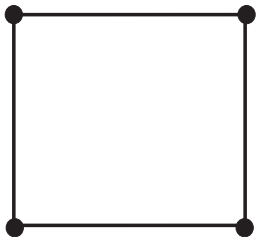

Fig. 10　新たに生じた最適解の巡回路

都市数が大きくなると，良い解が得られないことが多々ある．問題の規模に合わせて 3-opt 近傍，4-opt 近傍など，近傍を広げるとより厳密な解に近づくが，近傍を大きくしすぎると，新たに生じる巡回路の個数が爆発的に増加する．そのため 4-opt 近傍以上の近傍が用いられるのは非常に稀である．

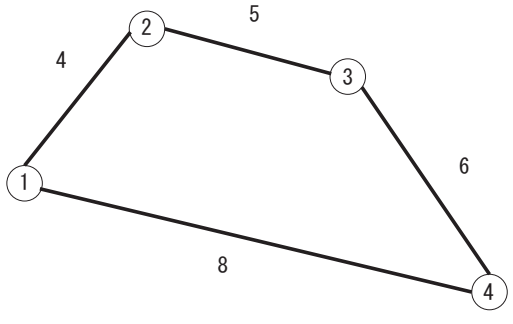

Fig. 11　巡回路 1→3→2→4→1

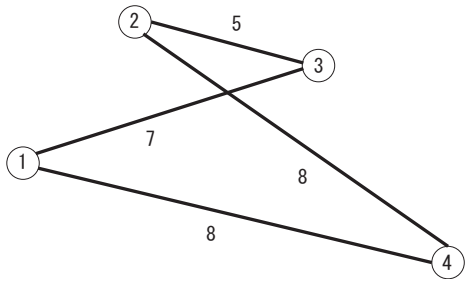

Fig. 12　巡回路 3→4→1→2→3

2.7　プロセススケジューリングのアルゴリズム

コンピュータに対して複数の実行可能な仕事がある場合，どの順序でＣＰＵを割り当てるかという問題がプロセススケジューリング問題である．プロセススケジューリング問題において，特に重要なことは以下の 4 点であると考えられている．

- スループット[4]の向上
- 応答時間の短縮
- 公平な割り当て
- 応答時間が予測可能であること

CPU を割り当てられたプロセスが，実行を終了する，もしくは入出力を行うまで CPU を独占するとしたら，他のプロセスの応答時間がそのプロセスの応答時間がそのプロセスの特性に大きく依存してしまうことになる．よって，現在の多くのオペレーティングシステムでは，プロセスの実行を途中で中断し，別のプロセスの実行を開始させるということを行う．これを横取り (preemption) という．スケジューリングは横取りを許す，許さないで性質が大きく異なる．横取りを許すスケジューリングを横取り可能 (preemptive) スケジューリング，横取り不可能 (nonpreemptive) スケジューリングという．プロセススケジューリングにおける代表的なスケジューリングにおける代表的なもののアルゴリズムを示す．

a) **到着順スケジューリング**(first come first service : FCFS)

システムに到着したプロセスから順番に処理する，横取りのないアルゴリズムである．プロセスは実行可能になると実行可能待ち行列の最後に置かれる．スケジューラは CPU を，実行可能待ち行列の先頭のプロセスから順に割り当てる．

b) **処理時間順スケジューリング**(shortest job first : SJF)

処理時間の短いプロセスから順に CPU に割り当てる，横取りのないアルゴリズムである．全プロセスの平均応答時間を最小にする．処理時間の長いプロセスほど後ろのプロセスの待ち時間を長くしてしまうからである．しかし，一般には，あらかじめそのプロセスの処理時間はわからないため，ユーザによる処理時間の指定，統計による処理時間の予測が行われる．

c) **優先度順スケジューリング**(priority)

各プロセスに優先度をつけ，優先度の高い順に実行を行う，横取りのないアルゴリズムである．優先度は，使用するメモリの大きさ，CPU 処理時間，入出力時間など，様々な基準によって定義される．優先度の低いプロセスがいつまでも CPU を獲得できないという飢餓状態 (starvation) が起こる．よってプロセスの優先度を待ち時間の長さに応じて除々に上げていく時効化 (aging) と

[4] 1 時間あたりのジョブ数最大化

いう方法を用いる．

d) **残余処理時間順スケジューリング**(shortest remaining processing time first：SRPT)

残余処理時間順に実行する，横取り有りのアルゴリズムである．新しいプロセスが実行可能になるたびにその時点で残っているプロセスの処理時間を比較して最も小さいものを優先的に実行する．処理時間順スケジューリングと同様，プロセスの処理時間を正確に予測するのが難しい．

e) **ラウンド ロビンスケジューリング**(round-robin)

横取りのあるスケジューリングの方針として最も広く用いられている．一定時間ごとにプログラムの実行を切り替えるアルゴリズムである．プログラムの実行時間が一定時間を過ぎたプロセスは実行を中断され，待ち行列の最後に回される．すなわち，待ち行列を巡回しながら実行されていくのである．