G-02

線形順序付け問題に対する局所探索法の効率化
Efficient local search for the linear ordering problem

大藤 聖也†　　山口 一章†
Seiya Daitou　Kazuaki Yamaguchi

1. まえがき

スポーツの大会等では，複数の参加者の順位を一意に決定する場合がある．水泳や走り幅跳びのように，それぞれの参加者がタイムや距離などの記録を持っている場合は容易に順位を決定できるが，テニスやサッカーのリーグ戦のように 1 対 1 での対戦結果の累計で順位付けが行われる場合は容易ではない．例えば，参加者 A，B，C に対し，A が B に勝ち，B が C に勝ち，C が A に勝ったとき，どのように順位をつけても下位の参加者が上位の参加者に勝っているという矛盾が生じる．ここで，大会の参加者と頂点を対応させ，各対戦について勝者から敗者に向かって有向辺を接続した有向グラフを考える．この有向グラフに閉路がない場合には参加者の順位を矛盾の無いように決定できる．閉路がある場合には，閉路を構成する辺の一部を削除，つまり，対戦結果を無視することで，閉路をなくして順位付けを行う．この際，削除する辺の本数（対戦結果を無視した数）を最小化することが，妥当な順位付けをすることと同義だと考えられる．これは最小帰還辺集合問題という最適化問題として知られる．例を図 1.1 に示す．例においては，BACD という順位付けが最善であり，B と D の対戦結果のみを無視すれば，対戦結果から矛盾が消える．

最小帰還辺集合問題の一般化として，各辺に重みが付けられた有向グラフにおいて，削除する辺の重みの和を最小にするような最適化問題が考えられている．この問題は線形順序付け問題（Linear Ordering Problem，以下 LOP と略記）と呼ばれる．より厳密な定義としては，各有向辺(u, v)に重みw_{uv}が付けられた辺重み付き有向グラフ$G = (V, E)$が与えられたときに，頂点の順序$\pi: \{1,2,\cdots,N\} \to V$（ただし$N = |V|$）に関する以下の目的関数$cost(\pi)$を最小化するような$\pi$を求める問題である．

$$cost(\pi) = \sum_{1 \le i \le j \le N} w_{\pi(j)\pi(i)}$$

LOP は NP 困難であり [1]，LOP に対する発見的手法は多数考案されている [2]．本研究では，最適解を求めることが困難な規模の入力に対して良質な解を高速に得ることを目指す．本研究では局所探索法を基にした発見的手法の効率化を提案する．また，計算機実験によって提案手法の有効性を評価する．

	A	B	C	D
A	／	×	○	○
B	○	／	○	×
C	×	×	／	○
D	×	○	×	／

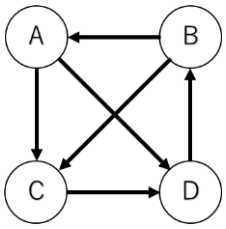

図 1.1　対戦結果（左）と対応する有向グラフ（右）．

†神戸大学大学院工学研究科，Graduate School of Engineering, Kobe University

2. 準備

局所探索法（Local Search，以下 LS と略記）は，多くの発見的手法の基となる考え方である．LS では，ある法則に従って，ある実行可能解に変更を加えることで別の実行可能解を生成する操作を近傍操作，近傍操作によって得られる解の集合を近傍と呼ぶ．また，ある解の近傍に，その解自身より目的関数値が真に優れた解が存在しないとき，その解は局所解であるという．LS は，初めに任意の実行可能解を保持した状態から，近傍の解への遷移を繰り返し行うことで，より優れた解を目指すアルゴリズムである．この近傍の解への遷移のうち，現在より目的関数値が優れた解への遷移を改善操作，現在と同値かそれより悪い解への遷移を改悪操作と呼ぶ．遷移先を近傍のみに制限することで高速に計算を行うことができる．

局所探索法の考え方を基にした解法は多く存在するが，その中で基本的な解法である山登り法，ランダム局所探索法，焼きなまし法の三つを説明する．山登り法は，現在保持している解πに対するすべての近傍の中から最も優れた解π'を選び，π'がπより優れている場合に，π'へ遷移し保持している解を常に改善させながら局所解に到達するまで探索を続ける手法である．ランダム局所探索法は，山登り法と同様の近傍操作を行い，π'がπより劣っている場合にも低確率でπ'へ遷移することで探索空間内の網羅性を向上させる手法である．焼きなまし法は，ランダム局所探索法と同様の操作を行う中に，温度というパラメータを用意し，解を悪くするような遷移が発生する確率を時間経過とともに低下させる手法である．

3. 線形順序付け問題に対する既存手法

順列を求める組合せ最適化問題に対する代表的な近傍操作には，交換操作や挿入操作がある．LOP においては，交換操作で改善できる解は挿入操作でも改善できるが，挿入操作で改善できる解を交換操作で改善できるとは限らないことが知られている．そのため，本研究においては近傍操作に挿入操作を採用し，ある解に対して一回の挿入操作を行って得られる解の集合をその解の近傍と呼ぶ．また，任意の解πに対して，先頭からi番目の頂点をj番目の位置に挿入する操作を$Insert(\pi, i, j)$と表し，その間の距離$|i-j|$を頂点の移動距離と呼ぶ．

長さNの任意の順列に対する近傍は，(移動する頂点の選び方N通り)×(移動先の選び方$N-1$通り)の$N(N-1)$通りである．しかし，この近傍すべてを探索して，その中でも最も優れた解へ遷移するのは効率的ではない．なぜなら，一回の挿入操作に必要な時間計算量が$O(N^2)$となってしまうためである．また，ある局所解πの近傍で最も優れた解がπ'で，その解π'の近傍で最も優れた解がπのとき，互いの解への遷移を続けてしまい，この局所解πを脱出できないためである．そこで，移動する頂点をランダムに選び，移

動先をすべて探索し，最も優れた解を探索し，それが改善操作の場合はその近傍の解へ遷移し，改悪操作の場合でも低確率でその近傍の解への遷移を許す．これにより，一回の探索及び挿入操作に必要な時間計算量を$O(N)$に抑えることができ，特定の局所解に停滞することを防ぐ．

3.1 ランダム局所探索法

本研究におけるランダム局所探索法（Random Local Search，以下 RLS と略記）は，まず初期解πの順列をランダムに生成し，そのコストを計算する．その後，指定したループ回数の間，以下のステップを繰り返し実行する．

（1）$[1,N]$のランダムな整数値iを生成し，先頭からi番目の頂点を移動する頂点とする．

（2）i番目の頂点をj番目$(j = 1,2,\cdots,N, j \neq i)$に移動したときのコスト変化量を計算し，コスト変化量が最小となる位置jをj_{best}，そのときのコスト変化量をΔcとする．

（3）コスト変化量Δcが負の場合（つまり改善操作の場合），$Insert(\pi,i,j_{best})$を実行する．Δcが 0 または正の場合，確率pで$Insert(\pi,i,j_{best})$を実行する．この確率pを改悪時遷移確率と呼ぶ．

（4）Δcが負の場合，その解とその時のコストを更新する．

各ステップで必要とする時間計算量は，ステップ(1)では$O(1)$，ステップ(2)ではコスト変化量をi番目の頂点に近い方から順に計算していくことで$O(N)$，ステップ(3)では挿入操作がある場合は$O(N)$，ない場合は$O(1)$，ステップ(4)では解の更新で$O(N)$，コストの更新で$O(1)$である．したがって，ステップ(1)～(4)を行う 1 ループあたりの時間計算量は$O(N)$である．厳密には，ステップ(1)～(4)の前段階でのコスト計算はすべての 2 頂点間の辺の重みをすべて調べる必要があるため，$O(N^2)$の時間計算量がかかるが，この作業は 1 度問題を解くのに 1 度しか行われない．これは，繰り返し行われる部分の実行時間よりも小さいので，実行時間の増大に大きく寄与することは無い．LOP に対する RLS とステップ 2 の最善の挿入先を探す$searchBest(\pi,i)$メソッドの疑似コードをそれぞれ$Algorithm1$と$Algorithm2$に示す．

$Algorithm1$は，1～4 行目で初期解の生成とそのコスト計算を行い，その後，5～16 行目を指定したループ回数だけ繰り返す．6 行目で移動する頂点iをランダムに選び，7 行目でその頂点iの移動先として最善の位置jとその挿入によるコスト変化量Δcを，$searchBest(\pi,i)$によって求める．

$Algorithm1$ LOP に対する RLS

```
1: π ←初期解を生成.
2: c ← πのコストを計算.
3: π_best ← π
4: c_best ← c
5: while 終了条件を満たしていない do
6:   i ← [1,N]からランダムに整数を決定し，移動する頂点とする.
7:   (j, Δc) ← searchBest(π, i)
8:   if Δc < 0 ‖ 確率p then
9:     π ← Insert(π, i, j)
10:    c ← c + Δc
11:    if c < c_best then
12:      π_best ← π
13:      c_best ← c
14:    end if
15:  end if
16: end while
17: return π_best
```

$Algorithm2$ $searchBest(\pi,i)$

```
1: Δc_best ← +∞
2: j_best ← i
   //末尾側への挿入のコスト計算.
3: Δc ← 0
4: for j = i + 1 to N do
5:   Δc ← Δc + w_π(j)π(j−1) − w_π(j−1)π(j)
6:   if Δc < Δc_best then
7:     Δc_best ← Δc
8:     j_best ← j
9:   end if
10: end for
   //先頭側への挿入のコスト計算.
11: Δc ← 0
12: for j = i − 1 to 1 do
13:   Δc ← Δc + w_π(j)π(j+1) − w_π(j+1)π(j)
14:   if Δc < Δc_best then
15:     Δc_best ← Δc
16:     j_best ← j
17:   end if
18: end for
19: return (j_best, Δc_best)
```

$\Delta c < 0$であるか，$\Delta c \geq 0$かつ改悪時遷移確率pで，9,10 行目で解の更新を行う．それが現在の最適解のコストよりも小さければ，12,13 行目で最善解とその時のコストを保持しておく．

$Algorithm2$は，4～10 行目で移動する頂点iより末尾側へ挿入した場合のコスト変化量を計算する．このとき，一つずつ末尾側へずらしていくことで計算量$O(N)$での計算を可能としている．同様にして，12～18 行目で移動する頂点iより先頭側へ挿入した場合のコスト変化量を計算する．19 行目で最も優れた遷移先を戻り値として返す．

3.2 兼本らによる RLS の改良

兼本ら[3]により，LOP に対する RLS においては，改悪操作が実行される際，頂点の移動距離が極端に長い場合や短い場合，他の局所解を発見しづらくなることが知られている．そこで，改悪操作時に限定して，移動する頂点の挿入先に制限を設ける．その制限方法として以下の二つがある．

1．最小移動距離d_{min}を用意し，移動距離がd_{min}以下となるような改悪操作は行わない．

2．改悪時に，目的関数値に関係なく移動距離が$N/2$となるような挿入操作を行う．

1 の手法は，改悪時の LOP の特徴を考慮した改悪操作を行いつつ，その操作によって解が悪くなりすぎないように配慮した設計となっている．これにより，むやみに解の性能を悪化させて時間がかかるのを防いでいる．一方，2 の手法では，1 の手法よりも改悪時の LOP の特徴を重要視し，改善につながるようなパターン数を最大化することで，他の局所解へ進みやすくなるよう設計されている．これにより，解空間をより網羅的に探索することを可能としている．

本実験では，2 の手法を用いるため，この手法を単に，兼本による手法と呼ぶ．具体的な RLS からの変更点は，探索時に，コスト変化量の最小値Δc_{min}とは別に，移動距離が$N/2$となる位置への挿入時コスト変化量Δc_{half}を記録しておき，$\Delta c_{min} < 0$ならば$\Delta c = \Delta c_{min}$，$\Delta c_{min} \geq 0$ならば$\Delta c = \Delta c_{half}$とし，$\Delta c$が負の場合，$Insert(\pi,i,j_{best})$を実行し，$\Delta c$が 0 または正の場合，確率$p$で$Insert(\pi,i,(i+\frac{N}{2})\%N)$を実行する点である．ただし，A%B は A を B で割った余りとする．これにより，想定通りの操作を行いつつ，実行

Algorithm3 兼本らによる手法での$searchBest2(\pi,i)$

1: $\Delta c_{best} \leftarrow +\infty$
2: $j_{best} \leftarrow i$
3: $\Delta c_{half} \leftarrow 0$
4: $j_{half} \leftarrow (i+\frac{N}{2})\%N$
//末尾側への挿入のコスト計算.
5: $\Delta c \leftarrow 0$
6: for $j = i+1$ to N do
7: $\Delta c \leftarrow \Delta c + w_{\pi(j)\pi(j-1)} - w_{\pi(j-1)\pi(j)}$
8: if $\Delta c < \Delta c_{best}$ then
9: $\Delta c_{best} \leftarrow \Delta c$
10: $j_{best} \leftarrow j$
11: end if
12: if $j = j_{half}$ then
13: $\Delta c_{half} \leftarrow \Delta c$
14: end if
15: end for
//先頭側への挿入のコスト計算.
16: $\Delta c \leftarrow 0$
17: for $j = i-1$ to 1 do
18: $\Delta c \leftarrow \Delta c + w_{\pi(j)\pi(j+1)} - w_{\pi(j+1)\pi(j)}$
19: if $\Delta c < \Delta c_{best}$ then
20: $\Delta c_{best} \leftarrow \Delta c$
21: $j_{best} \leftarrow j$
22: end if
23: if $j = j_{half}$ then
24: $\Delta c_{half} \leftarrow \Delta c$
25: end if
26: end for
//改善操作になるか改悪操作になるかに応じて，戻り値を決定.
27: if $\Delta c_{best} < 0$ then
28: return $(j_{best}, \Delta c_{best})$
29: else
30: return $(j_{half}, \Delta c_{half})$
31: end if

時間の増大を防ぐことができる．$\Delta c_{min} \geq 0$ならば，$\Delta c_{half} \geq 0$であることは明らかなので，この手法は$Algorithm1$をそのまま使用でき，RLS で用いた$searchBest(\pi,i)$に変更を加えることで実装できる．そのメソッドを$searchBest2(\pi,i)$と呼び，疑似コードを$Algorithm3$に示す．また，この手法は，RLS の改悪時の挿入先に変更を加えただけであり，そのときのコスト変化量も探索の過程で計算しているため，1 ループあたりの時間計算量は$O(N)$である．

$Algorithm3$の，$Algorithm2$からの主要な変更点は，3,4,12,13,23,24,28 行目である．3,4 行目，または 12,13 行目で，コスト変化量の探索時に，同時に$\frac{N}{2}$個離れた場所への挿入操作をした場合のコスト変化量を保持しておくことができる．

4. 提案手法

提案手法では，移動する頂点iの候補を重複しないように二つ用意し，それぞれの頂点iに対して最善の移動先jを同時に探索する．その後，それぞれのコスト変化量Δcに応じた処理を行う．また，本研究で提案する手法では，改悪操作において，RLS から性能の向上が確認された兼本による手法のうち，二番目に紹介した「改悪時に，目的関数値に関係なく移動距離が$N/2$となるような挿入操作を行う」というものを採用する．

移動する頂点の候補i_1, i_2をランダムに決定し，それぞれ

Algorithm4 提案手法

1: π ←初期解を生成.
2: $c \leftarrow \pi$のコストを計算.
3: $\pi_{best} \leftarrow \pi$
4: $c_{best} \leftarrow c$
5: while !終了条件 do
6: i_1, i_2 ←移動する頂点を$[1,N]$から異なる二つの整数をランダムに決定.
7: $(j_1, \Delta c_1) \leftarrow searchBest2(\pi, i_1)$
8: $(j_2, \Delta c_2) \leftarrow searchBest2(\pi, i_2)$
9: if $\Delta c_1 < \Delta c_2$ then
10: if $\Delta c_1 < 0$ || 確率pの抽選 then
11: $\pi \leftarrow Insert(\pi, i_1, j_1)$
12: $c \leftarrow c + \Delta c_1$
13: end if
14: else
15: if $\Delta c_2 < 0$ || 確率pの抽選 then
16: $\pi \leftarrow Insert(\pi, i_2, j_2)$
17: $c \leftarrow c + \Delta c_2$
18: end if
19: end if
20: if $c < c_{best}$ then
21: $\pi_{best} \leftarrow \pi$
22: $c_{best} \leftarrow c$
23: end if
24: end while
25: return π_{best}

の最善の移動先j_1, j_2とコスト変化量$\Delta c_1, \Delta c_2$を求める．$\Delta c_1 < \Delta c_2$のとき，$\Delta c_1 < 0$ならば$Insert(\pi, i_1, j_1)$を実行し，$\Delta c_1 \geq 0$ならば$Insert(\pi, i_1, (i_1+\frac{N}{2})\%N)$を実行する．$\Delta c_1 \geq \Delta c_2$のとき，$\Delta c_2 < 0$ならば$Insert(\pi, i_2, j_2)$を実行し，$\Delta c_2 \geq 0$ならば$Insert(\pi, i_2, (i_2+\frac{N}{2})\%N)$を実行する．この提案手法の疑似コードを$Algorithm4$に示す．また，この手法における時間計算量は，移動する頂点の候補i_1, i_2それぞれの計算は兼本による手法と同じため$O(N)$，その後の処理もi_1, i_2のどちらか一方に対して，兼本による手法と同じ処理を行うため$O(N)$である．したがって，この提案手法の 1 ループあたりの時間計算量も$O(N)$である．

$Algorithm4$は，1～4 行目で初期解の生成とそのコスト計算を行い，その後，5～16 行目を指定したループ回数だけ繰り返す．6 行目で異なる二つの移動する頂点i_1, i_2をランダムに選び，7,8 行目でその頂点i_1, i_2のそれぞれに対し，移動先として最善の位置j_1, j_2とその挿入によるコスト変化量$\Delta c_1, \Delta c_2$を，$searchBest2(\pi,i)$によって求める．$\Delta c_1, \Delta c_2$の小さい方に対して，0 より小さいか，改悪時遷移確率pの抽選に当たれば，9～19 行目で解の更新を行う．それが現在の最適解のコストよりも小さければ，21,22 行目で最善解とその時のコストを保持しておく．

5. 計算機実験

本章では，兼本による手法，提案手法の 2 つを用いて 485 問のベンチマーク問題を以下の条件で 5 回ずつ解き，その結果から 2 つのアルゴリズムの性能を比較する．

実験環境

- OS：Windows 10
- CPU：Intel Core i7-7700HQ 2.80GHz
- メモリ：16.0GB
- 言語：JavaSE 11

実験条件

- ループ数10^6

・改悪時遷移確率$p = 0.01$

なお，ベンチマーク問題として使用する LOLIB の 485 問のベンチマーク問題は，LOP において順方向の辺の重みの総和を最大化する問題として解いた際の最適解もしくは既知の最善解が判明している．

実験の結果，提案手法は兼本による手法に対して，192 勝 109 敗 184 分であり，提案手法が兼本による手法よりも良い性能を示した．

6. あとがき

本研究では，線形順序付け問題に対する局所探索法を用いた既存手法を改良することによって網羅性の低さを軽減し，アルゴリズムの効率化を達成した．

今後の課題として，移動する頂点の選択法や改悪時遷移確率に対する焼きなまし法の有効性などの調査が必要である．また，解探索に該当する処理の並列化を試みたが，並列化による処理時間の短縮よりも並列化の同期処理による処理時間の増大による影響の方が大きく，有効的な手段ではなかった．

参考文献

[1] M. Garey, D. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman and Company, New York, 1979.

[2] 櫻庭セルソ智，柳浦睦憲，“線形順序付け問題の解法，” オペレーションズ・リサーチ：経営の科学, vol.57, no.6, pp.327-334, June 2012.

[3] 兼本樹，山口一章，増田澄男，“線形順序付け問題に対する焼きなまし法の近傍選択法，” 平成 30 年電気関係学会関西連合大会　講演論文集，G10-4, 2018.

[4] Optsicom Project, “Linear Ordering Problem”, https://grafo.etsii.urjc.es/optsicom/lolib/, 参照 July. 2021.