2017年度「知能情報処理演習2」プログラミング課題：確率的探索法
– 進化と遺伝のアルゴリズムを用いて，一番良い答えを探し出す方法 –

1　テーマ概要

生物の遺伝と進化のメカニズムを計算機上で模倣して，適応や学習，最適化などの機能を人工的に実現しようという手法は，一般に**進化型計算 (Evolutionary Computation, EC)** と呼ばれている．**遺伝的アルゴリズム (Genetic Algorithm, GA)** は，その中の代表的なものである．

本テーマでは，遺伝的アルゴリズムを用いて，探索を行うプログラムを作成する．以下では，第2章でまず生物の遺伝と進化について簡単にみた後に，それを模した遺伝的アルゴリズムについて説明する．第3章では，探索とは何かを考えた後に，遺伝的アルゴリズムを用いて探索を行う例を示す．第4章で，演習課題であるナップザック問題を説明する．7週間にわたる演習は，以下のように進める：

1回目： 第2, 3章を理解した上で，3.3節に示された問題例に対するサンプルプログラムを動作させる．進化過程とともにどのような個体が増えてゆくかを観察し，パラメータ値を変更すると，それがどのように変わるかをみる．また，第2章に挙げられている交叉，突然変異，選択などの遺伝的操作が，プログラム中でどのように実装されているかを確認する．

2-3回目： サンプルプログラムに対して，4.3.1節に示されたエリート保存戦略を組み込むよう拡張する．最適化問題は，サンプルプログラムが対象としている1-max問題の範囲でよい．完成した者から，中間レポートの作成を行い，次の課題に進む．

中間レポート： 4回目の授業冒頭に，エリート戦略を実装した結果を報告したレポートを提出する．指定書式によるA4版1ページのレポートを印刷して提出する．4-5回目の授業において，提出されたレポートに基づき，PC端末の前で個別に諮問する．その際，プログラムやデータなども随時確認する．

4-6回目： 4.1節に示されている基本ナップザック問題を解くために，サンプルプログラムを拡張する．対象とする問題データは，授業中に伝えるURLからファイルで取得する．

1. 第1段階では，4.2節に示されたペナルティー法による適応度関数を用いることで実現する。それを完成させたのち，第2段階では，4.3.3節に示された欲張り法へと進む．
2. まず，純粋な欲張り法で解を求めるプログラムを作成し，それによって解を求める．得られた解は，ペナルティー法での遺伝的アルゴリズムで得られたものと比較してどうであったか．
3. 次に，欲張り法をデコーティングに用いた遺伝的アルゴリズムを実現する．
4. 基本問題で上記が完成した者は，4.4節に挙げられている拡張問題に挑戦し，上記と同様の考察を行う．
5. また，交叉や選択方法などを各自で工夫してプログラムすることが望ましい．

完成したプログラムにおいて，パラメータ値の変更，交叉や選択方法の違いなどによって，得られる解がどのように変わるかを数値実験により調べる．

最終レポート： 7回目の授業冒頭に，以上の結果を報告したレポートを提出する．指定書式によるA4版2ページのレポートを印刷して提出する．

7回目： 提出した最終レポートに基づき，個別に諮問する．同時に各自で，授業時間内にレポートの書式，内容も含めた最後の修正を行い，授業終了時に電子ファイルにてmanaba+Rに提出する．

レポートの書式や提出要領は，別途指示する．

図 1: 遺伝子の列としての染色体

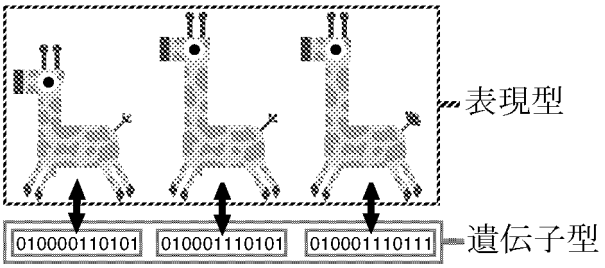

図 2: 遺伝子型とその表現型

2　遺伝的アルゴリズム

2.1　生物の進化と遺伝

地球上に出現して以来，生物は，環境に適応したものが生き延びて，親から子へ，子から孫へとその形質を伝えてきた．このうち，変動する環境中でもそれに適応したものが生き延びて子孫をつくるメカニズムは，**進化**として捉えられる．進化論の中でも C. Darwin が提唱した**自然選択 (natural selection) 説**はよく知られている．自然選択説の考え方は，生物個体の間での交叉や，個体の突然変異によって，それまでとは異なる形質が現れると，新しい個体は，そのときおかれた環境への適応の度合いに応じて，生き延びて子孫を増やしたり，逆に適応できずに死滅したりするというものである．また，親から子に形質が伝えられるメカニズムは，**遺伝**として研究されてきた．特に 20 世紀後半における分子生物学のめざましい発展により，遺伝と進化の機構が統合的に，分子レベルでも解明され，以下のようなことが知られるようになった．

生物個体には，その全ての形質を決定する設計図があり，**遺伝子 (gene)** として細胞の核の中に存在する．遺伝子は，**染色体 (chromosome)** と呼ばれる構造物の中に規則的に配置されている．より詳細については省き，ここでは次のように模式化した理解にとどめ，次節以降での遺伝的アルゴリズムで用いることにする．

染色体上では，1 つの遺伝子はそれぞれ 1 つの**遺伝子座**とよばれる位置に格納されている．多数の遺伝子座が，鎖のように 1 次元的に並んで 1 つの染色体となっている．また，それぞれの遺伝子座に格納することができる遺伝子は複数種類あり，それらを**対立遺伝子**とよぶ．この遺伝子の並び方こそが，その染色体が担う遺伝情報となっている．図 1 にその模式図を示す．

ある生物個体がもつ遺伝子の構成と配列を，その個体の**遺伝子型 (genotype)** といい，その遺伝情報に基づいて，おかれた環境の中で発現する形質を，その個体の**表現型 (phenotype)** という（図 2）．

遺伝子列に担われた遺伝情報は，個体が子個体を作って増殖する際にコピーされる．ただその際，僅かな確率でミスコピーが起こり，コピー元とは異なる遺伝子列が生成される．このミスコピーを，**突然変異 (mutation)** とよぶ．また，有性生殖をおこなう多くの生物では，2 体の親個体から子個体が生成されるが，その際に各親個体に由来する 2 つの遺伝子列が，互いの対応する位置でいずれも切断され，その前後が組み替えられて新たな遺伝子列が作られ，それが子個体の遺伝子列となる．この過程を**交叉 (crossover)** とよぶ．以上のように染色体，すなわち，遺伝子列は，親から子へと引き継がれる際に，交叉や突然変異によって新たな遺伝子列（遺伝子型）となることがあり，その結果として新たな形質（表現型）をもった個体が生まれることになる．自然選択説では，生み出された表現型が，その個体の置かれた環境に適応しているほどより高い割合で生存し，次に子個体を残してゆくことで，より高い割合で増殖してゆくことになる．

このように何種類かの遺伝子がいくつかの遺伝子座に格納された染色体は，まるで何種類かの文字からなる文字列であり，生物個体の遺伝情報は記号列によって表されていると見なすことができる．そこで，

上述のように簡略化したモデルを，計算機プログラムによって実現することを考える．以下に，それに適していると考えられる，遺伝と進化メカニズムの特徴を挙げて，本節のまとめとする．

- 各個体の遺伝情報は，遺伝子による記号列として表現されている．
- 個体の増殖に際しては，記号列がコピーされる．
- その際，子個体の記号列は親個体の記号列を単にコピーして生成されるだけでなく，二親間での交叉によって記号列を部分的に交換し，また，突然変異によって記号列をミスコピーすることで，親個体にはなかった新しい記号列（遺伝子型）や，その結果としての新しい形質（表現型）が生成される．
- 各世代の個体群においては，それぞれの個体がその環境への適応の度合いに応じて生存し，子個体を残すことで，増殖する割合が決まる．（個体に対する自然選択）

2.2 遺伝的アルゴリズムの基本的な考え方

遺伝的アルゴリズムは，1960 年代に米国の J. Holland が提案した適応システムを源流とする．その後，L. Fogel や De Jong らの関連研究を経て，1989 年に D. Goldberg がアルゴリズムの枠組みを整理して著作としてもまとめ，より広く用いられるきっかけとなったため，以下ではその枠組みに沿って説明する．

1 個体とその遺伝子型 ある個体の表現型を $\mathbf{x}$, 遺伝子型を以下のような N 個の記号 s_j $(j = 1, \ldots, N)$ の列 $\mathbf{s}$ で表すことにする．

$$\mathbf{s} = s_1\ s_2\ s_3\ \cdots\ s_j\ \cdots\ s_N \tag{1}$$

つまり $\mathbf{s}$ は，N 個の遺伝子座からなる 1 つの染色体に対応する．表現型 $\mathbf{x}$ の具体的な形や意味については，次の 3 章の後半であらためて説明することにして，ここでは触れない．s_j は j 番目の遺伝子座に格納されている遺伝子を表しており，対立遺伝子が n 種類あるとき，s_j は n 個の数値のいずれか 1 つををとる．例えば，最小の $n = 2$ に対しては，s_j は $\{0, 1\}$ の 2 値をとる binary 変数だと考えればよい．

M 個体からなる個体集合 次に，式 (1) のような長さ N の記号列で表される個体が，全部で M 個体集まった集合を考える．ある生物種の，ある世代における全個体の集まりを模したものであり，第 t 世代における個体集合を $P(t)$ と書くことにする．個体集合 $P(t)$ が親世代となって，各個体の遺伝子列 $\mathbf{s}$ のコピーや交叉，突然変異を経て，子世代の個体集合 $P(t+1)$ を生成する．ここで，どの世代においても，個体集合に含まれる個体の数は常に一定の M としておく．

世代を更新するための遺伝的操作 親世代の M 個体から，子世代の M 個体を生成するために，以下のような 3 つの操作をほどこす．いずれも，2.1 節でみた，生物の増殖における交叉や突然変異，そして，自然選択を模したものであり，これらを合わせて**遺伝演算子**とよぶ．

交叉 (crossover)：個体集合の中から，2 個の個体をいずれもランダムに選びだし，ある確率（**交叉確率**とよぶ) で，2 個体それぞれの遺伝子列 $\mathbf{s}_1, \mathbf{s}_2$（太文字であることに注意）の一部を交換する．具体的な交換方法については，後述する．

突然変異 (mutation)：各個体について，ある確率（**突然変異確率**とよぶ）で，各遺伝子座 j $(j = 1, \ldots, N)$ の遺伝子の値 s_j を，他の対立遺伝子の値と入れ替える．

選択 (selection)：個体集合に含まれる各個体について，その表現型 $\mathbf{x}$ が環境に適応している度合いに応じて，コピーする子個体の数を決定する．全個体数は M 個体と一定になるようにする

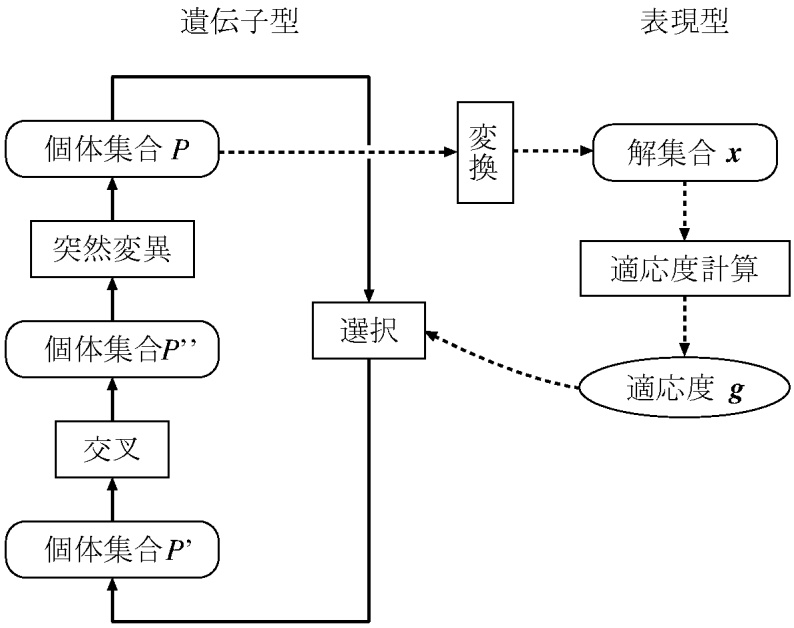

図 3: 遺伝的アルゴリズムにおける 1 世代の手順

ために，子個体を 2 つ以上残す個体（環境によく適応したもの）もある一方，1 つも残せない個体（環境にあまり適応しないもの）もあることになる．以下では，適応の度合いを数値的に表したものを**適応度**とよび，その値を g とする．

以上の遺伝的アルゴリズムの手順は，次の 2.3 節に挙げる各ステップで構成される．

2.3 遺伝的アルゴリズムの基本的な構成

ステップ 1：初期化 ランダムに M 個の個体の遺伝子型（N 文字からなる文字列）を生成して，初期世代における個体集合 $P(0)$ を作る．世代を表すループ変数 t は，$t = 0$ と設定する．

ステップ 2-1：評価 個体集合 $P(t)$ に含まれる全 M 個体に対して，その表現型 $\mathbf{x}_i$ に応じて決まる適応度 g_i $(i = 1, \ldots, M)$ を求める．

ステップ 2-2：選択 個体集合 $P(t)$ に対して，各個体の適応度 g_i に基づき，前節 2.2 に示した選択操作をほどこし，その結果得られた新たな M 個体からなる集合 $P'(t)$ を生成する．

ステップ 3：交叉 個体集合 $P(t)'$ に対して，前節 2.2 に示した交叉操作をほどこし，その結果得られた新たな M 個体からなる集合 $P''(t)$ を生成する．

ステップ 4：突然変異 個体集合 $P(t)''$ に対して，前節 2.2 に示した突然変異操作をほどこし，その結果得られた新たな M 個体からなる集合を生成し，それをもって次世代の個体集合 $P(t+1)$ とする．t を $t+1$ に 1 増加する．

ステップ 5：終了判定 進化過程の最終世代数 T をあらかじめ設定しておき，$t < T$ ならば，上記のステップ 2-1 に戻る．$t \geq T$ となった時点で，計算のループを終了させる．終了時点までに得られた最大適応度をもつ個体をもって，一番良い個体とする．

図 3 に，全 T 世代の繰り返しを行う上記の計算手順を示す．

2.4 単純 GA (Simple GA) における構成

前節のアルゴリズムのより具体的な構成法として，D. Goldberg が示した**単純 GA(Simple GA)** の構成を示す．

1. 個体の遺伝子型の与え方： 2 値 $\{0,1\}$ からなる長さ N の記号列を用いて，遺伝子型（染色体）を表す．例えば，$N=7$ のときには，

$$1\ 0\ 1\ 1\ 0\ 0\ 1$$

で 1 個体を表す．

2. 初期世代の個体集合の生成： 各個体の遺伝子列を乱数を用いてランダムに決定する．つまり，各遺伝子座の遺伝子の値として，$\{0,1\}$ のいずれかを等確率 0.5 で独立に与える．このようにして，全 M 個体を生成し，初期世代における個体集合とする．

3. 適応度の与え方： 各個体は，その適応度 g に比例した数の子個体をコピーして生成するものとする（次項でさらに説明する）．そのために，適応度は非負（正か 0）である必要がある．

4. 3 つの遺伝演算子の実現方法：

選択方法：ルーレット選択 全 M 個体の適応度 g_i $(i=1,\ldots,M)$ を計算し，その総和 $G=\sum_i^M g_i$ を求める．子個体を生成するためにコピーする親個体を個体集合 $P(t)$ の中から 1 つ選ぶに当たっては，i 番目の個体は確率 g_i/G で選ばれるように確率的に決める．それを M 回，独立に繰り返し，全 M 個体からなる集合 $P'(t)$ を生成する．この選択方法は，適応度に比例した確率に従って，コピーできる子個体数が決まることから，**適応度比例選択**あるいは**ルーレット選択**と呼ばれる．

交叉方法：1 点交叉 個体集合 $P'(t)$ の中からランダムに 2 個体ずつペアとして，全 $M/2$ ペア作る．各ペアに対して，交叉確率 p_c で交叉を実行する（逆に確率 $1-p_c$ で，そのペアに対する交叉は行わない）．交叉を行う場合には，長さ N の記号列中にある $N-1$ 箇所の区切りから等確率で 1 箇所を選び，交叉点とする．すなわち，交叉点の前後でペア間の記号列を交換する．
以下に，先頭から 3 文字目の直後を交叉点に選んだ場合の例を示す．

$$\underline{1}\ \underline{0}\ \underline{1}\,|\,\underline{1}\ \underline{0}\ \underline{0}\ \underline{1} \qquad\qquad \underline{1}\ \underline{0}\ \underline{1}\,|\,0\ 1\ 1\ 1$$
$$\longrightarrow$$
$$0\ 0\ 0\,|\,0\ 1\ 1\ 1 \qquad\qquad 0\ 0\ 0\,|\,\underline{1}\ \underline{0}\ \underline{0}\ \underline{1}$$

突然変異方法：ビット反転 個体集合 $P''(t)$ 中の全 M 個体に対して，各遺伝子座の遺伝子の値を，突然変異確率 p_m でビット反転させる．
以下に，先頭から 4 文字目を反転させた場合の例を示す．

$$1\ 0\ 1\ \underline{1}\ 0\ 0\ 1 \longrightarrow 1\ 0\ 1\ \underline{0}\ 0\ 0\ 1$$

5. 4 つのパラメータ値の設定： 以下の 4 つのパラメータ値を設定する．

個体集合に含まれる個体数（個体集合サイズ）： M

最終世代数： T

交叉確率： p_c

突然変異確率： p_m

なお，文字列長 N は，遺伝子型の与え方（コーディング方法）に含まれるものであり，繰り返し計算の実行に際して設定すべき上記 4 つのパラメータとは区別しておく．

3　最適な解を探索する方法

3.1　最適化問題

最適化 (optimization) とは，いくつもの答えがある中から一番優れたものを選び出す課題であり，日常的な多くの問題がこれに含まれる．

今回の実験課題は，以下のような問題である．

> 海外に住んでいる友だちのところに旅行にゆくために，荷造りの準備をしている．せっかく大きなスーツケースを用意したにも関わらず，飛行機に乗る際の預け入れ荷物には重量制限があり，20kg を超えることはできないことを知った．そこで，持って行きたいと思っていた友だちへのお土産や，自分の服や日用品，日本でしか買えない好きなおやつなど，準備した全ての品物をスーツケースに詰めることはできないこと分かり，先ほどから品物とスーツケースと体重計を前にして頭を悩ましている．重量制限を守った上で，できるだけ自分が持ってゆきたいものを詰め込むには，どうすれば良いだろう．
> まず，それぞれの品物を，どの程度持ってゆきたいか，という自分なりの評価基準を作ってみた．そして，その基準に従って評価値が最大になるように，20kg 以内という規則は満たした上で荷物を詰めることにしたい．
> しかしその後，どう詰めるべきか，やはり困ってしまった．1 つ目の方法としてやってみたのは，後悔しないように，全ての詰め込み方（どの品物は入れて，どの品物は入れないか）を考えてみようとしたのだが，準備していた品物の数がとても多く，全部の組合せを挙げてみるのは到底無理そうだ．そこで 2 つ目の方法として，評価の高いものから詰めたり，軽いものから詰めたりすればどうか，と思ったが，品物の重さと評価はまちまちで，最後まで詰めてゆくとどうもうまくゆかない気がする．その上，1 つの品物を諦めると代わりに別の 2 つの品物を詰めることができたりするので，そんなことを考えながら出し入れしていると，きりがなくなってしまいそうだ．

... こういった厄介な事態にすぐ突き当たるのが，実は，殆んどの最適化問題の特徴である．最初に考えた方法は，**列挙法**あるいは**全解探索法**などとよばれる方法である．もしも全ての組合せを列挙することができたならば，その中から必ず最良の答えを見つけることができるのは明らかなので**厳密解法**なのだが，品物が多数になるほど，組合せの数は**指数関数的爆発**を起こし，現実的には実行できないことが多い．

その次に考えた，何かうまいルールを考えてそれに従って詰める方法は，**発見的手法**あるいは**ヒューリスティック探索法**などとよばれる．運良く，良さそうなルールを思い付ければ幸運だが，常にそういうものを思いつ付けるとは限らないし，自分だけではなく，今まで誰も思い付けていないかも知れない．第一，それで最良の答えが見つかる保証は全くないにも関わらず，ルールを 1 つ決めてしまうと答えは 1 つしか出て来ないので，本当にそれで満足して良いのか，判断に迷うところだ．

そこで，両者の中間とでもいうべき方法として，**探索的手法**とも言われるものが色々考えられてきた．つまり，1 つ目の列挙法のように全部をしらみつぶしに探すのは到底無理だが，2 つ目の発見的手法のように 1 点決めをするよりは，色々と比較検討をしながらより良いものを探し出したい．ただしそのためには，良さそうな答えの候補をいくつも見つけ出してくる手だてが必要となるが．

実は，この探索的手法の代表的なものが，前章でみた遺伝的アルゴリズムなのである．ここでは，実際に遺伝的アルゴリズムを用いて，求めたい答えを探索してみることで，それを理解してゆくことにする．

なお，上記に挙げた課題は，**ナップザック問題 (Knapsack Problem, KP)** とよばれ，代表的な最適化問題として知られている．3.2 節で簡単に，最適化問題の例と，一般的な定式化を紹介しておく．

3.2 最適化問題の例

以下に，ナップザック問題の再掲も含めて，2 つの代表的な最適化問題を挙げる．

ナップザック問題：N 個の品物と，一定重量まで品物を入れることができるナップザックがある．各品物の重量および価値は分かっている．このとき，ナップザックの重量制限を超えない範囲で，価値の和が最大となるように入れる品物を決定せよ．

巡回セールスマン問題 (Traveling Salesman Problem, TSP)：N 個の都市があり，あるセールスマンが各都市を一度ずつ訪問しなければならない．各都市の位置（あるいは，各都市間の距離）は分かっている．このとき，巡回経路の総距離が最小になるような巡回路を求めよ．

また以下に，最適化問題の数理的な定式化を示す．対象とする問題において，決定すべき量（KP では各品物を入れるか否か，TSP ではどの順に各都市を巡るか）を**決定変数** とよぶ．本節では，決定変数を $\mathbf{x}$ で表すことにすると，最適化問題は一般に，次のように書くことができる．

最適化問題の定式化

$$\min_{\mathbf{x}} f(\mathbf{x}) \tag{2}$$

$$\text{ただし，}\quad \mathbf{x} \in F \tag{3}$$

$$F \subseteq X$$

ここで，$f(\mathbf{x})$ は**目的関数**とよばれ，$\mathbf{x}$ という答えの良さ（あるいは逆に，悪さ．両者は，符合が逆であるだけの違いである）を数量的に表す実数値をとる．KP では価値の和，TSP では巡回経路の総距離が，目的関数に相当する．$\mathbf{x}$ は一般に多次元の変数（ベクトル量）で，X という空間の要素だが，実際には空間 X 内の任意の点をとり得る訳ではなく，空間 X に含まれる部分空間である F 内に制限される．式 (3) はそれを表しており，最適化問題における**制約条件**とよばれる．KP においては重量制限が，TSP においては必ず一度ずつ訪問することが，制約条件となっている．制約条件を満たした上で，目的関数を最小にする $\mathbf{x}$ を探し出すことが最適化問題であり，式 (2) がそれを表現している．

3.3 最適化問題を遺伝的アルゴリズムで解く方法：ごく簡単な問題を例に

本節では，3.1, 3.2 節でみてきた最適化問題のうち，ごく簡単な下記の問題を例にとって，2 章で説明した遺伝的アルゴリズムによる探索的手法で解くことにする．そのためのプログラム例を，付録 A に示す．

問題：15 ビットの記号列に対して，全ビットの和が最大になるものを求めよ．

定式化 を行うと，

$$\begin{aligned} \max_{x_j \in \{0,1\}, j=1,\ldots,15} f(\mathbf{x}) &= \sum_{j=1}^{15} x_j \\ \text{ただし，}\quad \mathbf{x} &= (x_1, x_2, \cdots, x_{15}) \in \{0,1\}^{15} \end{aligned}$$

ナップザック問題 の一種とみることができ，品物数 $N = 15$ で，ナップザックの重量制限が無限大の場合に相当している．よって，15 個の品物全てを入れる $(x_j = 1, j = 1, \ldots, 15)$ のが最適解であることは，すぐに分かる．あるいは，一つしかない山の頂上を見つけることにも相当しているため，**1-Max 問題**ともよぶ．

ここでは，2.4 節で説明した単純 GA を用いて最適解を探索することにする．すなわち，**各個体を，この最適化問題の一つの解 $\mathbf{x}$ に対応させる**．このとき，遺伝子型の与え方や表現型との関係，適応度の与え方や目的関数との関係，遺伝演算子である選択，交叉，突然変異の方法は，それぞれ以下のようにとることにする．

1. **遺伝子型 $\mathbf{s}$ と表現型 $\mathbf{x}$** 表現型は，この最適化問題の解 $\mathbf{x}$ であり，$\mathbf{x}= (x_1, x_2, \cdots, x_{15}) \in \{0,1\}^{15}$ ．よって，遺伝子型 $\mathbf{s}$ も $\mathbf{x}$ と一致させればよく，$s_j = x_j$ $(j = 1, \ldots, 15)$ で与えられる，長さ $N = 15$ のビット列とする．

2. **適応度 g と目的関数 f** 目的関数 $f(\mathbf{x})$ が非負であり，これを最大にする $\mathbf{x}$ を探索する問題（最大化問題）であるので，目的関数 $f(\mathbf{x})$ を，遺伝的アルゴリズムにおけるその個体 $\mathbf{x}$ の適応度 $g(\mathbf{x})$ と一致させればよい．

なお一般に，表現型 $\mathbf{x}$ を遺伝子型 $\mathbf{s}$ に対応づけることをコーディング (coding)，逆に，遺伝子型 $\mathbf{s}$ を表現型 $\mathbf{x}$ に対応づけることをデコーディング (decoding) とよんでいる．最適化問題を遺伝的アルゴリズムで解く場合には，表現型 $\mathbf{x}$ は最適化問題の解に対応しており，その良さの尺度である適応度 g は，最適化問題における解の良さを数値的に表す目的関数 f と対応づけられる必要がある．本例では，g を f と全く同一にとることができたが，一般には適切に工夫して設計する必要がある．

遺伝子型 $\mathbf{s}$ は，$\mathbf{x}$ を遺伝子の記号列で表したものであり，これに対して，以下の 3 つの遺伝的操作がほどこされることになる．

3. **選択** 上記の適応度 g に基づく，ルーレット選択を用いる．

4. **交叉** 一点交叉とする．すなわち，15 ビットの列の中にある 14 箇所の区切りから 1 箇所をランダムに選んで交叉点とする．

5. **突然変異** ビット反転とする．すなわち，選ばれた遺伝子 x_j の値に応じて，$x_j = 0$ ならば $x_j = 1$ に，逆に $x_j = 1$ ならば $x_j = 0$ に変える．

6. **パラメータ値** 4 つのパラメータ M, T, p_c, p_m の値を適切に設定する．

付録 A に掲載されている C 言語のプログラム例は，ソースファイルを所定の URL（授業中に伝える）から取得し，各自で実行して動作を観察してみよう．プログラム例では，上記の 4 つのパラメータを冒頭に `define` 文で定義しており，$M = 15$, $T = 100$, $p_c = 0.3$, $p_m = 0.10$ としている．この値を各自で書き替えて，動作がどのように変わるか，観察してみよう．

また，付録 A のプログラムでは構造体を用いてある世代の個体集合を表している．i 番目の個体の j 番目の遺伝子座という意味を表すデータ構造として，構造体を用いていることを確認しよう．

4 最適化問題の課題：ナップザック問題

本実験課題では，3.1, 3.2 節で紹介したナップザック問題に対して，遺伝的アルゴリズムを用いて解くことを考える．以下では，再びナップザック問題をその定式化とともに挙げた上で，遺伝的アルゴリズムを適用するために，どのように具体的にアルゴリズムを構成するかを示す．

4.1 問題の復習

基本となるナップザック問題 N 個の荷物と，一定重量まで荷物を詰めることができるナップザックがある．j 番目の荷物 $(j = 1, \ldots, N)$ の重量を $a_j(> 0)$，価値を $c_j(> 0)$ とし，ナップザックの制限重量を $b(> 0)$ とする．ナップザックの制限重量内で，価値の和が最大になるよう，詰める荷物を決定せよ．

その定式化

$$\begin{aligned} \max_{x_j=\{0,1\},j=1,\ldots,N} f(\mathbf{x}) &= \sum_{j=1}^{N} c_j \cdot x_j \\ \text{ただし,} \quad & \sum_{j=1}^{N} a_j \cdot x_j \leq b \\ \mathbf{x} &= (x_1, x_2, \cdots, x_N) \in \{0,1\}^N \end{aligned} \tag{4}$$

ここで，決定変数 x_j $(j = 1, \ldots, N)$ は，j 番目の荷物を入れることを $x_j = 1$ で，逆に入れないことを $x_j = 0$ で表すものとする．

4.2 遺伝的アルゴリズムの基本的な構成方法

1. 遺伝子型 s と表現型 x 表現型は，この最適化問題の解 $\mathbf{x}$ であり，$\mathbf{x} = (x_1, x_2, \cdots, x_N) \in \{0,1\}^N$ ．よって，遺伝子型 $\mathbf{s}$ も $\mathbf{x}$ と一致させればよく，$s_j = x_j$ $(j = 1, \ldots, N)$ で与えられる，長さ N のビット列とする．

2. 適応度 g と目的関数 f 目的関数 $f(\mathbf{x})$ が非負であり，これを最大にする $\mathbf{x}$ を探索する問題（最大化問題）であるので，目的関数 $f(\mathbf{x})$ を，遺伝的アルゴリズムにおけるその個体 $\mathbf{x}$ の適応度 $g(\mathbf{x})$ と一致させればよい．

しかしながら，3.3 節の 1-Max 問題と異なる点は，ナップザック問題には制約条件 (4) が課せられていることであり，この制約に違反する詰め方は，いかに価値の和が多くても解としては適していない．そこで，制約条件を考慮するために一般に用いられる便利な方法が，以下のような**ペナルティ法**というものである．つまり，制約条件に違反している程度を違反量 $\sum_{j=1}^{N} a_j \cdot x_j - b$ として数量的に表した上で，違反に対するペナルティ（penalty, 罰）として，適応度から違反量に応じた値を差し引くことにする．

$$g(\mathbf{x}) = \sum_{j=1}^{N} c_j \cdot x_j - \alpha \cdot \max\left\{0, \sum_{j=1}^{N} a_j \cdot x_j - b\right\} \tag{5}$$

ここで，$\max\{0, \sum_{j=1}^{N} a_j \cdot x_j - b\}$ は，制約に違反した場合 ($\sum_{j=1}^{N} a_j \cdot x_j - b > 0$) にはその違反量を表し，違反していない場合 ($\sum_{j=1}^{N} a_j \cdot x_j - b \leq 0$) には 0 となるように，max 関数が利用されている．α は，違反量に応じてどの程度適応度を減じるかを決める係数であり，ペナルティ法を用いる場合に導入されるパラメータである．

ペナルティ法を用いる場合，ペナルティに相当する式 (5) の第 2 項（ペナルティ項ともよぶ）が大きいと，g 値が負になることもありうる．そこで，違反量が大きく $g < 0$ となった個体に対しては $g = 0$ として，g が非負となるように工夫をするとよい．つまり，$g' = \max\{0, g\}$ として非負となるように修正した g' 値を用いて選択を行う．

3. 遺伝演算子 選択，交叉，突然変異は，単純 GA のものをそのまま用いる．すなわち，ルーレット選択，一点交叉，ビット反転とする．

4. パラメータ値 4 つのパラメータ M, T, p_c, p_m，および，上述のペナルティ係数 α の値を適切に設定する．

4.3 アルゴリズムにおけるいくつかの拡張

4.3.1 エリート保存戦略

適応度の高い優れた個体は，高い割合でコピーを残すことができるが，遺伝演算子はいずれも確率的な演算であるために，優れた個体がコピーを残せなかったり，交叉や突然変異によって変えられてしまう可能性がある．そこで，各世代の個体集合の中で，最大の適応度をもつ個体（**エリート個体**とよぶ）に対してのみ，無条件に優遇することにする．つまり，交叉や突然変異がほどこされていないという意味でのエリート個体の完全なコピーが，次世代の個体集合に必ず 1 個体は含まれるように操作する．

4.3.2 交叉方法の拡張

単純 GA では一点交叉を用いるが，それ以外に多点交叉や一様交叉があり，ナップザック問題では，それらの方が良い結果を与えることもある．

多点交叉 一点交叉では，長さ N の記号列の中にある $N-1$ 箇所の区切りから，等確率で 1 箇所を選んで交叉点としたが，同様に等確率で 2 箇所あるいは 3 箇所以上を選んで交叉点とし，いずれの交叉点の前後でも 2 個体の親個体間で記号列を交換する．

以下に，先頭から 3 文字目の直後と 5 文字目の直後を交叉点に選んだ場合の，二点交叉の例を示す．

$$\underline{1}\ \underline{0}\ \underline{1}\,|\,\underline{1}\ \underline{0}\,|\,\underline{0}\ \underline{1} \qquad\qquad \underline{1}\ \underline{0}\ \underline{1}\,|\,0\ 1\,|\,\underline{0}\ \underline{1}$$

$$\longrightarrow$$

$$0\ 0\ 0\,|\,0\ 1\,|\,1\ 1 \qquad\qquad 0\ 0\ 0\,|\,\underline{1}\ \underline{0}\,|\,1\ 1$$

一様交叉 いずれの親個体の遺伝子を受け継ぐかを，（上記のように交叉点の前後で変えるのではなく）遺伝子座ごとに，等確率で独立に選ぶことにする．

以下に，1 行目のように継承元の親を決めた場合の例を示す．

$$\circ\ \bullet\ \bullet\ \circ\ \bullet\ \circ\ \circ$$

$$\underline{1}\ \underline{0}\ \underline{1}\ \underline{1}\ \underline{0}\ \underline{0}\ \underline{1} \qquad\qquad 0\ \underline{0}\ \underline{1}\ 0\ \underline{0}\ 1\ 1$$

$$\longrightarrow$$

$$0\ 0\ 0\ 0\ 1\ 1\ 1 \qquad\qquad \underline{1}\ 0\ 0\ \underline{1}\ 1\ \underline{0}\ \underline{1}$$

4.3.3 欲張り法の導入（制約条件を満たすためのペナルティ法以外の方法）

発見的手法（ヒューリスティックス）については，3.1 節で紹介したが，ナップザック問題に対してよく知られている発見的手法に，**欲張り法 (greedy algorithm)** がある．これは，荷物ごとに重量当りの価値 c_j/a_j を求め，その値の大きい荷物から順に，ナップザックの制限重量を超過しない限り，詰めてゆくという方法である．単純なアルゴリズムにも関わらず，かなり良好な解を与えることが多い．（欲張り法で解を求めるプログラムを作成し，実際に解を求めてみよう．その解を，遺伝的アルゴリズムを用いて得られた解と比較してみよう．）

さらにここでは，欲張り法の考え方を組み込んで，ナップザック問題の制約条件 (4) を満たすために，ペナルティ法を用いない以下の方法を説明する．

制限重量を超過するのは，遺伝子型 $\mathbf{s}$ において $s_j = 1$ となったとき常に $x_j = 1$ として，j 番目の荷物を詰めるからである．そこで，欲張り法の考え方にならって，まず $s_j = 1$ となった荷物 j をナップザックに詰める候補とする．候補となった荷物に対してのみ重量当りの価値 c_j/a_j を求め，その値の大きい候補

荷物から順に，ナップザックの制限重量を超過しない限り，詰めてゆく．こうすることで，制約条件 (4) は常に満たされる．よってまた，ペナルティ項を付加した適応度 g を用いる必要がなく，以下のように目的関数 f をそのまま非負の適応度 g として用いることができる．

$$g(\mathbf{x}) = \sum_{j=1}^{N} c_j \cdot x_j$$

4.4 ナップザック問題におけるいくつかの拡張

4.1 節に示した基本問題に対して，以下のように様々に拡張した問題を考えることができる．

4.4.1 複数の制約条件がある場合

拡張したナップザック問題 1 N 個の荷物と，**一定重量かつ一定容積まで**荷物を詰めることができるナップザックがある．j 番目の荷物 ($j = 1, \ldots, N$) の重量を $a_j(> 0)$，容積を $d_j(> 0)$，価値を $c_j(> 0)$ とし，ナップザックの制限重量を $b(> 0)$，制限容積を $e(> 0)$ とする．ナップザックの制限重量内，かつ，制限容積内で，価値の和が最大になるよう詰める荷物を決定せよ．

その定式化

$$\begin{aligned}
\max_{x_j=\{0,1\}, j=1,\ldots,N} f(\mathbf{x}) &= \sum_{j=1}^{N} c_j \cdot x_j \\
\text{ただし，} \quad & \sum_{j=1}^{N} a_j \cdot x_j \leq b \\
& \sum_{j=1}^{N} d_j \cdot x_j \leq e \\
\mathbf{x} &= (x_1, x_2, \cdots, x_N) \in \{0,1\}^N
\end{aligned}$$

4.4.2 複数のナップザックがある場合

拡張したナップザック問題 2 N 個の荷物と，一定重量まで荷物を詰めることができる **2 つのナップザック**がある．j 番目の荷物 ($j = 1, \ldots, N$) の重量を $a_j(> 0)$，価値を $c_j(> 0)$ とし，ナップザック 1,2 の制限重量をそれぞれ $b_1, b_2(> 0)$ とする．いずれのナップザックも制限重量を満たした上で，2 つのナップザックに詰めた荷物の価値の総和が最大になるよう，それぞれのナップザックに詰める荷物を決定せよ．

その定式化

$$\begin{aligned}
\max_{x_j=\{0,1,2\}, j=1,\ldots,N} f(\mathbf{x}) &= \sum_{j, x_j=1,2} c_j \\
\text{ただし，} \quad & \sum_{j, x_j=1} a_j \leq b_1 \\
& \sum_{j, x_j=2} a_j \leq b_2 \\
\mathbf{x} &= (x_1, x_2, \cdots, x_N) \in \{0,1,2\}^N
\end{aligned}$$

ここで，決定変数 x_j $(j = 1, \ldots, N)$ は，j 番目の荷物をナップザック 1 に入れることを $x_j = 1$ で，ナップザック 2 に入れることを $x_j = 2$ で，そして，いずれのナップザックにも入れないことを $x_j = 0$ で表す，3 値の変数 (trinary) とする．

4.4.3 複数の価値があり，それぞれの和をできるだけ大きくしたい場合

拡張したナップザック問題 3 N 個の荷物と，一定重量まで荷物を詰めることができるナップザックがある．j 番目の荷物 $(j = 1, \ldots, N)$ の重量を $a_j(> 0)$，**1 つ目の価値**（例えば，日本での売価）を $c_j(> 0)$，**2 つ目の価値**（例えば，米国での売価）を $d_j(> 0)$ とし，ナップザックの制限重量を $b(> 0)$ とする．ナップザックの制限重量内で，2 つの価値の和が**いずれも**できるだけ大きくなるよう，詰める荷物を決定せよ．（どのように，適応度を決めればよいだろうか？）

その定式化

$$
\begin{aligned}
\max_{x_j=\{0,1\}, j=1,\ldots,N} f_1(\mathbf{x}) &= \sum_{j=1}^{N} c_j \cdot x_j \\
\max_{x_j=\{0,1\}, j=1,\ldots,N} f_2(\mathbf{x}) &= \sum_{j=1}^{N} d_j \cdot x_j \\
\text{ただし，} \quad & \sum_{j=1}^{N} a_j \cdot x_j \leq b \\
\mathbf{x} &= (x_1, x_2, \cdots, x_N) \in \{0,1\}^N
\end{aligned}
$$

4.4.4 各荷物が複数個ある場合

拡張したナップザック問題 4 N 種類の**荷物がそれぞれ 2 個ずつ**と，一定重量まで荷物を詰めることができるナップザックがある．j 種類目の荷物 $(j = 1, \ldots, N)$ の重量を $a_j(> 0)$，価値を $c_j(> 0)$ とし，ナップザックの制限重量を $b(> 0)$ とする．ナップザックの制限重量内で，価値の和が最大になるよう，詰める荷物と**その個数**を決定せよ．

その定式化

$$
\begin{aligned}
\max_{x_j=\{0,1,2\}, j=1,\ldots,N} f(\mathbf{x}) &= \sum_{j=1}^{N} c_j \cdot x_j \\
\text{ただし，} \quad & \sum_{j=1}^{N} a_j \cdot x_j \leq b \\
\mathbf{x} &= (x_1, x_2, \cdots, x_N) \in \{0,1,2\}^N
\end{aligned}
$$

ここで，決定変数 x_j $(j = 1, \ldots, N)$ は，j 種類目の荷物をナップザックに 1 つ入れることを $x_j = 1$ で，2 つ入れることを $x_j = 2$ で，そしてナップザックに入れないことを $x_j = 0$ で表す，3 値の変数 (trinary) とする．

5　レポート評価

評価基準： 以下の項目を総合的に判断して，評価する．

基本項目：

- 正しく動作するプログラムであること
- プログラムでは，適切なコメント，適切な変数名，適切なインデンションなどが実現されていること
- 計算機実験の結果として探索の過程を示し，それに対する考察を加えていること
- パラメータ値の設定に応じて動作や探索性能がどのように影響されるか，考察を加えていること

発展項目：

- アルゴリズムに工夫があること
- プログラムの実装に工夫があること
- 拡張した問題に取り組んでいること
- その他，新規性のある工夫や変更を試みていること

参考文献

[1] 北野編,「遺伝的アルゴリズム 1」，産業図書，1993

[2] 三宮，喜多，玉置，岩本著,「遺伝アルゴリズムと最適化」，システム制御情報ライブラリー，1998

[3] 柳浦，茨木著,「組合せ最適化–メタ戦略を中心として–」，朝倉書店，2001

[4] 伊庭著,「探索のアルゴリズムと技法」，サイエンス社，2002

A　プログラム例

以下に，3.3 節で示した，一つしかない山の頂上を見つける「ごく簡単な問題」を遺伝的アルゴリズムで解くために作成したプログラム例を示す．ここでは，2.4 節で説明した単純 GA を用いている．つまり，遺伝子型はビット列で与えられ，選択方法はルーレット選択，交叉方法は一点交叉で，突然変異方法はビット反転である．

```
# 1-max-sga.c

#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

#define N 15   // 遺伝子長
#define M 15   // 個体数
#define T 100   // 世代数
#define Pc 0.30  // 交叉確率
#define Pm 0.10   // 突然変異確率

// 遺伝子型の定義
struct genotype {
  int gene[N];   // 遺伝子
  float fitness;   // 適応度
};

float evaluation(int *a);
void one_point_crossover(struct genotype *ind);
void mutation(struct genotype *ind);
void roulette_selection(struct genotype *ind);
int flip(float prob);
void print_process(struct genotype *ind, int generation);


int main(int argc, char *argv[]) {
  int i;   // 個体インデックス
  int j;   // 遺伝子座インデックス
  int t;   // 世代インデックス
  struct genotype individual[M];      // 個体

  // 乱数 seed の設定
  if(argc < 2) {   // プログラムの引数が足りない場合
    printf("Usage: %s [SEED_NUMBER]\n", argv[0]);
    exit(1);
  }
  else {
    srandom(atoi(argv[1]));
  }

  // ステップ 1 (0 世代目)
  for(i=0; i<M; i++) {
    for(j=0; j<N; j++) {
      individual[i].gene[j] = flip(0.5);
    }
    individual[i].fitness = evaluation(individual[i].gene); // 個体の適応度計算
  }
  print_process(individual, 0); // 初期世代の個体群を表示

  // ステップ 2 (1〜T 世代)
  for(t=1; t<=T; t++) {
    // 交叉
    one_point_crossover(individual);
    // 突然変異
    mutation(individual);
    // 子個体の適応度値計算
    for(i=0; i<M; i++) {
      individual[i].fitness = evaluation(individual[i].gene);
    }
    // ルーレット選択
    roulette_selection(individual);
```

```
    print_process(individual, t);
  }

  return(0);
}   // End of main()


// 個体の適応度計算
float evaluation(int *a) {
  int j;   // 遺伝子座インデックス
  int count = 0;   // 遺伝子中の'1' の数

  for(j=0; j<N; j++) {
    count += a[j];
  }
  return((float)count);
}   // End of evaluation()


// 一点交叉
void one_point_crossover(struct genotype *ind) {
  int i, ia, ib;   // 個体インデックス
  int j;   // 遺伝子座インデックス
  int c;   // 交叉点
  int test[M];   // 個体の利用フラグ
  int temp[N];   // 遺伝子を入れ替えるための仮変数
  int r;   // 乱数値

  for(i=0; i<M; i++) test[i] = 0;

  ia = ib = 0;
  for(i=0; i<M/2; i++) {
    // 個体をランダムにペアリング (親個体 ia,ib を選ぶ)
    // 親 ia を決定
    for(; test[ia]==1; ia=(ia+1)%M);
    test[ia] = 1;
    r = random() % (M-2*i) + 1;
    // (ia とは異なる) 親 ib を決定
    while(r>0) {
      ib=(ib+1)%M;
      for(; test[ib]==1; ib=(ib+1)%M);
      r--;
    }
    test[ib] = 1;
    // 個体 ia と ib の遺伝子を入れ替える
    if(flip(Pc)) {
      c = random() % N;
      for(j=0; j<c; j++) {
        temp[j] = ind[ia].gene[j];
        ind[ia].gene[j] = ind[ib].gene[j];
        ind[ib].gene[j] = temp[j];
      }
    }
  }
}   // End of one_point_crossover()


// 突然変異
void mutation(struct genotype *ind) {
  int i;   // 個体インデックス
  int j;   // 遺伝子座インデックス

  for(i=0; i<M; i++)
    for(j=0; j<N; j++)
      if(flip(Pm)) {
        ind[i].gene[j] = (ind[i].gene[j] + 1) % 2;
      }

}   // End of mutation()


// ルーレット選択
```

```
void roulette_selection(struct genotype *ind) {
  int h, i;   // 個体インデックス
  float total_fitness;   // 適応度の合計値
  float dart;   // 矢
  float wheel;  // ルーレット・ホイール
  struct genotype ind_new[M];   // 選択操作後の個体集合

  // 適応度の合計値を計算
  total_fitness = 0;
  for(i=0; i<M; i++) total_fitness += ind[i].fitness;

  // ルーレット・ホイールに従って次世代 (ind_new[]) を決定
  for(i=0; i<M; i++) {
    dart = (float)random() / RAND_MAX;
    h = 0;
    wheel = ind[h].fitness / total_fitness;
    while(dart > wheel && h < M-1) {
      h++;
      wheel += ind[h].fitness / total_fitness;
    }
    ind_new[i] = ind[h];
  }

  // 個体集合の更新
  for(i=0; i<M; i++) {
    ind[i] = ind_new[i];
  }

}   // End of roulette_selection()


// 引数‘prob’ の確率で 1 を返す
int flip(float prob) {
  float x = (float)random() / RAND_MAX;

  if(x<prob) return(1);
  else return(0);
}   // End of flip()


// 個体の中身や適応度値を画面に出力
void print_process(struct genotype *ind, int generation) {
  int i;   // 個体インデックス
  int j;   // 遺伝子座インデックス
  float max_fit, min_fit, avg_fit;   // 最大，最小，平均適応度

  // 各個体の中身を出力
  printf("\nGeneration: %d\n", generation);
  for(i=0; i<M; i++) {
    printf("%d: ", i);
    for(j=0; j<N; j++) {
      if(ind[i].gene[j] == 0) printf("%c", ' ');
      else printf("%c", '*');
    }
    printf(" : %.0f\n", ind[i].fitness);
  }

  // 個体集団の最大，最小，平均適応度を求める
  max_fit = min_fit = ind[0].fitness;
  avg_fit = ind[0].fitness / M;
  for(i=1; i<M; i++) {
    if(max_fit < ind[i].fitness) max_fit = ind[i].fitness;
    if(min_fit > ind[i].fitness) min_fit = ind[i].fitness;
    avg_fit += ind[i].fitness / M;
  }
  printf("max: %.2f  min: %.2f  avg: %.2f\n", max_fit, min_fit, avg_fit);

}   // End of print_process()
```