Level3 : 不連続関数における探索手法を検討し、その効率を示せ

Level3.1〜3.3 に示した関数はステップ関数 (不連続関数) であり、微分を活用した解法は適用できない。解空間 (x 軸) 全てを調べる事なく、一部を調べる事で最適解を求めるにはどうすれば良いか、検討せよ。

Level 3.1 : y=x*sin(x) with steps

まず、$y = x * sin(x)$ の最小値のもとめ方を検討する。
探索の手続きとして私たちは次の 2 つの方法を考えた。

- 方法 1
  1. ランダムにある x の値を出し、その x が正なら右に、負なら左に動く
  2. x の両隣を比較し、x より小さい値の方へ移動
  3. 両隣が x の値より大きくなると終了
- 方法 2
  1. 定義された x の範囲 $X - MIN <= a <= X - MAX$ となる座標 a を指定する.
  2. 指定した座標 a から定義された x の全体の範囲の 20%とする範囲を探索範囲とする.
  3. 決定した範囲を X-MIN に近いほうから順にすべて値を探索する.
  4. 手順 3 で探索した結果で、一番小さい値を解とする.

効率のよさを考慮し、方法 2 について詳しく考える。
図 2 に、方法 2 のフローチャートを示す。

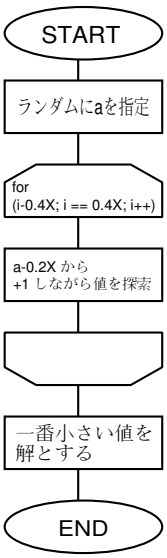

図 1: フローチャート

この方法は問題の式の範囲を X とし、ランダムに指定した場所を a とすると探索範囲 k は　$a - 0.2X \leqq k \leqq a + 0.2X$　となる。
よって、
問題空間サイズ 10 のとき $2 \times (10 \times 0.2) = 4$
問題空間サイズ 100 のとき $2 \times (100 \times 0.2) = 40$
問題空間サイズ n のとき　$2 \times (n \times 0.2) = 0.4n$
と、解を構成する要素の増加に伴う問題空間サイズの増加具合を計算量で示すことができる。

図 1 に問題空間サイズの変化のグラフを示す。

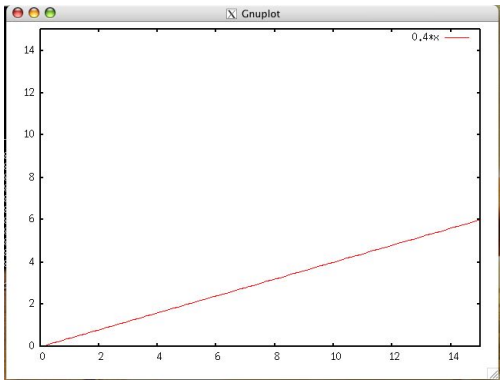

図 2: 問題空間サイズの増加の変化

また、1000/s の処理をする計算機があると仮定すると、要素数/問題空間サイズと全探索時の実行時間の関係は、表 1 のようになる。

表 1: 計算時間の例

全体の要素数 N	問題空間サイズ	実行計算時間 (秒)
10	4	0.004s
100	40	0.04s
1000	400	0.4s
10000	4000	4.0s

方法 2 は、全範囲の 40%を探索するため処理は全探索より早いという利点があるが、$y = x * sin(x)$ の指定範囲が大きければ大きいほど探索する範囲が広くなるため、処理速度が遅くなってくるという欠点がある。

Level 3.2 : ナップサック問題

複数の品物 (それぞれの品物は、重さと値段が異なる) が与えられた時、重さがナップサックに入る最大重量以内でなるべく合計の値段が最大になるような品物の組み合わせを求めよ。

私たちの考えた探索方法を以下に示す。

1. 商品を値段の高い順にソートする。

2. 上から重さを計算していく。

3. 重さが最大積載量に満たない場合、値段と商品名 (番号など) を値にして格納する

4. 重さが最大積載量を上回った場合には 2 のように計算はしない

この探索方法のフローチャートを図 3 に示す。

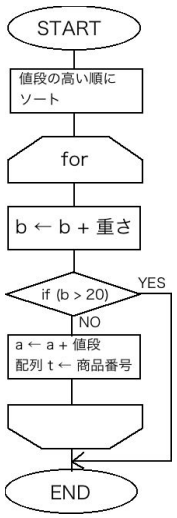

図 3: フローチャート

商品をマージソートでソートすると考えると、計算時間は $O_{(n)} = nlog_2 n$ となる。

問題空間サイズはソートを用いるため商品の数となる。図 4 にそのときの変化のグラフを示す。

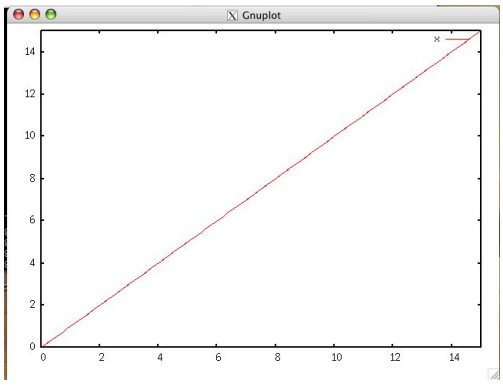

図 4: 問題空間サイズの増加の変化

また、1000/s の処理をする計算機があると仮定すると、要素数/問題空間サイズと全探索時の実行時間の関係は、表 2 のようになる。

表 2: 計算時間の例

全体の要素数 N	問題空間サイズ	実行計算時間 (秒)
10	33.21	0.033s
20	86.43	0.086s
30	147.20	0.147s
100	664.38	0.664s

この探索方法は、値段を基準にソートをしてるため計算が単純になり計算時間が短くなるという利点があるが、逆に値段でしか判別してないため重量が厳密にはならないという欠点がある。

Level 3.3 : 巡回セールスマン問題

ある 2 次元空間上のマップにおいて複数の都市が与えられた時、全ての都市を巡回するのにようするコスト (巡回経路長) を最小化せよ。

わたしたちが考えた巡回経路長を最小化する方法を以下に示す.

1. まず、スタート地点をランダムで決定する.

2. スタート地点を中心とした探索円を任意の近い都市が見つかるまで広げていく.

3. もし見つけた都市が行ったことのない都市だった場合にはその都市へ行く．そうでない場合は手順 2 に戻る.

4. 上記の手順でたどり着いた都市が「スタート地点である，かつその時点で全ての都市が探索済みである」場合には 1 周してきたという事で巡回が終了する.

図 5 にこの探索方法のフローチャートを示す。

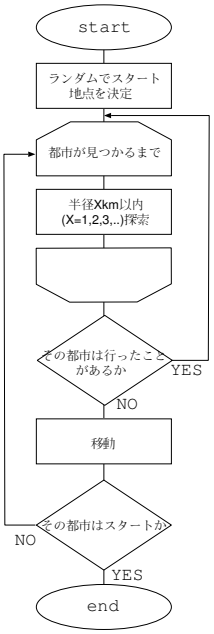

図 5: フローチャート

上記最小化法を用いた場合の問題空間は,
一つ目の都市を探す.....9 通り
二つ目の都市.....8 通り
・
・
・
一番最後の都市.....1 通り
つまり問題空間サイズが 10，20...N と増加していった時、

N!通り

である.

問題空間サイズは N!である。図 6 にそのときの変化のグラフを示す。

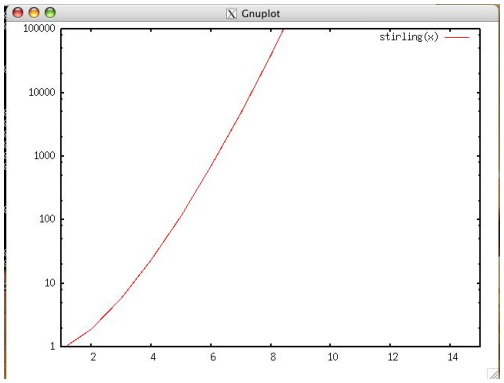

図 6: 問題空間サイズの増加の変化

また、N を 6、7、8、9 と変えていったとき,

1 秒間で 1000 個の要素を処理できる計算機

を用いた場合の計算時間の例を以下に示す.

表 3: 計算時間の例

全体の要素数 N	問題空間サイズ	実行計算時間 (秒)
6	720	0.72s
7	5040	5.04s
8	40320	40.32s
9	362880	362.88s

この方法は、一番近い都市を確実に見つけられるという利点があるが、場合によっては、同じ所を回ってしまうという欠点がある。