平成26年度　特別研究報告書

SOM-TSP 法についての検討と改良手法の提案

龍谷大学　理工学部　情報メディア学科

学籍番号　T110490　向　智也

指導教員　三好　力　教授

内容梗概

組み合わせ最適化問題の 1 つに,巡回セールスマン問題（traveling salesman problem：TSP）がある．この問題は他の組み合わせ最適化問題と比べてあまり制約が存在しないためベンチマーク問題として様々な解法が考えられている．そのうちの一つが自己組織化マップ(Self Organization Mapping：SOM)を用いた SOM- TSP 法である．

今回,既存研究である改良 SOM-TSP 法を元に,経路短縮を目的とした 交差経路の発見と補正,計算時間の短縮を目的とした近傍関数の簡略化などの改良手法を提案し,既存研究における解法との比較実験により検証を行った．

目次

第１章　はじめに

計算複雑性理論と呼ばれる計算の複雑さを扱う分野には,NP と呼ばれる問題の集まりがある［2］．NP は以下の二つの条件で定義される．

1. 非決定性チューリングマシンによって多項式時間で解くことができる問題
2. yes となる証拠が与えられたとき,その証拠が本当に正しいかどうかを多項式時間で判定できる問題

これらはお互いが同値であることが証明されており,一般的には 2 の定義で説明されることが多い．また, NP 自体はミレニアム懸賞金問題の一つである P≠NP 予想でも有名である．NP にはいくつかクラスが存在し,その中でも一般的に NP 困難と呼ばれるクラスが最も解くのが難しいとされているが,そのクラスに属する問題の中に,巡回セールスマン問題と呼ばれるものがある．

巡回セールスマン問題（traveling salesman problem：TSP）[1,3]とは,セールスマンがある都市を出発し,全ての都市に一度ずつ訪問してもう一度出発した都市に戻ってくるときにセールスマンが通った経路長が最短のものを求める問題である．この問題は他の組み合わせ最適化問題と比べると制約があまりなく,最適化アルゴリズムの基本性能をみる指標の一つとされているため多くの研究が行われてきた．手法の種類としては,分枝限定法[4]や焼きなまし法[5],ホップフィールドモデル[6]やボルツマンマシン[7]を用いて解く方法がある．しかし,どの方法においても最適解（理論上の最短経路の解）や準最適解（理論上の最短経路ではないが最短に近い経路）を求めるのに長い時間を必要とし,またパラメータ設定が困難という問題があった．

この問題に対して,1988 年に Angeniol 等は自己組織化マップ(Self Organization Mapping：SOM)を TSP の解法に用いた[9]．この解法では,現実的な所要時間で準最適解を求めることができ,また最適解を求めることが出来ないという点を除けば従来の問題点を全て克服した．また,先行研究として上記の Angeniol 等の解法を改良する研究が行われている[10,11]．この研究では,ノードの総数の変化と都市数に依存しているパラメータを追加し,近傍関数の大きさを制御するパラメータの更新式を変更することで,経路長合計をほとんど変えずに計算時間を短縮することに成功している．

本研究では,先行研究を元にした新たな近傍関数を提案する．近傍関数は直線で近似することで計算時間を短縮しながら,近傍関数の大きさを制御するパラメータの役割を新たなパラメータの追加によって引き継ぐ．また,合計経路の短縮方法として交差経路の訂正法を提案する．提案するアルゴリズムと従来のアルゴリズムを比較のための実験を行い,検証する．

第 2 章　基本事項

2.1　SOM とは

自己組織化マップ（Self-Organization-Mapping：SOM）[12]とは,競合学習型ニューラルネットワークの一つであり,入力層と出力競合層の 2 層から成る表現モデルである．自己組織化マップは 1980 年代にコホネンによって開発され,多次元データの分類,解析を容易にする技術として知られている．

自己組織化マップの特徴として,n 次元ベクトル集団を学習することで 2 次元マップにそれらのベクトルの関係を写像することができることが挙げられる．似ているベクトルは 2 次元マップ上の近い位置に配置され,そうでないベクトルは遠い位置に配置される.汎用性が高く,認識,予測,分類など様々な応用が可能である.

2.2　SOM の基本的なアルゴリズム

コホネンは,人間の脳の情報処理の仕方を以下の式で表した.

$$m_i(t+1) = m_i(t) + h_{ci}(t)[x(t) - m_i(t)] \tag{2.1}$$

この式は,次のような意味を持っている.神経細胞(ノード)i　が時刻 t で処理している情報処理能力を$m_i(t)$とする．外部から入力信号 x(t) が入ってきた時,細胞はこの入力信号を学習して次の時刻には入力信号により近い情報処理能力$m_i(t+1)$を持つようになる．この時,x(t) が n 次元の入力ベクトルである場合,参照ベクトルとも呼ばれる$m_i(t)$もまた同じ n 次元の要素を持つ.そして,$h_{ci}(t)$は学習率係数を含めた近傍関数を意味する.

ここで,$t = 0,1,2,\ldots$は離散時間座標である．出力競合層のベクトルは参照ベクトル$m_i(t)$で表され,入力層の次元に合わせて n 個の要素を持っている．図 2.1 に示すように,視覚的に出力を見るために,普通は 2 次元に配列されている.

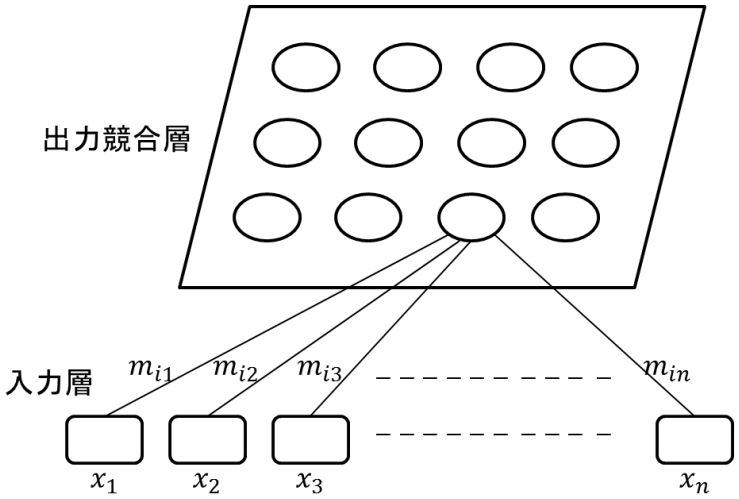

図 2.1：SOM の構造

自己組織化マップの学習は次のように行われる．入力ベクトル x(t)はある測度,例えばユークリッド距離 $|x(t)-m_i(t)|$ を最小にするノード i を探し,それに添え字 C をつける．

$$|x(t)-m_c| = min|x(t)-m_i(t)| \tag{2.2}$$

式(2.2)で決められた参照ベクトル$m_c(t)$を持つノードを勝者ノードと呼ぶ.式(2.1)と式(2.2)での学習の状態を図 2.2,図 2.3 に示す．まず入力ベクトルが提示されると,その入力ベクトルに一番近いノードが勝者ノードとなる．そして勝者ノードの周囲のノード,つまり図 2.3 の円で囲われている領域を近傍領域と定義する．この時,領域の形は正方形でもよい．たとえば基本ノードの配列を六角形の形をした場合は周囲 6 個の六角形が近傍領域ということになる．その近傍内の全てのノードは入力ベクトルを学習し,式(2.1)に従って入力ベクトルの方向へ修正される．この学習を繰り返し行っていく．この時近傍領域の範囲は最初は広く指定しておき,学習が進行するにつれて近傍領域を狭めていく.

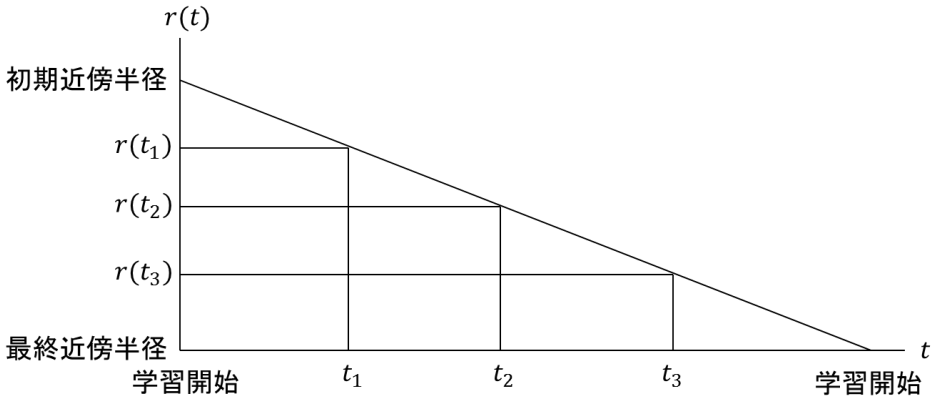

図 2.2:近傍領域を表す円の半径のグラフ

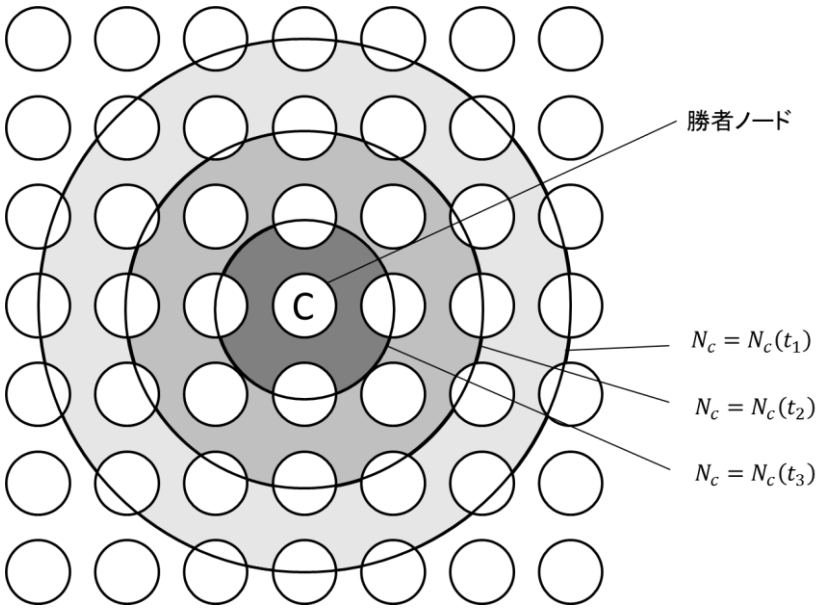

図 2.3:SOM の学習の繰り返しにおける近傍領域の変化

式(2.1)において,$h_{ci}(t)$は学習率係数$\alpha(t)$と近傍関数$h(d,t)$により以下の式のように表現できる.

$$h_{ci}(t) = \alpha(t) * h(d,t) \tag{2.3}$$

近傍領域は,学習される近傍領域を指定する関数である．近傍領域は,出力競合層の勝者ノードの近傍を意味し,学習によって参照ベクトルが更新される領域である．学習をする際,最初は近傍領域をその範囲を大きくとり,学習が進行するとともに徐々にその領域を狭める．近傍領域を減少させる関数には,線形型とステップ型が存在する．ここではよく使用されるガウス型近傍関数について説明する．この時, d は勝者ノードから参照ベクトルまでの距離,$k\big(r(t)\big)$は学習時間 t のときの近傍領域の最大距離で,ガウス関数の波幅の係数である.

$$(d,t) = exp\left(-d^2/k\left(r(t)\right)^2\right) \tag{2.4}$$

また,d は式(2.5)で表せる.

$$d^2 = (x-a_i)^2 + (y-b_i)^2 \tag{2.5}$$

ここで,(a_i, b_i)は勝者ノードの位置,(x,y)は半径$r(t)$から成る円形領域の内側にあるノードの位置を示す.

近傍領域の半径$r(t)$は,学習の初めでは大きく,学習が進行すると小さくなっていく．また, 半径$r(t)$の円領域に含まれる範囲のノードや参照ベクトルは,指数関数によって決められる重みを持って学習する．すなわち,勝者ノードに近いほどその類似性が大きくなるように学習され,勝者ノードから遠ざかるほど,その類似性が小さくなるように学習される．そして,半径$r(t)$の範囲外のノードは学習が行われないようになる.

したがって,式(2.1)によって学習している間,近傍領域N_c内のノードに関しては$h_{ci}(t) = \alpha(t) * h(d,t)$,近傍領域$N_c$外のノードに関しては,$h_{ci}(t)= 0$ である.つまり,近傍領域の外側の領域では学習は行われない.

以上より,近傍関係を以下の式で表す.

$$\begin{aligned} h_{ci}(t) &= \alpha(t) * h(d,t) \quad && (i \in N_c) \\ h_{ci}(t) &= 0 && (i \notin N_c) \end{aligned} \tag{2.6}$$

この時,$\alpha(t)$を学習率係数と呼び,$0 < \alpha(t) < 1$の値を持つ．$\alpha(t)$とN_cの大きさはどちらも学習時間が経つにつれて普通,単調減少させる．例えば,$\alpha(t)$であれば次の式で定義してもよい.

$$\alpha(t) = \alpha_0(1 - t/T) \tag{2.7}$$

ここで,a_0はaの初期値であり,普通 0.2～0.5 の値を選ぶ．T は行われるべき学習での予定された全更新学習回数である．ただし,式(2.1)中,比例係数の$a(t)$は学習の最初では大きな値をと

るように設定し,学習の進行とともにその値を減少させるようにする．また,近傍領域$N_c = N_c(t)$も式(2.6)と同様に減らしていってもよい．つまり,

$$N_c(t) = N_c(0)(1 - t/T) \tag{2.8}$$

ここで,$N_c(0)$は初期値である.

コホネンの自己組織化マップアルゴリズムは,整理すると以下の手順で行う.

1. 出力層にノードを配置し,それぞれの持つ参照ベクトルを乱数で初期化する.
2. 入力ベクトルに最も近い(類似している)競合層での参照ベクトルを探し,これを勝者ノードとする.
3. この勝者ノード及びその近傍内のノードを式(2.1)に従って更新する．また,学習が進むにつれて近傍領域を狭め,学習率係数の値も式(2.4)のように減少させていく.

自己組織化マップの学習を行うために,学習パラメータを設定する必要がある．ここに主要な学習パラメータとその説明を示す.

学習回数:競合層に対して行う学習の回数を設定する

学習係数:一回の学習でノードをどれだけ更新するか設定する

近傍領域:学習に影響を与える範囲の初期値を設定する

これらの自己組織化マップアルゴリズムを,Angeniol 等は TSP に応用した．そのため,SOM-TSP 法においても以上のパラメータは必要となってくる.

第 3 章　既存技術

先行研究である Angenial 等の解法を改良した SOM-TSP 法[9,10,11]について解説する．二次元平面上に M 個の都市が与えられているとき,各都市に$i = 1,2,\dots,M$の番号を付け,各 i 番目の都市は二次元座標(x_1^i, x_2^i)を持つ．また, ノードの個数を N 個とし,各ノードには j = 1,2,…,N の番号を付け,都市と同様に各 j 番目のノードは 2 次元座標(c_1^j, c_2^j)を持つ.

アルゴリズムの処理の流れを図 3.1 に示す．アルゴリズム開始時には適当な座標位置にノードが1つ存在するだけだが,後述するノード複製の処理により,ノードはリング状に成長していく.そして,最終的に各都市と各ノードの数と座標がほぼ同じになり全都市がそれぞれノードを「確保」したとき処理は終了し,ノードの番号の順番が経路となる.　図 3.1 にフローチャートを示す.

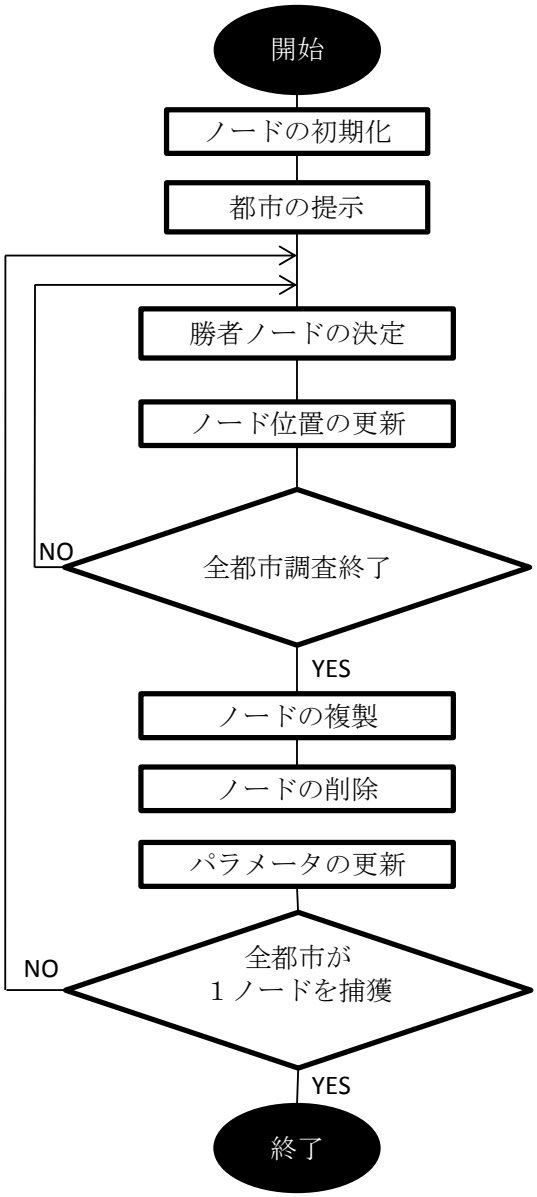

図 3.1:改良アルゴリズムのフローチャート

図 3.1 における各処理についてそれぞれ解説する.

- **ノードの初期化**

二次元平面上のランダムな座標(c_1^1, c_2^1)にノードを 1 つ与える．これが初期ノードとなる.

- **都市の提示**

与えられた都市を二次元平面上に配置する．また,都市調査を行う順番をランダムに決める.

- **勝者ノードの決定**

都市 i とのユークリッド距離が最少であるノードj_cを選ぶ．次に, 調査する都市 i と全ノードとのユークリッド距離V_jを以下の式により求める.

$$V_j = \left(x_1^i - c_1^j\right)^2 + \left(x_2^i - c_2^j\right)^2 \tag{3.1}$$

さらに,以下の式によりV_jを計算し, V_jが最も小さいノードを勝者ノードj_cとする.

$$V_{jc} = minV_j \tag{3.2}$$

- **ノード位置の更新**

勝者ノードj_cとその近傍のノード j を,調査する都市 i に近づくように更新する.

まず,更新しようとしているノード j が勝者ノードからリング状でどのくらいの位置関係にあるのかを以下の式により距離尺度 n として求める.

$$n = min|(j - j_c)modN, (j_c - j)modN| \tag{3.3}$$

式(3.3)において,更新するノード j がリング状の勝者ノードj_cから見て右回りおよび左回りに何番目のノードであるかということがわかるが,そのうち値の小さなものを選ぶ.

次に,全てのノード j を以下の式により更新する.

$$c_k^j \leftarrow c_k^j + f(G,n)\left(x_k^i - c_k^j\right)(k:1,2) \tag{3.4}$$

ここで,$f(G,n)$は更新率であり以下の式で表される.

$$f(G,n) = \frac{1}{\sqrt{2}} exp\left(\frac{-n^2}{G^2}\right) \tag{3.5}$$

G は更新する近傍領域を調整するパラメータであり,全都市調査終了ごとに後述の処理で更新する.

- **ノードの複製**

1 回の全都市調査において,同じノードが 2 つ以上の都市で勝者ノードj_cとして選ばれた場合, 全都市調査終了後にノードを複製する．複製されたノードは同じ座標をもち,勝者ノードj_cと隣接するノードj_{c+1}との間に挿入される.

- **ノードの削除**

もし連続した 3 回の全都市調査の間に,一度も勝者ノードj_cとして選ばれなかったノードを削除する.

- **パラメータの更新**

1回の全都市調査が終了した後,パラメータ G を以下の式で更新する.

$$G \leftarrow \alpha(t)G \tag{3.6}$$

$\alpha(t)$は更新係数であり,初期値は$0.0 < \alpha < 1.0$の範囲の値を選ぶ．$\alpha(t)$は以下の式で更新される.

$$\alpha(t+1) = \alpha(t) + \gamma\left(C(t) - C(t-1)\right)/M \tag{3.7}$$

この時,γは更新用のパラメータである．$\gamma < 0$のとき,$\gamma = 0$の場合と比べて計算時間が大幅に短縮されることが確認されており,具体的には$G = 10$,$\alpha(0) = 0.95$,$\gamma = -0.2$の条件下で,532 都市までの TSP に対して,$\gamma = 0$のときと比較して計算時間が約 1/5 に短縮される.

第 4 章　提案手法

4.1　交差経路の短縮

Angenial 等の解法によって求められた経路は最適解を求めることができず,場合によっては交差する経路が発生してしまう．この時, 交差する経路は交差しない経路と比較すると明らかに無駄な経路を通っていると考えられるため,訂正することで経路の短縮につながる.

本研究では,K 点交差経路短縮法として以下のアルゴリズムを提案した．また,K≥ 4であり,この手順が行われるのは全ての都市が1つノードを捕獲した後に一度だけである.

1. k=4 から K まで,k を 1 ずつ増やしながら 2 から 5 の手順を繰り返す.
2. あるノードを1つ目とした連続する k 個のノードに$1,2,\dots,k-1,k$の番号を付ける.
3. 1 と $2, k-1$と k のノードそれぞれをつないだ線が交差していたとき,2 と$k-1$のノードの本来のノード番号をそれぞれ入れ替える.
4. 3～k-2 のノード番号を逆順に付け直す.
5. 全てのノードを 1 つ目として 1 から 4 の手順を行う.

K 点交差経路短縮法における発見方法は,「直線を境界線として, 線分を構成する 2 つの点が直線に対して両側に存在するとき, 直線と線分は交差する」という定義をもとにして判定している[8].

4.2　近傍関数の簡単化

SOM において,近傍関数はノードの更新に関わる重要な関数である．既存研究では,近傍関数$f(G,n)$にはガウス型関数,距離尺度 n,パラメータ G を用いている．また,G の更新にはパラメータ γ を用いている．既存研究のように, 近傍関数にはガウス型関数を用いるのが一般的である．しかし, 勝者ノードに近いノードほど大きく更新する, 計算回数が増えるほど近傍領域を小さくする, という 2 つの効果さえ得られれば, 近傍関数に一次関数を用いることも可能である．また, 一次関数にはガウス型関数と比較すると計算量が小さいため, 計算時間を短縮できるという強みも存在する.

本研究では,既存研究における近傍関数を簡単化した新たな近傍関数$f(SL,n)$を提案する.近傍関数$f(SL,n)$は指数関数の部分を一次方程式で表し,距離尺度 n はそのまま用いる.

近傍関数$f(SL,n)$は,以下の式で表される.

$$\begin{cases} f(SL,n) = 0.0 & (\mathrm{SL} * \mathrm{n} + 1.0 \leq 0.0) \\ f(SL,n) = \frac{\mathrm{SL}*\mathrm{n}+1.0}{\sqrt{2}} & (\mathrm{SL} * \mathrm{n} + 1.0 > 0.0) \end{cases} \tag{4.1}$$

この時,SL は近傍領域における調整パラメータである．SL は一次方程式における傾斜を表し,SL の値の更新は1回の都市調査ごとに行う．更新式は既存研究においてパラメータ G の更

新に用いた式を改変したものを使用する．具体的には以下の式で更新する．また, 初期値は 0.005 である.

$$SL \leftarrow SL + \alpha(\mathrm{N}/M) \tag{4.2}$$

この時,αは更新用のパラメータであり,このパラメータにはいくつか候補が存在する.

図 4.1 に近傍関数$f(G,n)$と$f(SL,n)$の比較を示す．このグラフはそれぞれ最初の都市調査時における近傍関数を表しており, $f(SL,n)$のグラフは$f(G,n)$のグラフを近似したものとなっていることがわかる.

実験では, 既存研究の近傍関数$f(G,n)$と新たな近傍関数$f(SL,n)$それぞれにおける計算時間, 経路合計, 全都市調査の回数を比較する．また, αの値をいくつか変化させ,最適なαを求める.

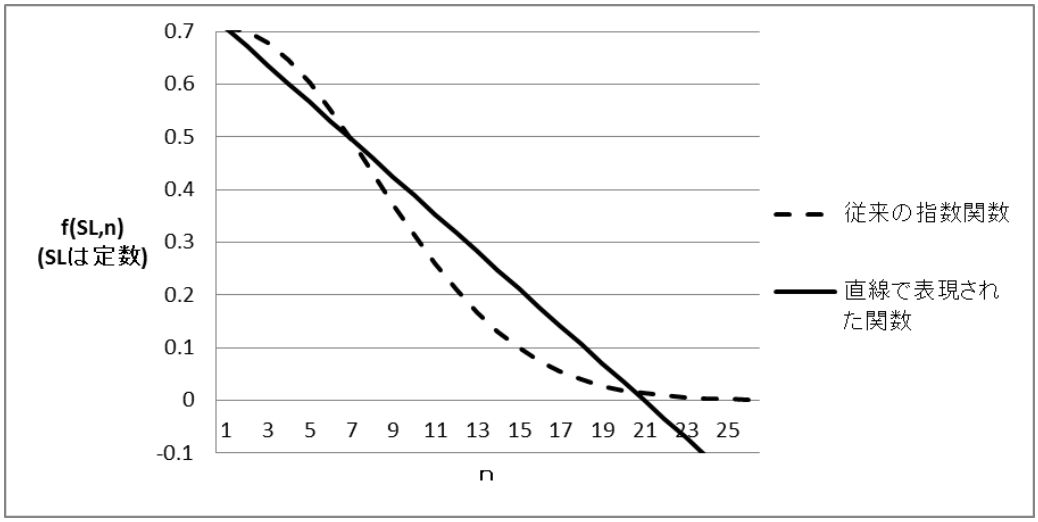

図 4.1: 従来の近傍関数と提案した近傍関数の比較

4.3 短縮と簡単化を行う手法

4.1,4.2 章において,K 点交差経路短縮法と新しい近傍関数$f(SL,n)$をそれぞれ提案した．また, この二つの手法はそれぞれ干渉しあう可能性は低いため, それぞれを同時に解法に組み込むことが可能である.

第 5 章　実験・考察

5.1　K 点交差経路短縮法を用いた実験

5.1.1　実験方法（1）

交差経路の発生において,k が大きい交差経路ほど発生しにくいことが経験上判明している. そのため,実験では既存研究である Angeniol 等の解法を改良した解法に,K= 4,5のときの K 点交差経路短縮法を行う場合と行わない場合それぞれを比較し,その効果を検証する.

実験では全都市が 1 つのノードを捕獲するまでを 1 回の計算とし,1 回の計算ごとに計算時間と経路合計を出力する. また, K 点交差経路短縮法の効果を正確に検証するため,まず K 点交差経路短縮法を行わない場合で 1 回の計算を行い,この時に都市調査を行った時と同じ順番で,K 点交差経路短縮法を行う場合で 1 回の計算を行う。そして, これらの手順を 100 回行い,それぞれの平均値を算出し,比較する.

実験ではノードと都市の座標が一致したかの判定を小数点第 5 位の精度にしている. これは, SOM-TSP 法は都市順番を求めるアルゴリズムであること, TSP に用いるのが都市の配置であることから, ある程度小さな数は無視できると判断したためである.

実験に用いる都市配置には,理論的最短経路長の分かっている「kroA100 都市問題」,「米国 532 都市問題」,「pr2392 都市問題」における都市配置を用いた. また,米国 532 都市問題の最短経路長は,定義[13]より 2 点 p,q 間のユークリッド距離 d(p,q)を求める式を式(5.1)のように変更している. また, 実験に用いた計算環境を図 5.2 に示す.

$$d(p,q) = \sqrt{\frac{p^2+q^2}{10}} \tag{5.1}$$

都市問題名	最短経路長
kroA100都市問題	21285.4
米国532都市問題	27686
pr2392都市問題	378032

図 5.1:各都市の最短経路長

計算環境	
CPU	Intel(R) Core(TM) i7-3770K CPU @ 3.50GHz 3.90 GHz
OS	Ubuntu Linux 13.10
メインメモリ	32 GB
外部メモリ	2TB HDD

図 5.2:計算環境

5.1.2　実験結果および考察（1）

実験結果を図 5.3,図 5.4 に示す．図 5.3 より,経路長合計において K 点交差経路短縮法がある場合はない場合と比べ,0.2%ほど短縮されている．また, 図 5.4 より,計算時間においてK 点交差経路短縮法がある場合はない場合と比べ,とても小さいながらも増加している．

これらのことから,実験結果より以下のことを読み取ることが出来る．

- 経路長合計において,K 点交差経路短縮法はとても小さいながらも効果が見込める．また,都市数と経路短縮効果はあまり関係なく,一定の短縮効果が見込める．

- 図 5.4 における pr2392 都市問題において,K=4 より K=5 のときのほうが計算時間が短くなっている．これは, K 点交差経路短縮法のアルゴリズム上計算時間が短縮されることはあり得ないため,実験環境による誤差であると思われる．よって, K 点交差経路短縮法による計算時間の増加は実験環境に左右されるほど小さいことが分かる．

経路長合計（平均）	K点交差経路短縮法			経路短縮率（K=4）	経路短縮率（K=5）
	なし	K=4	K=5		
karA100都市問題	22327	22286	22280	0.9982	0.9979
米国532都市問題	29855	29783	29778	0.9976	0.9974
pr2392都市問題	425060	424350	424267	0.9983	0.9981

図 5.3:K 点交差経路短縮法の結果（経路長合計）

計算時間（平均）	K点交差経路短縮法			時間短縮率（K=4）	時間短縮率（K=5）
	なし[秒]	K=4[秒]	K=5[秒]		
karA100都市問題	$8.7275 * 10^{-4}$	$8.7315 * 10^{-4}$	$8.7427 * 10^{-4}$	1.0005	1.0017
米国532都市問題	$2.1177 * 10^{-1}$	$2.1181 * 10^{-1}$	$2.1182 * 10^{-1}$	1.0002	1.0003
pr2392都市問題	3.9742	3.9745	3.9741	1.0001	1.0000

図 5.4:K 点交差経路短縮法の結果（計算時間）

5.2　新しい近傍関数を用いた実験

5.2.1　実験方法　（2）

実験は,既存研究における Angeniol 等の解法を改良した解法,新たな近傍関数$f(SL,n)$を用いた解法で行う。近傍関数$f(SL,n)$を用いる場合, αの値を-0.06 から-0.14 まで−0.02 ずつ変化させ, それぞれの場合においても実験を行う．

実験では全都市が 1 つのノードを捕獲するまでを 1 回の計算とし,1 回の計算ごとに計算時間と経路合計,全都市調査の回数を出力する．そして,これを 100 回行い,先ほどと同じくそれぞれの平均値に加え,近傍関数の効果を詳細に見るため,最大値,最少値,全都市調査回数も算出

し,比較する．また, ノードと都市の座標の一致判定は先ほどと同じく小数点以下第 5 位の精度で行い, 実験に用いる都市配置には先ほどと同じく「kroA100 都市問題」,「米国 532 都市問題」,「pr2392 都市問題」における都市配置を用いた． また,計算環境も先ほどと同じく図 5.2 のものである.

5.2.2 実験結果 (2)

実験結果を図 5.5, 図 5.6, …, 図 5.13に示す．図 5.7, 図 5.8, 図 5.11 より, 経路長合計において$f(SL,n)$を用いた解法は$f(G,n)$を用いた解法と比べ増加しており,特に図 5.10 の$\alpha = -0.14$の場合においては約 3%増加している．また, 図 5.6, 図 5.9, 図 5.12 より, 計算時間において$f(SL,n)$を用いた解法は$f(G,n)$を用いた解法と比べて短縮されており,特に図 5.5 の$\alpha = -0.14$の場合においては約 50%短縮している．また, 計算回数において図 5.6 では$\alpha = -0.08, -0.1$のとき,図 5.10, 図 5.13 では$\alpha = -0.1$のとき$f(SL,n)$を用いた解法は$f(G,n)$を用いた解法と同じくらいの計算回数で計算を行うことが出来ている.

これらのことから,実験により以下のことを読み取ることが出来る.

- 経路長合計において, $f(SL,n)$を用いた解法ではαの値が大きいほど経路長合計が小さくなる傾向にある．また, 図 5.4 より図 5.10 の経路短縮率のほうが大きいことから,都市数が多いほど経路長合計が大きくなる傾向にあることが分かる.

- 計算時間において,$f(SL,n)$を用いた解法ではαの値が小さいほど計算時間が短くなる傾向にある．また, 図 5.5 より図 5.11 の時間短縮率のほうが小さいことから,都市数が多いほど計算時間が長くなる傾向にあることが分かる.

- $f(SL,n)$を用いた解法と$f(G,n)$を用いた解法の計算回数がほぼ同じなとき,計算時間は$f(SL,n)$を用いた解法のほうが小さくなっている．このことから, 一次関数の計算時間の短さによる近傍関数の計算時間の短縮という目標は達成できたと思われる.

kroA100都市問題				
経路長合計	平均経路長	最大経路長	最少経路長	経路短縮率(平均)
f(G,n)	22052	23478	21363	
f(SL,n)(α = −0.06)	22306	23729	21486	1.0115
f(SL,n)(α = −0.08)	22530	23889	21406	1.0217
f(SL,n)(α = −0.1)	22610	24425	21493	1.0253
f(SL,n)(α = −0.12)	22654	24577	21523	1.0273
f(SL,n)(α = −0.14)	22908	25192	21718	1.0388

図 5.5: $f(G,n)$と$f(SL,n)$をそれぞれ用いた解法の比較(kroA100, 経路長合計)

kroA100都市問題				
計算時間	平均時間[ミリ秒]	最大時間[ミリ秒]	最少時間[ミリ秒]	時間短縮率（平均）
f(G,n)	8.6719	9.7000	8.2310	
f(SL,n)(α = −0.06)	5.9084	6.3240	5.7620	0.6813
f(SL,n)(α = −0.08)	5.0386	5.4840	4.9290	0.5810
f(SL,n)(α = −0.1)	4.5962	5.0480	4.4770	0.5300
f(SL,n)(α = −0.12)	4.2532	4.7220	4.1420	0.4905
f(SL,n)(α = −0.14)	4.0386	4.3530	3.9070	0.4657

図 5.6: $f(G,n)$と$f(SL,n)$をそれぞれ用いた解法の比較（kroA100, 計算時間）

kroA100都市問題			
計算回数	平均[回]	最大[回]	最少[回]
f(G,n)	28	31	27
f(SL,n)(α = −0.06)	33	32	32
f(SL,n)(α = −0.08)	29	28	28
f(SL,n)(α = −0.1)	27	26	26
f(SL,n)(α = −0.12)	26	25	25
f(SL,n)(α = −0.14)	25	24	24

図 5.7: $f(G,n)$と$f(SL,n)$をそれぞれ用いた解法の比較（kroA100, 計算回数）

米国532都市問題				
経路長合計	平均経路長	最大経路長	最少経路長	経路短縮率(平均)
f(G,n)	29795	30924	29016	
f(SL,n)(α = −0.06)	29796	30822	28967	1.0000
f(SL,n)(α = −0.08)	29994	30812	29303	1.0067
f(SL,n)(α = −0.1)	30131	31062	29405	1.0113
f(SL,n)(α = −0.12)	30382	31701	29581	1.0197
f(SL,n)(α = −0.14)	30514	31273	29786	1.0241

図 5.8: $f(G,n)$と$f(SL,n)$をそれぞれ用いた解法の比較（att532, 経路長合計）

米国532都市問題				
計算時間	平均時間[秒]	最大時間[秒]	最少時間[秒]	時間短縮率(平均)
f(G,n)	0.2120	0.2561	0.2093	
f(SL,n)(α = −0.06)	0.1574	0.2056	0.1561	0.7425
f(SL,n)(α = −0.08)	0.1335	0.1349	0.1325	0.6296
f(SL,n)(α = −0.1)	0.1204	0.1225	0.1193	0.5678
f(SL,n)(α = −0.12)	0.1125	0.1178	0.1117	0.5308
f(SL,n)(α = −0.14)	0.1061	0.1079	0.1053	0.5003

図 5.9: $f(G,n)$と$f(SL,n)$をそれぞれ用いた解法の比較(att532, 計算時間)

米国532都市問題			
計算回数	平均[回]	最大[回]	最少[回]
f(G,n)	30	30	29
f(SL,n)(α = −0.06)	35	35	35
f(SL,n)(α = −0.08)	32	32	31
f(SL,n)(α = −0.1)	30	30	29
f(SL,n)(α = −0.12)	28	29	28
f(SL,n)(α = −0.14)	28	28	27

図 5.10: $f(G,n)$と$f(SL,n)$をそれぞれ用いた解法の比較(att532, 計算回数)

pr2392都市問題				
経路長合計	平均経路長	最大経路長	最少経路長	経路短縮率(平均)
f(G,n)	424600	432616	418203	
f(SL,n)(α = −0.06)	426526	434707	418566	1.0045
f(SL,n)(α = −0.08)	428301	437422	421696	1.0087
f(SL,n)(α = −0.1)	430526	438680	423041	1.0140
f(SL,n)(α = −0.12)	433482	441050	427151	1.0209
f(SL,n)(α = −0.14)	436299	442149	428645	1.0276

図 5.11: $f(G,n)$と$f(SL,n)$をそれぞれ用いた解法の比較(pr2392, 経路長合計)

pr2392都市問題				
計算時間	平均時間[秒]	最大時間[秒]	最少時間[秒]	時間短縮率(平均)
f(G,n)	3.9770	4.0039	3.9595	
f(SL,n)(α = −0.06)	3.1032	3.1166	3.0881	0.7803
f(SL,n)(α = −0.08)	2.6604	2.6752	2.6526	0.6689
f(SL,n)(α = −0.1)	2.3885	2.3999	2.3776	0.6006
f(SL,n)(α = −0.12)	2.2151	2.2279	2.2051	0.5570
f(SL,n)(α = −0.14)	2.0951	2.1037	2.0875	0.5268

図 5.12: $f(G,n)$と$f(SL,n)$をそれぞれ用いた解法の比較(pr2392, 計算時間)

pr2392都市問題			
計算回数	平均[回]	最大[回]	最少[回]
f(G,n)	32	32	31
f(SL,n)(α = −0.06)	37	38	37
f(SL,n)(α = −0.08)	34	34	34
f(SL,n)(α = −0.1)	32	32	32
f(SL,n)(α = −0.12)	31	31	30
f(SL,n)(α = −0.14)	30	30	29

図 5.13: $f(G,n)$と$f(SL,n)$をそれぞれ用いた解法の比較(pr2392, 計算回数)

5.3 短縮と簡単化を行う実験

5.3.1 実験方法 (3)

実験は,既存研究における Angeniol 等の解法を改良した解法,新たな近傍関数$f(SL,n)$を用いた解法で行う. 先ほどと同じく,近傍関数$f(SL,n)$を用いる場合, αの値を-0.06から-0.14まで-0.02ずつ変化させ, それぞれの場合においても実験を行う. また, K 点交差経路短縮法を行う場合においても実験を行う. この時, K=4,5 である.

実験では全都市が 1 つのノードを捕獲するまでを 1 回の計算とし,1 回の計算ごとに計算時間と経路合計を出力する. そして,これを 100 回行い,それぞれの平均値,最大値,最少値を算出し,比較する. また, ノードと都市の座標の一致判定は先ほどと同じく小数点以下第 5 位の精度で行い, 実験に用いる都市配置には先ほどと同じく「kroA100 都市問題」,「米国 532 都市問題」,「pr2392 都市問題」における都市配置を用いた. また,計算環境も先ほどと同じく図 4.2 のものである.

5.3.2 実験結果および考察 (3)

実験結果を図5.14, 図 5.15, …, 図 5.31に示す．実験により以下のことを読み取ることが出来る．

- 全ての図における経路短縮率,時間短縮率において,K 点交差経路短縮法を用いた場合のほうが値が 0.005 程度大きいまたは小さい場合が存在する．これは先ほどの交差経路実験では比較の際に都市調査の順番を固定していたのに対し, この実験では固定していないためである．このことから, K 点交差経路短縮法の効果は安定しているものの, 誤差の範囲程度の効果しかなく, 大きな効果は望めないと思われる．

- 図 5.16,図 5.22 などの経路短縮率を見てみると, αの値が小さいほど K 点交差経路短縮法を用いた場合の短縮率は小さくなっていることが分かる．このことから, $f(SL,n)$を用いた解法においてαの値が小さいほど K 点交差経路短縮法の効果が大きいと考えられるため, $f(SL,n)$と K 点交差経路短縮法の相性は良いと思われる．

- 経路長合計において, 図 5.14, 図 5.20, 図 5.26 におけるα = −0.14のとき,K 点交差経路短縮法を行わない場合よりも K=5 の場合のほうが短縮されている．また, 図 5.17, 図 5.23, 図 5.29 において K 点交差経路短縮法を行わない場合よりも行う場合のほうが,計算時間が増加している場合があるものの最大で約 0.2%しか増加しておらず,都市数が大きくなるほど計算時間への影響は小さくなっていることが分かる．このことから, 提案手法をそれぞれ別に使用するよりも, 組み合わせて使用することで更なる効果が見込める．

kroA100都市問題						
経路長合計（平均）	K点交差経路短縮法			経路短縮率（なし）	経路短縮率（K=4）	経路短縮率（K=5）
	なし	K=4	K=5			
f(G,n)	22052	22038	22110		0.9994	1.0026
f(SL,n)(α = −0.06)	22306	22225	22249	1.0115	1.0079	1.0089
f(SL,n)(α = −0.08)	22530	22519	22408	1.0217	1.0212	1.0161
f(SL,n)(α = −0.1)	22610	22555	22420	1.0253	1.0228	1.0167
f(SL,n)(α = −0.12)	22654	22620	22710	1.0273	1.0257	1.0299
f(SL,n)(α = −0.14)	22908	22674	22672	1.0388	1.0282	1.0281

図 5.14: $f(G,n)$と$f(SL,n)$をそれぞれ用いた解法の比較（kroA100, 経路長合計, 平均）

kroA100都市問題						
経路長合計（最大）	K点交差経路短縮法			経路短縮率（なし）	経路短縮率（K=4）	経路短縮率（K=5）
	なし	K=4	K=5			
f(G,n)	23478	23482	23488		1.0002	1.0004
f(SL,n)(α = −0.06)	23729	23814	24217	1.0107	1.0143	1.0315
f(SL,n)(α = −0.08)	23889	23852	24481	1.0175	1.0159	1.0427
f(SL,n)(α = −0.1)	24425	24593	23861	1.0403	1.0475	1.0163
f(SL,n)(α = −0.12)	24577	24369	24656	1.0468	1.0379	1.0502
f(SL,n)(α = −0.14)	25192	24801	24358	1.0730	1.0563	1.0375

図 5.15: $f(G,n)$と$f(SL,n)$をそれぞれ用いた解法の比較（kroA100, 経路長合計, 最大）

kroA100都市問題						
経路長合計（最少）	K点交差経路短縮法			経路短縮率（なし）	経路短縮率（K=4）	経路短縮率（K=5）
	なし	K=4	K=5			
f(G,n)	21363	21368	21328		1.0002	0.9984
f(SL,n)(α = −0.06)	21486	21396	21503	1.0057	1.0015	1.0065
f(SL,n)(α = −0.08)	21406	21603	21516	1.0020	1.0112	1.0072
f(SL,n)(α = −0.1)	21493	21573	21610	1.0061	1.0098	1.0116
f(SL,n)(α = −0.12)	21523	21395	21471	1.0075	1.0015	1.0050
f(SL,n)(α = −0.14)	21718	21680	21790	1.0166	1.0148	1.0200

図 5.16: $f(G,n)$と$f(SL,n)$をそれぞれ用いた解法の比較（kroA100, 経路長合計, 最少）

kroA100都市問題						
計算時間(平均)	K点交差経路短縮法			時間短縮率（なし）	時間短縮率（K=4）	時間短縮率（K=5）
	なし[ミリ秒]	K=4[ミリ秒]	K=5[ミリ秒]			
f(G,n)	8.6719	8.7620	8.6464		1.0104	0.9868
f(SL,n)(α = −0.06)	5.9084	5.9247	6.0050	0.6813	0.6832	0.6925
f(SL,n)(α = −0.08)	5.0386	5.0565	5.0584	0.5810	0.5831	0.5833
f(SL,n)(α = −0.1)	4.5962	4.6137	4.6060	0.5300	0.5320	0.5311
f(SL,n)(α = −0.12)	4.2532	4.2691	4.2833	0.4905	0.4923	0.4939
f(SL,n)(α = −0.14)	4.0386	4.0497	4.0481	0.4657	0.4670	0.4668

図 5.17: $f(G,n)$と$f(SL,n)$をそれぞれ用いた解法の比較（kroA100, 計算時間, 平均）

kroA100都市問題						
計算時間（最大）	K点交差経路短縮法			時間短縮率（なし）	時間短縮率（K=4）	時間短縮率（K=5）
	なし[ミリ秒]	K=4[ミリ秒]	K=5[ミリ秒]			
f(G,n)	9.7000	10.4080	9.7030		1.0730	1.0003
f(SL,n)(α = −0.06)	6.3240	6.4810	6.4950	0.6520	0.6681	0.6696
f(SL,n)(α = −0.08)	5.4840	5.5320	5.4450	0.5654	0.5703	0.5613
f(SL,n)(α = −0.1)	5.0480	5.1210	5.0800	0.5204	0.5279	0.5237
f(SL,n)(α = −0.12)	4.7220	4.6810	4.7230	0.4868	0.4826	0.4869
f(SL,n)(α = −0.14)	4.3530	4.5080	4.4420	0.4488	0.4647	0.4579

図 5.18: $f(G,n)$と$f(SL,n)$をそれぞれ用いた解法の比較（kroA100, 計算時間, 最大）

kroA100都市問題						
計算時間（最少）	K点交差経路短縮法			時間短縮率（なし）	時間短縮率（K=4）	時間短縮率（K=5）
	なし[ミリ秒]	K=4[ミリ秒]	K=5[ミリ秒]			
f(G,n)	8.2310	8.0710	8.2990		0.9806	1.0083
f(SL,n)(α = −0.06)	5.7620	5.7530	5.8380	0.7000	0.6989	0.7093
f(SL,n)(α = −0.08)	4.9290	4.9460	4.9580	0.5988	0.6009	0.6024
f(SL,n)(α = −0.1)	4.4770	4.4910	4.4790	0.5439	0.5456	0.5442
f(SL,n)(α = −0.12)	4.1420	4.1520	4.1460	0.5032	0.5044	0.5037
f(SL,n)(α = −0.14)	3.9070	3.9040	3.9300	0.4747	0.4743	0.4775

図 5.19: $f(G,n)$と$f(SL,n)$をそれぞれ用いた解法の比較（kroA100, 計算時間, 最少）

米国532都市問題						
経路長合計（平均）	K点交差経路短縮法			経路短縮率（なし）	経路短縮率（K=4）	経路短縮率（K=5）
	なし	K=4	K=5			
f(G,n)	29795	29758	29707		0.9988	0.9971
f(SL,n)(α = −0.06)	29796	29762	29750	1.0000	0.9989	0.9985
f(SL,n)(α = −0.08)	29994	29879	29883	1.0067	1.0028	1.0030
f(SL,n)(α = −0.1)	30131	29983	30072	1.0113	1.0063	1.0093
f(SL,n)(α = −0.12)	30382	30141	30106	1.0197	1.0116	1.0105
f(SL,n)(α = −0.14)	30514	30258	30269	1.0241	1.0155	1.0159

図 5.20: $f(G,n)$と$f(SL,n)$をそれぞれ用いた解法の比較（att532, 経路長合計, 平均）

米国532都市問題						
経路長合計（最大）	K点交差経路短縮法			経路短縮率（なし）	経路短縮率（K=4）	経路短縮率（K=5）
	なし	K=4	K=5			
f(G,n)	30924	30289	30440		0.9794	0.9843
f(SL,n)(α = −0.06)	30822	30479	30702	0.9967	0.9856	0.9928
f(SL,n)(α = −0.08)	30812	30614	30720	0.9964	0.9900	0.9934
f(SL,n)(α = −0.1)	31062	30808	30865	1.0044	0.9962	0.9981
f(SL,n)(α = −0.12)	31701	30995	30852	1.0251	1.0023	0.9977
f(SL,n)(α = −0.14)	31273	30947	31243	1.0113	1.0007	1.0103

図 5.21: $f(G,n)$と$f(SL,n)$をそれぞれ用いた解法の比較（att532, 経路長合計, 最大）

米国532都市問題						
経路長合計（最少）	K点交差経路短縮法			経路短縮率（なし）	経路短縮率（K=4）	経路短縮率（K=5）
	なし	K=4	K=5			
f(G,n)	29016	29193	28839		1.0061	0.9939
f(SL,n)(α = −0.06)	28967	29100	28980	0.9983	1.0029	0.9988
f(SL,n)(α = −0.08)	29303	29103	29090	1.0099	1.0030	1.0026
f(SL,n)(α = −0.1)	29405	29300	29296	1.0134	1.0098	1.0097
f(SL,n)(α = −0.12)	29581	29189	29316	1.0195	1.0060	1.0103
f(SL,n)(α = −0.14)	29786	29482	29616	1.0265	1.0160	1.0207

図 5.22: $f(G,n)$と$f(SL,n)$をそれぞれ用いた解法の比較（att532, 経路長合計, 最少）

米国532都市問題						
計算時間(平均)	K点交差経路短縮法			時間短縮率（なし）	時間短縮率（K=4）	時間短縮率（K=5）
	なし[秒]	K=4[秒]	K=5[秒]			
f(G,n)	0.2120	0.2133	0.2118		1.0059	0.9989
f(SL,n)(α = −0.06)	0.1574	0.1571	0.1573	0.7425	0.7408	0.7420
f(SL,n)(α = −0.08)	0.1335	0.1338	0.1335	0.6296	0.6308	0.6294
f(SL,n)(α = −0.1)	0.1204	0.1202	0.1209	0.5678	0.5669	0.5701
f(SL,n)(α = −0.12)	0.1125	0.1122	0.1123	0.5308	0.5291	0.5298
f(SL,n)(α = −0.14)	0.1061	0.1064	0.1062	0.5003	0.5017	0.5008

図 5.23: $f(G,n)$と$f(SL,n)$をそれぞれ用いた解法の比較（att532, 計算時間, 平均）

米国532都市問題						
計算時間（最大）	K点交差経路短縮法			時間短縮率（なし）	時間短縮率（K=4）	時間短縮率（K=5）
	なし[秒]	K=4[秒]	K=5[秒]			
f(G,n)	0.2561	0.2210	0.2166		0.8630	0.8458
f(SL,n)(α = −0.06)	0.2056	0.1586	0.1589	0.8027	0.6191	0.6202
f(SL,n)(α = −0.08)	0.1349	0.1355	0.1360	0.5267	0.5289	0.5309
f(SL,n)(α = −0.1)	0.1225	0.1215	0.1220	0.4782	0.4744	0.4763
f(SL,n)(α = −0.12)	0.1178	0.1142	0.1135	0.4601	0.4457	0.4430
f(SL,n)(α = −0.14)	0.1079	0.1083	0.1075	0.4213	0.4228	0.4198

図 5.24: $f(G,n)$と$f(SL,n)$をそれぞれ用いた解法の比較（att532, 計算時間, 最大）

米国532都市問題						
計算時間（最少）	K点交差経路短縮法			時間短縮率（なし）	時間短縮率（K=4）	時間短縮率（K=5）
	なし[秒]	K=4[秒]	K=5[秒]			
f(G,n)	0.2093	0.2104	0.2098		1.0054	1.0024
f(SL,n)(α = −0.06)	0.1561	0.1560	0.1563	0.7456	0.7452	0.7469
f(SL,n)(α = −0.08)	0.1325	0.1327	0.1325	0.6331	0.6341	0.6329
f(SL,n)(α = −0.1)	0.1193	0.1192	0.1199	0.5699	0.5696	0.5727
f(SL,n)(α = −0.12)	0.1117	0.1110	0.1112	0.5337	0.5305	0.5313
f(SL,n)(α = −0.14)	0.1053	0.1053	0.1050	0.5031	0.5033	0.5016

図 5.25: $f(G,n)$と$f(SL,n)$をそれぞれ用いた解法の比較（att532, 計算時間, 最少）

pr2392都市問題						
経路長合計（平均）	K点交差経路短縮法			経路短縮率（なし）	経路短縮率（K=4）	経路短縮率（K=5）
	なし	K=4	K=5			
f(G,n)	424600	423988	423605		0.9986	0.9977
f(SL,n)(α = −0.06)	426526	425655	425432	1.0045	1.0025	1.0020
f(SL,n)(α = −0.08)	428301	427455	427233	1.0087	1.0067	1.0062
f(SL,n)(α = −0.1)	430526	429658	429397	1.0140	1.0119	1.0113
f(SL,n)(α = −0.12)	433482	431922	431874	1.0209	1.0172	1.0171
f(SL,n)(α = −0.14)	436299	433430	433117	1.0276	1.0208	1.0201

図 5.26: $f(G,n)$と$f(SL,n)$をそれぞれ用いた解法の比較（pr2392, 経路長合計, 平均）

pr2392都市問題						
経路長合計（最大）	K点交差経路短縮法			経路短縮率（なし）	経路短縮率（K=4）	経路短縮率（K=5）
	なし	K=4	K=5			
f(G,n)	432616	431080	431210		0.9965	0.9967
f(SL,n)(α = −0.06)	434707	431283	434104	1.0048	0.9969	1.0034
f(SL,n)(α = −0.08)	437422	433494	434822	1.0111	1.0020	1.0051
f(SL,n)(α = −0.1)	438680	434906	437752	1.0140	1.0053	1.0119
f(SL,n)(α = −0.12)	441050	439473	437988	1.0195	1.0158	1.0124
f(SL,n)(α = −0.14)	442149	441143	439641	1.0220	1.0197	1.0162

図 5.27: $f(G,n)$と$f(SL,n)$をそれぞれ用いた解法の比較（pr2392, 経路長合計, 最大）

pr2392都市問題						
経路長合計（最少）	K点交差経路短縮法			経路短縮率（なし）	経路短縮率（K=4）	経路短縮率（K=5）
	なし	K=4	K=5			
f(G,n)	418203	416932	416735		0.9970	0.9965
f(SL,n)(α = −0.06)	418566	417420	419959	1.0009	0.9981	1.0042
f(SL,n)(α = −0.08)	421696	421665	420366	1.0084	1.0083	1.0052
f(SL,n)(α = −0.1)	423041	420626	420466	1.0116	1.0058	1.0054
f(SL,n)(α = −0.12)	427151	421893	424866	1.0214	1.0088	1.0159
f(SL,n)(α = −0.14)	428645	425389	425661	1.0250	1.0172	1.0178

図 5.28: $f(G,n)$と$f(SL,n)$をそれぞれ用いた解法の比較（pr2392, 経路長合計, 最少）

pr2392都市問題						
計算時間(平均)	K点交差経路短縮法			時間短縮率（なし）	時間短縮率（K=4）	時間短縮率（K=5）
	なし[秒]	K=4[秒]	K=5[秒]			
f(G,n)	3.9770	3.9870	3.9752		1.0025	0.9996
f(SL,n)(α = −0.06)	3.1032	3.1101	3.1052	0.7803	0.7820	0.7808
f(SL,n)(α = −0.08)	2.6604	2.6606	2.6620	0.6689	0.6690	0.6693
f(SL,n)(α = −0.1)	2.3885	2.3899	2.3896	0.6006	0.6009	0.6009
f(SL,n)(α = −0.12)	2.2151	2.2197	2.2168	0.5570	0.5581	0.5574
f(SL,n)(α = −0.14)	2.0951	2.0953	2.0955	0.5268	0.5269	0.5269

図 5.29: $f(G,n)$と$f(SL,n)$をそれぞれ用いた解法の比較（pr2392, 計算時間, 平均）

pr2392都市問題						
計算時間（最大）	K点交差経路短縮法			時間短縮率（なし）	時間短縮率（K=4）	時間短縮率（K=5）
	なし[秒]	K=4[秒]	K=5[秒]			
f(G,n)	4.0039	4.0432	4.0034		1.0098	0.9999
f(SL,n)(α = −0.06)	3.1166	3.1456	3.1185	0.7784	0.7856	0.7789
f(SL,n)(α = −0.08)	2.6752	2.6874	2.6721	0.6681	0.6712	0.6674
f(SL,n)(α = −0.1)	2.3999	2.5857	2.4039	0.5994	0.6458	0.6004
f(SL,n)(α = −0.12)	2.2279	2.2287	2.2272	0.5564	0.5566	0.5562
f(SL,n)(α = −0.14)	2.1037	2.1049	2.1670	0.5254	0.5257	0.5412

図 5.30: $f(G,n)$と$f(SL,n)$をそれぞれ用いた解法の比較（pr2392, 計算時間, 最大）

pr2392都市問題						
計算時間（最少）	K点交差経路短縮法			時間短縮率（なし）	時間短縮率（K=4）	時間短縮率（K=5）
	なし[秒]	K=4[秒]	K=5[秒]			
f(G,n)	3.9595	3.9599	3.9590		1.0001	0.9999
f(SL,n)(α = −0.06)	3.0881	3.0966	3.0919	0.7799	0.7821	0.7809
f(SL,n)(α = −0.08)	2.6526	2.6501	2.6494	0.6699	0.6693	0.6691
f(SL,n)(α = −0.1)	2.3776	2.3778	2.3776	0.6005	0.6005	0.6005
f(SL,n)(α = −0.12)	2.2051	2.2109	2.2071	0.5569	0.5584	0.5574
f(SL,n)(α = −0.14)	2.0875	2.0847	2.0873	0.5272	0.5265	0.5272

図 5.31: $f(G,n)$と$f(SL,n)$をそれぞれ用いた解法の比較（pr2392, 計算時間, 最少）

第 6 章　おわりに

本研究では, 既存研究における Angeniol 等の解法を改良した改良 SOM-TSP 法を元に, 交差経路の発見し補正する K 点交差経路短縮法, 近傍関数の計算に一次関数を用いることで計算時間を短縮する新しい近傍関数$f(SL,n)$をそれぞれ提案した．実験結果より, K 点交差経路短縮法は短縮率は小さいながらも, さまざまな都市数において経路短縮効果が見込めることが分かる．また, 新しい近傍関数$f(SL,n)$を用いた解法では一次関数を用いることによる効果が確認でき, 経路長合計においては約 1%と小さいながらも増加する傾向にあったものの, 計算時間において約 50%の短縮という大きな効果が見込める．

今後の研究では, K 点交差経路短縮法において K の値を増やした場合, また都市数を 10000 程度に増やした場合においての計算時間への影響がどれほどなのか実験してみる必要があると思われる．新しい近傍関数$f(SL,n)$のパラメータ SL において, 今回パラメータ SL の更新式は$f(G,n)$のパラメータ G の更新式を改変したものを用いているが, この改変の際に都市数の増減による近傍領域の変化の機能を変更している．しかし,この機能自体は有用なものであるため,これを組み込んだ更新式を提案することで経路長合計の短縮も行うことが出来る可能性は十分に存在すると思われる．

謝辞

本研究を進めるにあたり,様々なご指導を頂きました三好力教授に深く感謝いたします．また発表を通じて多くの示唆を頂いた三好研究室の皆様に感謝します.

参考文献

1) 巡回セールスマン問題とは
http://e-words.jp/w/E5B7A1E59B9EE382BBE383BCE383ABE382B9E3839EE383B3E5958FE9A18C.html
2) 山本芳嗣, 久保幹雄
巡回セールスマン問題への招待
（朝倉書店,1997）
3) Traveling Salesman Problem
http://www.math.uwaterloo.ca/tsp/index.html
4) ウィリアム・J・クック,松浦俊輔（翻訳）
驚きの数学:巡回セールスマン問題
（青土社,2013）
5) 巡回セールスマン問題（交叉を用いた解法）
http://www.info.kanagawa-u.ac.jp/~hatori/study/yousi2004-5.htm
6) ホップフィールド・ネットワーク
http://www.sist.ac.jp/~kanakubo/research/neuro/hopfieldnetwork.html
7) ボルツマンマシンによる巡回セールスマン問題
http://www.info.kanagawa-u.ac.jp/~hatori/study/yousi2003-2.htm
8) もっと簡単にー線分交差判定ー
http://www5d.biglobe.ne.jp/~tomoya03/shtml/algorithm/Intersection.htm
9) 大北 正昭 (監修),T.コホネン (原著),徳高 平蔵 堀尾 恵一 大藪 又茂 藤村 喜久郎
自己組織化マップ 改訂版
（丸善出版, 改訂版, 2012)
10) 藤村喜久郎,徳高平蔵,石川眞澄
他都市巡回セールスマン問題での改良 SOM-TSP 法の性能評価
（電気学会論文誌 C, Vol.119-C, No.7, pp-875-882, 1999 年）
11) 藤村喜久郎,徳高平蔵,大島靖広,田中慎一,岸田悟
Kohonen 自己組織化特徴マップを用いた巡回セールスマン問題解法の改良
（電気学会論文誌 C, Vol.6-C(3), pp-350-358, 1996 年）
12) 徳高平蔵, 大北正昭, 藤村喜久郎
自己組織化マップとその応用
（シュプリンガー・ジャパン株式会社, 2007）
13) M.Padberg, G.Rinaldi
Optimization of a 532-city symmetric traveling salesman problem by branch and cut
（Operations Research Letters archive, Vol.6, pp.1-7, 1987）

付録

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

#define MAX_NODE 40000
#define MAX_POINT 30000
#define MAX_TOSI 20000

struct node_point{
        int j;              //ノードの番号
        double x;           //x 座標
        double y;           //y 座標
        int nc;
        //0:1 度も勝者ノードに選ばれない
         //1:1 度勝者ノードに選ばれる
         //2:1 度以上勝者ノードに選ばれる
        int nd;
        //勝者ノードに選ばれなければ
        //1 加算し、3 になったら削除
        int nk;             //0:捕獲されていない
                            //1:捕獲されている
};

struct node_point node[MAX_NODE], n;

struct tosi_point{
        double x;
        double y;
        int h;      //0:ノードを捕獲していない
                    //1:ノードを捕獲した
};
struct tosi_point tosi[MAX_TOSI];

int n_end;                  //ノードの個数を記録
int nb_end;                 //前回の全都市調査
                            //のノード数を記録
int t_end;                  //都市数
int h_tosi = 0;             //捕獲したノードの数
double l2 = 1.0 / sqrt(2.0);    //計算の短縮
double al ;                 //計算の短縮

int tekiou_check(int i);        //i:都市の配列番号
void node_kousin(int jc, int i, double sl);
    //jc:ノードの配列番号, i:都市の配列番号,
        // sl:パラメータ
void node_dreate(int jc);
        //jc:削除ノードの配列番号
void node_create(int jc);
        //jc:ノードの配列番号
void kousa_syori(int a, int b, int c, int d);
void kousa(void);

////////////////////////////////////////////////////  適応度調査
int tekiou_check(int i){
        double v, vc;
        int j = 0, jc;

        vc = (((node[j].x - tosi[i].x) *
         (node[j].x - tosi[i].x)) +
         ((node[j].y - tosi[i].y) *
         (node[j].y - tosi[i].y))) + 1.0;
        for (j = 0; j < n_end; j++){
                v = ((node[j].x - tosi[i].x) *
                  (node[j].x - tosi[i].x))
                  + ((node[j].y - tosi[i].y) *
                  (node[j].y - tosi[i].y));
                if (vc > v){
                        vc = v;
                        jc = j;
                }
        }
        return jc;
}

////////////////////////////////////////////////////  更新処理
void node_kousin(int jc, int i, double sl){
        int j;
        double n, f, tn;

        for (j = 0; j < n_end; j++){

          if (node[j].nk == 1) continue;

         if (node[j].j >= node[jc].j){
          n = node[j].j - node[jc].j;
           if (n > n_end - node[j].j + node[jc].j) {
            n = n_end - node[j].j + node[jc].j;
           }
         }else {
```

```
            n = node[jc].j - node[j].j;
            if (n > n_end - node[jc].j + node[j].j){
              n = n_end - node[jc].j + node[j].j
            }
           }

           tn = (sl * n) + 1.0;
           if (tn < 0) tn = 0.0;
           f = l2 * tn;

           node[j].x += (f * (tosi[i].x - node[j].x));
           node[j].y += (f * (tosi[i].y - node[j].y));
          }
}

////////////////////////////////////////////////////  ノードの削除
void node_dreate(int jc){
        int j;

        for (j = jc; j < n_end - 1; j++){
                node[j] = node[j + 1];
                node[j].j--;
        }

        n_end--;
}

////////////////////////////////////////////////////  ノードの生成
void node_create(int jc){
        int j;

        for (j = n_end; j > jc + 1; j--){
                node[j] = node[j - 1];
                node[j].j++;
        }

        node[jc + 1] = node[jc];
        node[jc + 1].nd = 0;
        node[jc + 1].nc = 0;
        node[jc + 1].nk = 0;
        node[jc + 1].j++;
        n_end++;
}


////////////////////////////////////////////////  交差判定＆修正
void kousa_syori(int a, int b, int c, int d){
int i;

if (node[a].x > node[b].x){
        if (((node[a].x < node[c].x) &&
        (node[a].x < node[d].x)) ||
         ((node[b].x > node[c].x) &&
         (node[b].x > node[d].x)))return;
}else{
        if (((node[b].x < node[c].x) &&
         (node[b].x < node[d].x)) ||
        ((node[a].x > node[c].x) &&
         (node[a].x > node[d].x)))return;
}
if (node[a].y > node[b].y){
        if (((node[a].y < node[c].y) &&
        (node[a].y < node[d].y)) ||
        ((node[b].y > node[c].y) &&
         (node[b].y > node[d].y)))return;
}else{
        if (((node[b].y < node[c].y) &&
         (node[b].y < node[d].y)) ||
         ((node[a].y > node[c].y) &&
        (node[a].y > node[d].y)))return;
}
if ((((node[a].x - node[b].x)*
(node[c].y - node[a].y)) +
((node[a].y - node[b].y)*
(node[a].x - node[c].x))) *
(((node[a].x - node[b].x)*
(node[d].y - node[a].y)) +
 ((node[a].y - node[b].y)*
(node[a].x - node[d].x))) < 0){
      if ((((node[c].x - node[d].x)*
       (node[a].y - node[c].y)) +
       ((node[c].y - node[d].y)*
       (node[c].x - node[a].x))) *
      (((node[c].x - node[d].x)*
       (node[b].y - node[c].y)) +
      ((node[c].y - node[d].y)*
       (node[c].x - node[b].x))) < 0){
            for (i = 0; b + i > c - i; i++){
               n = node[b + i];
               node[b + i] = node[c - i];
               node[c - i] = n;
            }
       }
}
```

```
}

void kousa(void){
        int i, a, b, c, d, e;

        for (i = 0; i < n_end; i++){
         a = i; b = i + 1; c = i + 2; d = i + 3; e = i + 4;
         if (b >= n_end)b -= n_end;
         if (c >= n_end)c -= n_end;
         if (d >= n_end)d -= n_end;
         if (e >= n_end)e -= n_end;
         kousa_syori(a, b, c, d);
         kousa_syori(a, b, d, e);
        }
}

////////////////////////////////////////////////////////////////// メイン
int main(int argc, char **argv){
        FILE *fp;
        int i, j, jc, rc = 0;
        double sl = -0.05;
        double tmpx, tmpy;

////////////////////////////////////// ノードの初期化
        srand((unsigned)time(NULL));
        node[0].j = 1;
        node[0].nc = 0;
        node[0].nd = 0;
        node[0].nk = 0;
        node[0].y = rand() % MAX_POINT;
        node[0].x = rand() % MAX_POINT;

        n_end = 1;

////////// 都市座標の読み取り＆順番入れ替え
        if ((fp = fopen("tosi.txt", "r")) == NULL) {
                printf("file open error!!\n");
                exit(EXIT_FAILURE);
        }

        i = 0;
while ((fscanf_s(fp, "%lf %lf", &tosi[i].x, &tosi[i].y))
        !=EOF){
                tosi[i].h = 0;
                i++;
        }
        t_end = i;

        al = -0.1 / (double)t_end;    //計算の短縮

        for (i = 0; i < t_end; i++){
                j = rand() % (t_end - 1);
                tmpx = tosi[j].x;
                tmpy = tosi[j].y;
                tosi[j].x = tosi[i].x;
                tosi[j].y = tosi[i].y;
                tosi[i].x = tmpx;
                tosi[i].y = tmpy;
        }

        while (1){

//////////////////////////////////// 適応度調査＆更新処理
        for (i = 0; i < t_end; i++){
        if (tosi[i].h == 1)continue;

        jc = tekiou_check(i);
        node[jc].nd = 0;

        if (node[jc].nc == 0)node[jc].nc++;
        else if (node[jc].nc == 1)node[jc].nc++;

        node_kousin(jc, i, sl);

        if (fabs(tosi[i].x - node[jc].x) < 0.00001 &&
        fabs(tosi[i].y - node[jc].y) < 0.00001){
                tosi[i].h = 1;
                node[jc].nk = 1;
                h_tosi++;
        }
        }

////////////////////////////////////////// ノードの複製＆削除
        for (j = 0; j < n_end; j++){
                if (node[j].nk != 1){
                        if (node[j].nc == 0) {
                        node[j].nd++;
                        }
                if (node[j].nd == 3){
                        node_dreate(j--);
                        continue;
                }
        }
        if (node[j].nc == 2) node_create(j);
        }
```

```
////////////////////////////////////////////////////  パラメータ更新
        nb_end = n_end;

////////////////////////////////////////////////////  捕獲チェック
        for (i = 0; i < t_end; i++){
                if (tosi[i].h != 1) break;
                if (i == t_end - 1) rc = 1;
        }
        if (rc == 1)break;

////////////////////////////////////////////////////  一部初期化
        for (i = 0; i < n_end; i++){
                node[i].nc = 0;
        }
        }

////////////////////////////////////////////////////  終了処理
        kousa();
        fclose(fp);
        return 0;
}
```