粒子群最適化に基づく巡回セールスマン問題に対する近似解法に関する研究

北海道大学大学院情報科学研究科

本庄 将也

概要

近年注目されている実数値最適化手法の一つに粒子群最適化（Particle Swarm Optimization, PSO）がある．PSO は群知能の一種であり，複数の探索単位 (粒子) が互いに情報共有を行いながら解の探索を行う．多点探索を行うメタヒューリスティクスとしては遺伝的アルゴリズム（Genetic Algorithm, GA）が有名であるが，多くの実数値最適化問題において PSO のほうが GA に比べて高速に良い解を発見できることが知られている．

本研究では，組合せ最適化問題の一種である巡回セールスマン問題（Traveling Salesman Problem, TSP）に対して短時間で良い解を得ることを目的として，PSO を基にしたアルゴリズムである挿入操作 PSO 戦略を提案する．提案手法では，粒子の解候補は実数値ベクトルではなく巡回路として表現され，粒子間の相互作用は部分経路挿入 によって行われる．本論文では，挿入操作 PSO 戦略について説明し，数値計算実験からパラメータと得られる解の良さと必要な時間の関係について調査し，パラメータ調整の指針を示す．また，各ベンチマーク問題に対して 提案手法と GA などの代表的なメタヒューリスティクスを適用し，提案手法がこれらの手法より短時間で良い解を求められることを示す．

目次

第 1 章

序論

1.1 研究背景

巡回セールスマン問題 （Traveling Salesman Problem, TSP）は古くから研究されている組合せ最適化問題の一つであり, 訪問すべき都市群と都市間の距離が与えられたとき, すべての都市を一度ずつ訪問して出発した都市に戻る経路 （巡回路）の中で総経路長が最も小さい巡回路を見つける問題である．TSP の定義は単純であるが，複雑性クラスは NP 困難に属しており，実時間内に厳密解を得ることが一般的に困難である．TSP は運搬経路計画，遺伝子地図，X 線結晶構造解析，VLSI 設計など広い分野で応用されており, 総経路長の小さい解を短時間で求める手法の研究が古くから盛んに研究されている．現在，TSP に対して有効な近似解法として，LKH[1, 2] や GA-EAX[3, 4] が知られている．LKH は α-value による候補辺の絞り込みと高速 Lin-Kernighan 法からなる TSP ソルバであり，1,000 都市程度の問題に対しては数秒で，10,000 都市程度の問題に対しては 1 時間程度でほぼ厳密解を求めることができる．また，この高速性を活かして，LKH で得られた複数の解を初期解として，遺伝的アルゴリズム（Genetic Algorithm, GA）によって解の混合を行い，再度 LKH で解を求めていく手法も提案されている．GA-EAX は GA をベースとして，枝組み立て交叉（Edge Assembly Crossover, EAX）と呼ばれる TSP 向けの特殊な交叉法を使った TSP ソルバであり，100,000 都市程度の問題に対して LKH より優れた解を求めることができ，Mona Lisa TSP Challenge において現在最も優れた解を導いた．これらの手法にも採用されるように，GA は多くの TSP ソルバで利用されている．TSP ソルバにおける GA の特徴として，様々な候補解を持つことによって解空間の広い範囲を探索できることや，突然変異による局所最適解を脱出する機構などがある

が，強力なヒューリスティクスを適用しないかぎり，良い解が見つかるまでに長い時間がかかる問題点もある．

一方，近年注目されている実数値最適化手法の一つに粒子群最適化（Particle Swarm Optimization, PSO）がある．PSO では，解空間を動き回る複数の探索単位（粒子）は自身がそれまでの探索で発見し た最良解 （*pbest*）と群全体の中での最良解（*gbest*）の情報を共有しながら探索を進める．このとき，各粒子には *pbest* と *gbest* への引力が働き，これまでに発見された良い解の近傍を中心に探索する傾向がある．PSO は多くの多峰性の大域的最適化問題に対して，最適解もしくは準最適解を実用的な計算時間内に求めることが可能なことが示されている．また，そのアルゴリズムが非常に単純であることも特徴である．しかし，PSO は組合せ最適化問題である TSP にそのまま適用することは難しい．これまでに，いくつかの TSP 向けに改良された PSO[5, 6, 7] が提案されているが，総経路長が小さな解を得るためにはとても長い時間が必要なことや，アルゴリズムの空間計算量が大きいことなどから，非実用的である．

1.2　研究目的および方法

本研究では，GA などの既存の近似解法と比較して，高速に高精度の解を発見可能な近似解法の構築を目的として，PSO を基にしたアルゴリズムである挿入操作 PSO 戦略（Insertion-based PSO strategy, IPSO）を提案する．提案手法では，粒子の解候補は実数値ベクトルではなく巡回路として表現され，粒子間の相互作用は部分経路挿入によって行われる．この変更により，PSO の *pbest* と *gbest* の周囲を探索するというポリシーを継承しながら，TSP を解くことを可能にする．また，提案手法の特徴や有効性を議論するために数値計算実験を行う．まず, パラメータと得られる解の良さと必要な時間の関係について調査し，パラメータ調整の指針を示す．次に，各ベンチマーク問題に対して提案手法と GA などの代表的なメタヒューリスティクスを適用し，提案手法がこれらの手法より短時間で良い解を求められることを示す．これらの実験から得られた知見をもとに，さらに高精度な解を得るために，粒子間通信を制限した挿入操作 PSO 戦略を提案し，数値計算実験から目的が達成できたことを示す．そして，提案手法のさらなる高速化として，アルゴ リズムの並列化について検討する．粒子の更新が探索の大部分を占めていることと，それらが独立に処理されていることに着目し，これらをスレッドを用いて並列処理することで，大幅な計算時間の削減が可能であることを示す．

1.3　論文の内容の新規性と有効性

本論文の新規性は，PSO の短時間で高精度の解を発見可能という特性に着目し，この特性を生かした挿入操作 PSO 戦略を開発，代表的なヒューリスティクスとの比較から，優位性を示した点である．LKH や GA-EAX などの強力な近似解法は，解の更新操作を徹底的に作り込むことで，高精度を実現している．一方，挿入型粒子群最適化はいたって単純な操作により探索が進む．現時点では提案手法より LKH や GA-EAX のほうが優れた解を導き出せるが，提案手法は LKH や GA-EAX の作り込まれた操作を導入することができ，さらに高速に良い解を導くことが可能であると考える．また，簡単に並列化が可能であり，大規模並列実行環境における運用に適したアルゴリズムになっている点で有効性があると考えられる．

1.4　論文の構成

本論文は全 7 章で構成される．第 1 章では研究背景および目的を述べ，提案手法のアプローチについて説明する．第 2 章では巡回セールスマン問題の定義や種類，問題の難しさ，従来解法などについて述べる．第 3 章では提案手法のベースとなっている粒子群最適化について説明する．また，粒子群最適化を基にした巡回セールスマン問題の解法についての従来研究についても説明する．第 4 章では提案手法である挿入操作 PSO 戦略について詳細に説明し，数値計算実験からパラメータチューニングや既存の近似解法と比較して性能評価を行う．パラメータチューニングでは，部分経路挿入に用いる部分経路の長さと群全体の振る舞いの関係性を明らかにし，提案手法が高速に良い解を見つけることができる理由や，さらに高精度な解を得るために必要なことを考察する．既存近似解法との比較では，制限時間を設ける場合に提案手法が最も良い解を得ることができることを示す．第 5 章では 4 章での結果をもとに，より高精度な解が得られるように挿入操作 PSO 戦略を改良する．具体的には，*gbest* を廃止して群全体での巡回路の共有をやめる代わりに，粒子を重複が可能な小規模の群に割り当て，その小群の中の最良解 *lbest* を共有するように変更する．この変更により，解空間内における粒子の密集を起こりにくくし，局所最適解に陥りにくくなる．数値計算実験では，実際に改良前より良い解が得られることを示す．第 6 章では提案手法をスレッドプログラミングにより並列実装し，数値計算実験から大幅に計算時間を減少可能であることを示す．最後に第 7 章において本論文の結論を記述

する.

第 2 章

巡回セールスマン問題

巡回セールスマン問題（Traveling Salesman Problem, TSP）は古くから知られる組合せ最適化問題のひとつである．この問題を簡単な例を以下に示す．

- あるセールスマンは自社を出発して複数の取引先企業に行かなければならない．出発してからすべての取引先を訪問して自社に戻ってくるまでの移動に必要な時間をできるだけ小さくしたいときに，どのような順番で取引先を訪問すればよいか．

上記の問題は，訪問すべき場所が多い場合，訪問順の候補が膨大なため，解くことが困難である．そのため，実世界でこのような問題に対しては，制限時間内でできるだけ良い解を求める解法が用いられる．

本章では，本研究が対象とする問題である巡回セールスマン問題について説明する．また，巡回セールスマン問題の解法について代表的な手法を紹介する．

2.1 巡回セールスマン問題とは

巡回セールスマン問題（以下 TSP）は都市集合と都市間のコストが与えられたときに，総コストが最小となる巡回路を求める問題である．以下に用語を説明する．

都市集合と都市数

セールスマンが訪問すべき都市の集合を都市集合 $\boldsymbol{V}$ と呼ぶ．都市集合 $\boldsymbol{V}$ に含まれる都市はすべて独立である．つまり，$\boldsymbol{V}$ に含まれる任意の i，j について必ず $i \neq j$ が成り立つ．また，都市集合 $\boldsymbol{V}$ の要素数を都市数 n と呼ぶ．つまり，$n = |\boldsymbol{V}|$ が

成り立つ．都市数は次元数と呼ばれることもある．本論文では，都市数 n の都市集合を $\boldsymbol{V} = \{0, 1, 2, \cdots, n-1\}$ と表記する．

巡回路

セールスマンはある都市 $i \in \boldsymbol{V}$ を起点として，$\boldsymbol{V}$ に含まれる i 以外のすべての都市を 1 度ずつ訪問し，最期に i と戻る．この訪問順序を巡回路 $\boldsymbol{T}$ と呼ぶ．巡回路 $\boldsymbol{T}$ は $\boldsymbol{T} = (0, 1, 2, \cdots, n-1)$ のように訪問した都市の順を列挙することで表せる．この例では，都市 0 を起点として，都市 1，都市 2 と順番に訪問し，都市 $n-1$ を訪問した後で起点である都市 0 に戻る巡回路を表している．$\boldsymbol{T}^i$ は i 番目に訪問する都市を表す．このとき，i は $[0, n-1]$ の範囲の整数であり，$\boldsymbol{T}^0$ は始点と終点になる都市を表す．この巡回路表現において，巡回路 $\boldsymbol{T}$ は以下の要件を満たしている．

1. $|\boldsymbol{T}| = n$ である．
2. $i \in \{0, 1, 2, \cdots, n-1\}$ を満たすすべての i に対して，$\boldsymbol{T}^i \in \boldsymbol{V}$ である．
3. $i, j \in \{0, 1, 2, \cdots, n-1\}, i \neq j$ を満たすすべての i, j に対して，$\boldsymbol{T}^i \neq \boldsymbol{T}^j$ である．

ある TSP に対する巡回路は複数存在し，これらの集合を巡回路集合 $\mathbb{T}$ と表記する．また，巡回路は始点と終点になる都市に特別な意味を持たないので，ある巡回路 $\boldsymbol{T}_a$ と $\boldsymbol{T}_b = (\boldsymbol{T}_a^m, \boldsymbol{T}_a^{m+1}, \cdots, \boldsymbol{T}_a^{n-1}, \boldsymbol{T}_a^0, \cdots, \boldsymbol{T}_a^{m-1})$ を満たす巡回路 $\boldsymbol{T}_b$ は同じ巡回路を表している．

コスト

コストは都市から都市への移動にかかる費用である．距離という言葉が用いられることもあるが同義である．コストは各都市の組み合わせごとに設定され，都市 i から都市 j までのコストを c_{ij} と表記する．巡回路 $\boldsymbol{T}$ のコスト $C_{\boldsymbol{T}}$ は，巡回路を 1 周するために必要なコストであり，式 2.1 に示すコスト関数 C を用いて算出する．

$$C(\boldsymbol{T}) = \sum_{i=0}^{n-2} c_{\boldsymbol{T}^i \boldsymbol{T}^{i+1}} + c_{\boldsymbol{T}^{i+1} \boldsymbol{T}^0} \tag{2.1}$$

解，厳密解

TSP において，解とは巡回路と等価である．一方，厳密解は，巡回路集合 $\mathbb{T}$ に含まれる巡回路の中で最もコストが小さい巡回路である．厳密解 $\boldsymbol{T}_o$ は巡回路のコスト関数 C を用いると式 2.2 で定義される．

$$\boldsymbol{T}_o = \arg\min_{\boldsymbol{T}\in\mathbb{T}} C(\boldsymbol{T}) \tag{2.2}$$

グラフ

TSP はしばしばグラフ理論の言葉で説明される．TSP は都市と都市間のコストを頂点と辺の重みに対応させた重み付きグラフ G と考えることができ，TSP の目的はこの重み付きグラフ G 上で重みの総和が最小になるハミルトン閉路を探索することとなる．

2.2　巡回セールスマン問題の分類

TSP は都市間のコストの性質によって分類される．ここでは，

- 対称巡回セールスマン問題 (Symmetric Traveling Salesman Problem, STSP)
- 非対称巡回セールスマン問題 (Asymmetric Traveling Salesman Problem, ATSP)
- メトリック巡回セールスマン問題 (Metric Traveling Salesman Problem, Metric TSP)
- ユークリディアン巡回セールスマン問題 (Euclidean Traveling Salesman Problem, ETSP)

について述べる．

2.2.1　対称巡回セールスマン問題

対称 TSP においては，以下の式が成り立つ．

$$c_{ij} = c_{ji} (i \neq j, \forall i, \forall j \in \boldsymbol{N}_n) \tag{2.3}$$

ここで，$\boldsymbol{N}_n$ は 1 から n までの整数の集合である．これは，都市間を移動する向きによらずに，2 都市間のコストが与えられることを表す．このことから，対称 TSP においては，(2.4) 式が成り立つ．

$$e_{ij} = e_{ji} (i \neq j, \forall i, \forall j \in \boldsymbol{N}_n) \tag{2.4}$$

ある巡回経路に対して逆回りの巡回経路は，(2.4) 式から同一であるので，対称 TSP においては $(n-1)!/2$ 通りの巡回経路が存在する．これらの性質を利用することで，効率的な巡回経路の探索を行うことができるため，この問題を対象とした近似解法が多く提案されている．

2.2.2　非対称巡回セールスマン問題

(2.3) 式が成り立つことが保証されない TSP を，対称 TSP に対して非対称 TSP と呼ぶ．例えば，配送経路の最適化において，ある地点から別の地点までの最短経路に一方通行の経路が含まれる場合を考える．このとき，移動の向きによっては最短経路を通ることができず，より長い迂回路を通らなくてはならないため，このような配送経路の最適化問題は非対称巡回セー ルスマン問題として表される．非対称 TSP は，対称 TSP を含むより一般的な問題である．

2.2.3　メトリック巡回セールスマン問題

メトリック巡回セールスマン問題は，都市間の移動コストが三角不等式を満たす問題である．このとき，以下の式が成り立つ．

$$c_{ij} + e_{jk} \leq c_{ik} (i \neq j \neq k, \forall i, \forall j, \forall k \in \boldsymbol{N}_n) \tag{2.5}$$

2.2.4　ユークリディアン巡回セールスマン問題

ユークリディアン巡回セールスマン問題においては，都市ごとにユークリッド空間上の座標が与えられ，都市間の移動コストは，都市間の直線距離によって定義される．このため，ETSP は，Metric TSP と STSP の両方に含まれる．

2.3　巡回セールスマン問題の難しさ

2.3.1　組合せ総数

n 都市の TSP の解の候補数について考える．上記の性質から，都市数 n の TSP は長さ n の円順列であるので，この巡回路集合の大きさ $|\mathbb{T}|$ は $(n-1)!$ である．対称 TSP の場合，数珠順列となるので $|\mathbb{T}| = (n-1)!/2$ である．仮に 1 秒間に 1000 万個の巡回路の

表 2.1　対称 TSP の組合せ総数と計算時間

都市数 n	10	15	20	30	100
組合せ総数	1.81×10^5	4.36×10^{10}	6.08×10^{16}	4.42×10^{30}	4.67×10^{155}
処理時間	0.0181 秒	1.21 時間	193 年	1.40×10^{16} 年	1.48×10^{141} 年

コストを評価できるコンピュータがあったとして，対称 TSP のすべての巡回路を評価するのにどれほどの時間がかかるか表 2.1 にまとめる．この結果から 15 都市程度なら全巡回路の比較で最適解を求められるが，それ以上の TSP に対しては現実的ではないことがわかる．

組合せ総数による TSP の困難さの説明は直感的に非常にわかりやすい．しかし，工夫次第では簡単に最適解が求まる問題の可能性があり，TSP の難しさの説明としては簡単なものに留まっている．

2.3.2　計算複雑性

計算複雑性理論では問題の難しさを数学的に扱う．この中でも，特に重要な概念として，$\mathcal{P}$ と $\mathcal{NP}$ がある．現状を踏まえて，簡潔に言えば，$\mathcal{P}$ とは多項式時間のアルゴリズムで解ける決定問題の集合で，$\mathcal{NP}$ は指数時間のアルゴリズムで解ける決定問題の集合である．TSP は $\mathcal{NP}$ を拡張した $\mathcal{NP}$ 困難と呼ばれるクラスに分類されることが証明されている．つまり，n 都市の TSP の厳密解を得るには n の指数時間オーダーの計算時間が必要であることが示されている．

概要としては上記の説明で十分であるが，用語本来の意味を正しくは説明できていないので，下記に用語の詳細について説明する．

時間計算量の基礎

計算複雑性理論では，それぞれのアルゴリズムはチューリングマシンによって評価される．チューリングマシンは，無限に長いテープの上に読み書きができるヘッドがついており，それがテープ状を左右に動きながら動作する仮想的なコンピュータである．チューリングマシンは $M = \langle Q, \Gamma, b, \Sigma, \delta, q_0, F \rangle$ で記述される．Q は内部状態集合，Γ はテープの記号の有限集合，$b \in \Gamma$ は空白記号，$\Sigma \subseteq \Gamma \setminus \{b\}$ は入力記号集合，$\delta : (Q \setminus F) \times \Gamma \to Q \times \Gamma \times \{L, R\}$ は遷移関数で，“機械が状態 $q \in Q \setminus F$ でヘッド位置の記号が $a \in \Gamma$ である場合，状態 $q' \in Q$ に移し，ヘッド位置に記号 $a' \in \Gamma$ を書き込み，

ヘッド位置を $m \in \{L, R\}$ 方向に 1 つずらす”と命令の集合である．$q_0 \in Q$ は初期状態，$F \subset Q$ は受理状態集合である．チューリングマシンでは遷移関数 δ は決定的である．つまり，入力 (q, a) と出力 (q', a', m) が一対になっている．

上記のチューリングマシンと違い，ある入力に対して複数の出力を許すチューリングマシンを非決定性チューリングマシンという．非決定性チューリングマシンでは $\delta : (Q \setminus F) \times \Gamma \to (Q \times \Gamma \times \{L, R\})$ となる．複数の出力を得た場合に自身のクローンを作り，それぞれの出力状態に遷移するチューリングマシンと想像するとわかりやすい．チューリングマシンが現実のコンピュータと非常に近いのに対し，非決定性チューリングマシンが現在の技術で構成不可能であるが，非決定性チューリングマシンは計算問題の重要なクラスの複雑さを特徴づけるための数学的手段として導入された [8]．また，非決定性チューリングマシンに対して，通常のチューリングマシンを決定性チューリングマシンと呼ぶ．

クラス $\mathcal{P}$

クラス $\mathcal{P}$ は，決定性チューリングマシンで多項式時間で解ける決定問題の集合である．決定問題は，yes か no で応える問題のことで，判定問題とも呼ばれる．多項式時間とは，問題の規模 n に対して $O(n)$ の処理時間ということである．$O(n)$ は漸近的記法であり，ある式において n に大きな値を入れた場合にどの項がほとんどを占めるか示す記号である．例えば，$f(x) = 6n^3 + 1n^2 + 4n + 3$ の場合，$f(x) = O(n^3)$ となる．クラス P に属する問題は，ソートの判定問題や最小全域木の判定問題，グラフの 2 頂点間の最短経路の判定問題などがある．

クラス $\mathcal{NP}$

クラス $\mathcal{NP}$ は，非決定性チューリングマシンで多項式時間で解ける決定問題の集合である．また，クラス $\mathcal{NP}$ に属する問題は，解の探索は難しいが，得られた解の検証は簡単であることが示されている．例えば，充足可能性問題（SAT）は与えられた論理式が真となるような変数群の組合せが存在するか判定する問題であるが，そのような変数群を見つけるのは困難であったとしても，見つかった変数群は実際に代入することで簡単に検証可能である．

また，SAT は $\mathcal{NP}$ 完全な問題であることが証明されている [9]．$\mathcal{NP}$ 完全問題とは，$\mathcal{NP}$ に属するすべての問題をその問題に変換可能である問題である．文献 [9] において Stephen A. Cook は非決定性チューリングマシンで多項式時間で終了するあらゆるプロ

グラムを SAT で表現できることを示した．その他に $\mathcal{NP}$ 完全であることが知られている問題の多くは，SAT から多項式時間で変換可能であることに基づいている．代表的な $\mathcal{NP}$ 完全問題として，頂点被覆問題，ハミルトン閉路問題，ナップサック問題などがある．$\mathcal{NP}$ 完全なナップサック問題は“与えられた制約を満たしつつ，価値の目標を満たす解が存在するか判定する”決定問題である．一方，“与えられた制約を満たしつつ，価値が最大になるような解を探索する”ナップサック問題は，後述する $\mathcal{NP}$ 困難に属する．

$\mathcal{NP}$ 困難

問題が $\mathcal{NP}$ 困難であるとは，その問題自体は $\mathcal{NP}$ に入っていないかもしれないが，$\mathcal{NP}$ に含まれるすべての問題がその問題に多項式時間で帰着できるときにいう．しばしば，より簡単に，$\mathcal{NP}$ に属する問題と比較して，少なくとも同等以上に難しい問題と説明される．$\mathcal{NP}$ 困難な問題としては，決定問題へと変換したときに $\mathcal{NP}$ 完全になる最適化問題や，その逆で，解の探索問題へと変換した $\mathcal{NP}$ 完全問題が多い．ハミルトン閉路探索問題やナップサック問題，最小頂点被覆問題が代表例である．その他にも，停止性問題も SAT から多項式時間で変換可能であることを理由に $\mathcal{NP}$ 困難に属する．

巡回セールスマン問題がどのクラスに属しているかは Richard M. Karp によって証明されている [10]．その証明では，巡回セールスマン問題は決定問題として扱われており，コストが a 以下の巡回路が存在するか判定する問題に変更されている．この巡回セールスマン決定問題は，ハミルトン閉路問題から多項式時間変換可能であるので，$\mathcal{NP}$ 完全である．また，最適化問題としての巡回セールスマン問題も，ハミルトン閉路問題から同様に変換可能である．しかし，こちらは $\mathcal{NP}$ に属していないため，$\mathcal{NP}$ 困難となる．

2.4　巡回セールスマン問題の解法

上記に説明したように，巡回セールスマン問題は $\mathcal{NP}$ 困難であり，最適解を多項式時間で解くアルゴリズムは解明されていない．そのため，巡回セールスマン問題に対しては，実時間内で出来るだけ良い解を求めるために，様々な近似解法が研究されてきた．本節では，本研究と関連の深い手法に関して，いくつかの従来研究を紹介する．

2.4.1　構築法

構築法とは，その名の通り，巡回路を生成する手法である．必ずしも厳密解は見つからないが，ある程度の精度の解を高速に生成することができる．この特性を活かして，後述するメタヒューリスティクスを用いた近似解法の初期解として利用されることも多い．また，コストに三角不等式が成り立つ，メトリック TSP に対しては，精度保証が証明されている手法が多い．以下の構築法のそれぞれの手法で精度保証についても説明するが，この精度保証はメトリック TSP に対してのみ有向である．

最近傍法

最近傍法（Nearest Neighbor, NN）は，ある都市からスタートし，現在の都市から未訪問の都市の中で最も距離が近い都市に遷移することを繰り返して巡回路を形成する．NN のみで最適解を得ることは難しいが，得られた巡回路の部分経路はコストが小さいという特徴がある．NN のアルゴリズムを以下に示す．

1. 頂点集合 V から任意の頂点を現在の頂点 $v = v_0 \in V$ として選ぶ．
2. 未訪問の頂点集合 V' を $V' = V \setminus \{v_0\}$ のように初期化する．
3. 現在の頂点 v と未訪問の頂点集合 V' の各頂点を結ぶ辺の集合 $\{e(v,i)|i \in V'\}$ の中から最もコストが小さい辺 $e(v,i)$ を探索する．
4. 現在の頂点を $v = i$ と更新する．
5. 未訪問の頂点集合 V' から i を取り除く（$V' \leftarrow V' \setminus \{i\}$）
6. 未訪問の頂点集合 V' が空であれば終了する．
7. (3) に戻る．

NN は時間計算量が $O(n^2)$ で，精度は $O(logn)$ であることが知られている．

挿入法

挿入法は構築法に分類される近似解法である．挿入法では，初期の巡回経路として，すべての都市を含まない不完全な巡回経路を作成する．初期の巡回経路としては，問題に含まれる都市からランダムに選択された 3 都市のみからなる巡回経路などが用いられる．不完全な巡回経路が作成されたのち，巡回経路に含まれない都市から一都市を選び，最もコストの増加量が小さい辺へ挿入する操作を繰り返す．都市の辺への挿入は，以下の操作に

よって行われる．ただし，挿入する都市を i，挿入先の辺を e_{jk} とする．

1. 巡回経路から e_{jk} を取り除く．
2. 巡回経路へ e_{ji}，e_{ik} を加える．

挿入する都市を選択する順番によって，最終的に得られる巡回経路は異なる．挿入する都市の選択基準として，コストの増加量が最も大きくなる都市を選択する方法（最遠挿入法），コストの増加量が最も小さくなる都市（最近挿入法）を選択する方法，$minC(e_{ik}) + C(e_{kj}) - C(e_{ij})$ が最小になる都市 k を選択する方法（最安挿入法）などが考案されている．これらの手法は精度保証 $r = 2$ を持つことが知られている [11].

2.5　メタヒューリスティクスを用いた近似解法

ここでは TSP の従来解法として,

- 山登り法 (Hill Climb Method, HC)
- 遺伝的アルゴリズム (Genetic Algorithm, GA)
- アントコロニー最適化 (Ant Colony Optimization, ACO)
- 擬似焼きなまし法 (Simulated Annealing, SA)
- タブーサーチ (Tabu Search, TS)
- 局所クラスタリング組織化法 (Local Clustering Organization, LCO)
- ハイブリッド手法

について述べる.

2.5.1　山登り法

山登り法は最も基本的な局所探索のひとつ [12] であり, TSP を含む多くの組み合わせ最適化問題に対して適用可能な方法である.

山登り法では, まず現在保持している解 (暫定解) を少しだけ変更して生成される解 (近傍解) を調べる．もし近傍解の巡回路長のほうが現在のものより短いのであればその近傍解を暫定解とする．以降, 近傍解を調べて暫定解よりも短い巡回路であれば暫定解を近傍解で置き換える, という操作を繰り返す．この操作を暫定解より短い巡回路長の近傍解が見つからなくなるまで繰り返し, 得られた解が山登り法の解となる [13].

2.5.2　遺伝的アルゴリズム

遺伝的アルゴリズム (Genetic Algorithm, GA) は J.H.Holland により提案された手法で [14, 15], 自然界における進化のアナロジーを用いて解の探索を行う.

GA では解を複数用意し, それらに対して複数の遺伝操作を適用することで解を改善する. 適用される遺伝操作は, 2 つの個体から新しい個体を得る交叉 (crossover), 遺伝子の一部分をランダムに変化させる突然変異 (mutation), 適応度により次の世代に生き残る物を選ぶ選択 (selection) などがある. ここで個体とは対象となる問題の解, 適応度とは問題の目的関数により得られる評価値などである.

TSP に対して GA を適用する場合は, 個体として巡回経路が, 適応度として巡回経路の長さがあてはめられることが多い. GA を TSP に対して適用した文献として [16][17][18][3][19] などがある.

この中で, [16][3] は TSP を解くことに特化した交叉方法を提案し, 従来の交叉方法を用いた場合と比較して短い巡回路を導出している. また [19] は [3] の手法に対して解の多様性を考慮した評価関数を導入することで, [3] で得られた解よりも更に短い巡回路を導出している.

2.5.3　アントコロニー最適化

アントコロニー最適化法 (Ant Colony Optimization, ACO) とは M.Dorigo により提案された, アリが食物を見つけるために集団として行動するときの群行動を基にした手法である [20].

現実世界において, 巣 (コロニー) から出発したアリはランダムに移動して食物を探す. 運よく食物を見つけた場合は, 見つけたアリはフェロモンと呼ばれる誘引物質を残しながら巣へ戻る. このフェロモンを同様にランダムに移動しながら食物を探すアリが見つけると, 他のアリはランダムな行動を止めてフェロモンの跡をたどるようになる. そして同じように食物を見つけるとフェロモンを残しながら巣へ戻ることで経路上のフェロモンが濃くなり, より多くのアリが通るようになる. しかし, フェロモンは時間とともに蒸発し, その量は減っていく. このことから, 長い経路よりも短い経路のほうがフェロモンが蒸発するよりも早く分泌されるため, フェロモンの濃度が高く保たれやすい. したがって, 巣から食料までの短い経路が見つかると, 他のアリもその経路をたどる可能性が高くなり, 正の

フィードバック効果によって全てのアリが同じ経路をたどることになる.

アントコロニー最適化はこの考え方をもとに, 実際のアリを模したエージェントに巡回路を生成させ, 距離の長さに応じてフェロモンを残す. エージェントは次の都市までの距離及びフェロモンの濃度に応じて都市を選択する. このようにエージェントの移動とフェロモンの増強を繰り返すことで探索を行う.

ACO を TSP に対して適用した文献として [20][21][22] などがある. 現実世界における道路状況のように, 都市間のコストが動的に変化するような TSP に対して ACO は他の手法に比べて良い結果が得られている.

2.5.4　擬似焼きなまし法

擬似焼きなまし法 (Simulated Annealing, SA) とは S.Kirkpatrick, C.D.Gelatt, M.P.Vecchi らにより提案された, 金属工学における焼きなましの工程を模した最適化手法である [23]. 本来焼きなましとは金属を熱した後に徐々に冷やし, その結晶を成長させることでその金属の欠陥を減らす作業のことである. 金属を加熱することで金属原子はエネルギーの高い状態に遷移する. この後ゆっくりと温度を下げていくことにより, 原子は初期の状態よりもエネルギーが更に低い状態が得られる可能性が高くなる.

SA では山登り法と同様に現在の解の近傍解を調べるが, 山登り法と異なり, 解が改善されなくても温度と改悪量によって決定される確率によって近傍解への遷移も許容する. 温度の値は冷却スケジュールにより徐々に小さくなっていく. このため, 探索の初期では受理確率が高いため広範囲の探索を行い, 探索が進むにつれて温度が下がり, 次第に改善する解のみを受理するので探索が収束していく.

SA を TSP に対して適用した文献として [23][24] などがある. SA は適切な初期温度と冷却スケジュールを設定することで, 局所解に陥りにくい大域的な最適化が可能である. しかし, 問題により適切な初期温度および冷却スケジュールが異なるため, 良い結果を得るためには予備実験などによるパラメータの事前調整が不可欠であり, 準備に手間がかかるという欠点を持つ.

2.5.5　タブーサーチ

タブーサーチ (Tabu Search, TS)[25] とは F.Glover により提案された手法で, 一度探索した解を記憶しておくことで再び同じ解を探索することを防ぎつつ解を探索する. タ

ブーサーチでは近傍解の中から最も巡回路長が短い解を選択する. この時, タブーリストに解の遷移を記憶しておく. タブーリストに書き込まれている解遷移を禁止することで同じ解を探索することを防ぐ. 操作がタブーリストに記載されていない場合は近傍解のほうが巡回路長が長くなってしまってもその解を選択する. このことにより局所解で探索が停滞することを防いでいる.

TS を TSP に対して適用した文献として [24][26] などがある. この手法は扱うパラメータが少ないため, パラメータの調整に要する時間が短く, かつ, かなり精度の良い解を導出することができる. しかし, 探索の際には近傍を探索する度にその近傍がタブーリストに記載されているかを確認するため, 探索に時間がかかるのが欠点である.

2.5.6　局所クラスタリング組織化法

局所クラスタリング組織化法 (Local Clustering Organization, LCO) とは古川らにより提案された手法で [27], 局所的な最適化を繰り返すことで全体の最適化を行う手法である. LCO ではクラスタリングと呼ばれる手法を用いて暫定解の一部分の最適化を行う. クラスタリングには単純交換法 (Simple Exchange Method, SEM), 逆位交換法 (Inverse Exchange Method, IEM), 平滑法 (Smoothing Method, SM) の 3 つの方法があり, それぞれあらかじめ指定した割合で選択され, 適用される.

LCO を TSP に対して適用した文献として [27] などがある. LCO は他の手法と比較して, 比較的高精度な解を非常に短時間で導出することが可能である. しかし, 解が改善されるときのみ解遷移を行うため, 一度局所解に陥ってしまうと脱出することが不可能という欠点がある.

2.5.7　ハイブリッド手法

ハイブリッド手法とは, 複数の解法を組み合わせた手法のことである. TSP の解を求める手法は, 上で述べたものをはじめ数多く存在するが, それぞれの手法が解を求めるための戦略は, それぞれ異なっている. 例えば, 同じ程度の解精度を示す方法 A と方法 B が存在しても A と B の解の探索方法などは異なる. このため, 方法 A で陥らないような局所解に方法 B では陥るということも起こりうる. これに対して, いくつかの方法をハイブリッド (組み合わせ) することによって, それぞれの方法の欠点を補いあう試みがなされている. 実際に複数の手法を組み合わせることで, それぞれ単体の手法よりも精度の良い解

を導出した例として, [24][28] などがある.

複数の手法をハイブリッドすることにより, 一方の手法で陥ってしまう局所解から, もう一方の手法によって脱出することが可能となり, 解精度が向上することが期待できる. しかし, ハイブリッドすることにより, それぞれ単体の手法で用いられているパラメータのほかに, 組み合わせる割合を決めるパラメータなどが増えることが考えられ, 効果的に適用するためには予備実験などの手間がかかるという欠点もある.

2.6　TSPLIB

TSPLIB[29, 30] は，様々なタイプの TSP を収録したライブラリである．近似解法の比較をする場合によく利用される．本論文においても，提案手法と既存手法の比較をする際に使用する.

2.6.1　TSPLIB に掲載されている問題

TSPLIB は，TSP とその関連問題のライブラリである．収録されている問題群には次のものがある.

- 対称巡回セールスマン問題（Symmetric Traveling Salesman Problem, STSP）
- 非対称巡回セールスマン問題（Asymmetric Traveling Salesman Problem, ATSP）
- ハミルトン閉路問題（Hamiltonian Cycle Problem, HCP）
- Sequential Ordering Problem (SOP)
- 積載量制限のある配送経路問題 (Capacitated Vehicle Routing Problem, CVRP)

STSP と ATSP については前述のとおりである．以下では，STSP と ATSP 以外の問題について説明する.

ハミルトン閉路問題

HCP は，グラフが与えられたときに，すべての頂点を 1 度ずつ通る閉路が存在するか調べる判定問題である．ハミルトン閉路問題は $\mathcal{NP}$ 完全であり，最適解の探索が難しい問題である.

表 2.2　TSPLIB に収録されている対称 TSP の仕様部分で使用されるタグとその説明

タグ	データ型	値が示す内容
NAME	文字列	この問題の名前
TYPE	文字列	問題の種類．STSP の場合，“TSP”と記述する．
COMMENT	文字列	コメント．多くの場合，問題の作成者や投稿者が記述される．
DIMENSION	整数	TSP においては都市数
EDGE_WEIGHT_TYPE	文字列	辺のコストの与えられ方

Sequential Ordering Problem

SOP は ATSP に特別な制約が追加された問題である．SOP は，都市集合とそれらの都市間の非対称なコストが与えられたときに，ある制約を満たしながら都市 1 から都市 n までの経路コストが最小であるハミルトン路を見つける問題である．制約は，“都市 a の前に都市 b を訪問する”という形式で与えられる．

積載量制限のある配送経路問題

CVRP は TSP ととても良く似た問題である．TSP が 1 人で任意の都市からすべての都市を回るのに対し，CVRP では数台のトラックが集配拠点から手分けしてすべての都市を巡る．それらの都市は集配拠点からの荷物を待っており，トラックに 1 度に積載できる荷物の量は限られている．このような制約において，最も効率の良い配送経路を探索する問題である．

2.6.2　TSPLIB に収録されている対称巡回セールスマン問題

本論文では，TSP の中でも最も有名で基本的である，対称 TSP を扱う．TSPLIB 内の STSP の多くは 2 次元ユークリッド TSP であり，これを本論文の対象とする．

TSPLIB に収録されている問題は，決められたフォーマットで書かれたファイルである．ファイルの内容は仕様部分（specification part）とデータ部分（data part）に分けられる．それぞれの内容は“⟨ タグ ⟩ : ⟨ 値 ⟩”の形式で記述される．STSP の仕様部分に関するタグについて表 2.2 にまとめる．“EDGE_WEIGHT_TYPE”は，どのように辺のコストが与えられるか示している．例えば，2 次元ユークリッド TSP の場合，

“EUC_2D”と記述して，データ部分ではそれぞれの都市の座標を記述する．データ部分では，“EDGE_WEIGHT_TYPE”に従ってデータの記述を行う．EUC_2D の場合，タグ“NODE_COORD_SECTION”につづいて，“⟨ 都市番号 ⟩ ⟨x 座標 ⟩ ⟨y 座標 ⟩”のように記述する．都市番号は整数，x, y 座標は実数で指定する．ファイルの末尾にはタグ“EOF”を記述し，ファイルの終端であることを示す．

辺のコスト行列を得るには，EDGE_WEIGHT_TYPE ごとに定められた距離関数を用いる．EUC_2D では，辺 $e(a, b)$ のコストは都市 a と都市 b の座標を使って 2.6 式によって定義される．

$$e(a, b) = nint\left(\sqrt{(a_x - b_x)^2 + (a_y - b_y)^2}\right) \tag{2.6}$$

ここで，$nint$ は四捨五入関数であり，次式のようにガウス記号を用いても表現できる

$$e(a, b) = \left[\sqrt{(a_x - b_x)^2 + (a_y - b_y)^2} + 0.5\right] \tag{2.7}$$

第 3 章

粒子群最適化

3.1 粒子群最適化

粒子群最適化 (Particle Swarm Optimization, PSO) [31, 32] は, 1995 年に Kennedy と Eberhart によって提案された実数値最適化手法の一つであり, 鳥や魚などの群 (swarm) の採餌行動などに見られる群行動を探索手法に応用したものである．複数の粒子（particle）が解空間内に散りばめられ，それらが相互作用しながら動き回ることで探索が進む．具体的には，ある粒子 i は n 次元の実数値ベクトルで表現される位置 $\boldsymbol{x}_i$ と速度 $\boldsymbol{v}_i$ の情報を保持しており，位置 $\boldsymbol{x}_i$ が問題の解を表現する．ステップ t における粒子 i の位置 $\boldsymbol{x}_i(t)$ と速度 $\boldsymbol{v}_i(t)$ は，(3.1)，(3.2) 式に従い更新される．

$$\boldsymbol{v}_i(t) = w\boldsymbol{v}_i(t-1) + c_1 r_1(\boldsymbol{p}_i - \boldsymbol{x}_i(t)) + c_2 r_2(\boldsymbol{g} - \boldsymbol{x}_i(t)) \tag{3.1}$$

$$\boldsymbol{x}_i(t+1) = \boldsymbol{x}_i(t) + \boldsymbol{v}_i(t) \tag{3.2}$$

ここで，w，c_1，c_2 は定数，r_1，r_2 は i に対して独立な $[0, 1]$ の乱数，$\boldsymbol{p}_i$ はその粒子 i がこれまでに発見した最良な位置（*pbest*），$\boldsymbol{g}$ は群全体としてこれまでに発見した最良な位置（*gbest*）である．$\boldsymbol{p}_i$ と $\boldsymbol{g}$ は粒子 i の更新時に適宜更新する．これらの操作を一定回数繰り返し，より良いを探索する．以後，単に $\boldsymbol{x}_i$ と $\boldsymbol{v}_i$ と表記したときは，$\boldsymbol{x}_i(t)$ と $\boldsymbol{v}_i(t)$ を意味する．PSO は，遺伝的アルゴリズム（Genetic Algorithm，GA）[14] と比較して高速に良い解を発見でき，次元数の増加や多峰性が強い問題に対しても GA より有効であることがわかっている．[33].

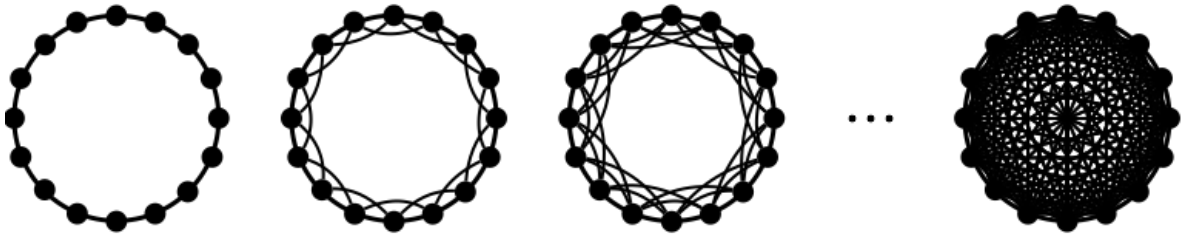


図 3.1　粒子間の通信を制限した PSO で用いられる近傍粒子グラフ．黒点は粒子，黒点を結ぶ線は近傍関係を示す．左から $k = 2$，$k = 4$，$k = 6$，$k = m - 1$．

3.2　Local version

また，粒子間の通信を制限した PSO[34] も提案されている．この PSO では，各粒子は近傍と呼ばれるその粒子と通信可能な複数の粒子が設定されている．そして，粒子 i の速度 $\boldsymbol{v}_i$ の更新には，*gbest* の代わりに，i と近傍内の粒子が探索した中での最良解（*lbest*）を用いる．具体的な速度 $\boldsymbol{v}_i$ の更新式を (3.3) 式に示す．

$$\boldsymbol{v}_i(t) = w\boldsymbol{v}_i(t-1) + c_1 r_1(\boldsymbol{p}_i - \boldsymbol{x}_i(t)) + c_2 r_2(\boldsymbol{l}_i - \boldsymbol{x}_i(t)) \tag{3.3}$$

ここで，$\boldsymbol{l}_i$ は粒子 i の *lbest* である．各粒子 i の近傍 A_i はすべての粒子を 1 次元格子状に辺で接続したときに d 個隣りまでの粒子の集合となる．粒子間の近傍関係により作られるグラフを図 3.1 に示す．

各頂点の次数 k は $k = 2d$ であり，粒子数を m とすると $2 \leq k < m$ が成り立つ．特別な例として，すべての粒子を近傍と設定する場合，*gbest* を用いる PSO と同じ動作となる．このとき，すべての粒子は次数 $k = m - 1$ となる．これ以降，*gbest* を利用した PSO は，$k = m - 1$ を用いる *lbest* を利用した PSO として扱う．この拡張された PSO では，k が小さな値のときに局所最適解に陥りにくい特徴がある．

3.3　TSP 向けに拡張された PSO

PSO を TSP に向けに改良した手法として，Swap Operator を用いる PSO[5, 7] や Fuzzy 行列を用いる PSO[6] が提案されている．

3.3.1　交換子を用いた PSO

交換子を用いた PSO[5] では，巡回路を粒子の位置 $\boldsymbol{x}_i$ としてもち，巡回路中の i 番目と j 番目の要素を入れ替える操作を交換子 $SO(i,j)$, 交換子を複数並べた列を交換子列 SS とし定義し, 粒子の速度 $\boldsymbol{v}_i$ を交換子列 SS とした．粒子の位置 $\boldsymbol{x}_i$ は交換子列 SS に含まれる交換子 SO を順に適用することで更新され，速度 $\boldsymbol{v}_i$ である交換子列 SS は，ある巡回路 $\boldsymbol{a}$ から別の巡回路 $\boldsymbol{b}$ に変更するために必要な交換子列を $SS(\boldsymbol{a} \to \boldsymbol{b})$ と表記すると，現在の巡回路を *pbest* に変更するために必要な交換子列 $SS(\boldsymbol{x}_i \to \boldsymbol{p}_i)$ と，現在の位置を *gbest* に変更するために必要な交換子列 $SS(\boldsymbol{x}_i \to \boldsymbol{g})$ を用いて更新する．しかし，この手法では $SS(\boldsymbol{x}_i \to \boldsymbol{p}_i)$ と $SS(\boldsymbol{x}_i \to \boldsymbol{g})$ に含まれる交換子をランダムにいくつか選択して混ぜ合わせるため，更新後の位置 $\boldsymbol{x}_i$ は $\boldsymbol{p}_i$ にも $\boldsymbol{g}$ にも似ていない巡回路となることが多く，PSO の“良い位置の近傍を重点的に探索する”という戦略を取り込めていない．

文献 [7] では，交換子を用いた PSO の改良を行っている．1 つ目は速度 $\boldsymbol{v}_i$ の計算方法の変更である．PSO の t ステップ目の速度 $\boldsymbol{v}_i(t)$ は，次式のように展開することができる．

$$\begin{aligned}\boldsymbol{v}_i(t) &= w\boldsymbol{v}_i(t-1) + c_1 r_1 \{\boldsymbol{p}_i(t) - \boldsymbol{x}_i(t)\} + c_2 r_2 \{\boldsymbol{g}(t) - \boldsymbol{x}_i(t)\} \\ &= c_1 r_1 \{\boldsymbol{p}_i(t) - \boldsymbol{x}_i(t)\} + c_2 r_2 \{\boldsymbol{g}(t) - \boldsymbol{x}_i(t)\} \\ &\quad + w\left[w\boldsymbol{v}_i(t-2) + c_1 r_1 \{\boldsymbol{p}_i(t-1) - \boldsymbol{x}_i(t-1)\} + c_2 r_2 \{\boldsymbol{g}(t-1) - \boldsymbol{x}_i(t-1)\}\right]\end{aligned} \tag{3.4}$$

改良交換子 PSO では，(3.4) 式の $\boldsymbol{v}_i(t-2)$ の項を省略し，現在の粒子と *pbest*，*gbest* の差と 1 ステップ前における粒子と *pbest*，*gbest* の差の情報だけを用いて現在の速度 $\boldsymbol{v}_i(t)$ を計算している．この変更により，$\boldsymbol{v}_i(t-1)$ を用いた場合によく起こっていた，1 ステップ前に適用した SO を再度適用してしまい，探索が進まなくなる問題点を克服した．2 つ目は，高速化のために粒子を速度 $\boldsymbol{v}_i$ に基づいて更新した後に，巡回路に 2-opt を適用するように変更した点である．しかし，上記ような変更を加えても依然として計算速度も遅く，解精度も高くない．

3.3.2　Fuzzy 行列を用いた PSO

Fuzzy 行列を用いた PSO[6] では，都市数を n とすると，都市 i から都市 j に遷移する優先度 w_{ij} を要素にもつ $n \times n$ の行列 W を粒子の位置としている．行列 W は, W 内で最も高い w_{ij} である (i,j) を部分経路 $\boldsymbol{s} = (i,j)$ とし，これ以降，部分経路の末尾の都市

を k として，$\boldsymbol{w}_k = (w_{k1}, w_{k2}, \cdots, w_{kn})$ 内で $l \notin \boldsymbol{s}$ を満たす w_{kl} が最も大きな値となる l をこの部分経路 $\boldsymbol{s}$ の末尾に付け加える操作をすべての都市を訪問するまで繰り返し，最後に末尾から始点へのパスを加えることで巡回路に変換できる．この方法では，$n \times n$ の 2 次元の行列 W を n^2 次元数ベクトルに変換し，PSO の更新式をそのまま適用して解探索を行う．しかし，この方法では各粒子ごとに $n \times n$ の行列 W をもつため，大規模な問題を扱うには大きな計算空間が必要となる．

第 4 章

挿入操作 PSO 戦略

PSO は高速に高精度な解を発見可能な手法として知られている．特に，有名な近似解法であり，同じ多点探索を行う手法である GA と比較して，次元数や多峰性が増加に強いことが示されている．このような PSO の探索性能を TSP でも発揮できないか研究されてきたが [5, 6]，これらは，低速で解精度も悪かったり，空間計算量が大きく実装が困難だったり，お世辞にも良い手法とはいえない．本論文では，上記の手法と全く異なるアプローチを用いて，高速に高精度な解を発見可能な，PSO をベースとした TSP ソルバについて提案する．

4.1 挿入操作 PSO 戦略のアプローチ

本論文の提案手法である，挿入操作 PSO 戦略 (Insertion-based PSO strategy，IPSO) のアプローチを説明する．

PSO では，各粒子は，その粒子のローカルな最良解と粒子群のグローバルな最良解の影響を受けて，自身の位置を更新する．これを TSP で考えると，各最良解の経路情報を受けて自身の経路を更新することを意味する．その簡単な実装としては，各最良解の経路の一部を自身の経路に埋め込むことが考えられる．しかし，通常の PSO は数値ベクトルを粒子の位置としてもっており，この数値ベクトルの操作で部分経路の挿入を実現することは困難である．そこで，PSO を拡張し，各粒子を都市の巡回路で表現し，部分経路の挿入を簡単に行うことを可能とする．

4.2　挿入操作 PSO 戦略の概要

提案手法 IPSO の概要について説明する．IPSO では，複数の粒子が相互作用しながら解空間を動き回ることにより探索が進む．粒子はそれぞれ解と *pbest* を持っている．解は巡回路であり，*pbest* はその粒子がそれまでに探索した解の中で最も良い解である．また，すべての粒子は *gbest* と呼ばれる，群全体のこれまでの探索で発見した最も良い解を共有している．それぞれの粒子は，*pbest* と *gbest* の部分経路を解に挿入することにより，次の探索点を生成する．

4.3　挿入操作 PSO 戦略における用語

ここで，提案手法の用語について説明する．

粒子

各粒子は粒子 1 や粒子 2 のように“粒子”＋整数記号の形式か，1，2 のように整数記号のみで表記する．粒子の整数番号は“1”から始まる連続した整数を用いる．m 個の粒子を用いる場合，粒子 1，粒子 2，$\cdots$，粒子 m と名付けられる．また，これらの粒子の集合を群 $\mathbb{P}$ と呼び，m 個の粒子からなる群は $\mathbb{P} = \{1, 2, \cdots, m\}$ と表記する．

解

各粒子が持っている現在の探索点である巡回路を解と呼ぶ．特に粒子 i の解を $\boldsymbol{x}_i$ と表記する．解は巡回路であるため，訪問する都市を列挙した表現も用いられる．現在の解 $\boldsymbol{x}_i$ が都市 1，都市 2，$\cdots$，都市 n，都市 1 の順に訪問する巡回路である場合，$\boldsymbol{x}_i = (1, 2, \cdots, n)$ と表現する．また，出発点は関係ないので，$(v_0, v_1, v_2, \cdots, v_{n-1}) = (v_{k\%n}, v_{(1+k)\%n}, v_{(2+k)\%n}, \cdots, v_{(n-1+k)\%n})$ が成り立つ．ここで，k は整数，% は余剰演算子である．

pbest* ，*gbest

それぞれの粒子がその時点までの探索で発見した最も良かった解を *pbest* と呼ぶ．粒子 i の *pbest* を $\boldsymbol{p}_i$ と表記する．

また，群全体でその時点までの探索で発見した最も良かった解を *gbest* と呼び，$\boldsymbol{g}$ と表

記する．*pbest* と *gbest* はその定義から次の関係が成り立つ．

$$\boldsymbol{g} = \arg\min_{\boldsymbol{p}_i \in \boldsymbol{P}} C(\boldsymbol{p}_i) \tag{4.1}$$

ここで，$\boldsymbol{P}$ はすべての粒子の *pbest* の集合，C は巡回路のコスト関数である．

ステップ

各粒子は解の更新の機会を均等に与えられる．すべての粒子が，解の更新と *pbest* および *gbest* の更新判定，更新処理を 1 度完了することを 1 ステップと定義する．つまり，“IPSO を n ステップ実行する”と表現するときは，上記の処理が n 回繰り返されたことを意味する．

2 つの巡回路の距離

PSO において，粒子は実数値ベクトル空間である解空間を動きまわる．このとき，それぞれの粒子間には物理的な距離が存在し，それはユークリッド距離で表現される．各粒子の位置は，$(\boldsymbol{p}_i - \boldsymbol{x}_i)$ や $(\boldsymbol{g} - \boldsymbol{x}_i)$ という，現在地点から *pbest* や *gbest* へ向かうベクトルを用いて，これまで良かった解への引力を受けながら更新される．PSO の探索戦略を取り入れる上で，2 つの実行可能解の距離を定義する必要がある．

2 つの円順列の距離としては，2 の円順列をベクトル形式で表現してハミング距離を取る方法や，ハミング距離で表現できない部分経路の一致も考慮した方法 [35] などが提案されている．しかし，IPSO では 2 つの巡回路の経路の一致率の逆数を距離と定義する．2 つの巡回路で経路が重複するほど距離が近いというこの定義は直感的にわかりやすい．経路の一致率を説明するために，辺を導入する．辺とは，2 つの都市を直接接続する道であり，途中に他の都市には滞在しない．例えば，辺 $e(1,2)$ は都市 1 と都市 2 を接続する道で，この 2 都市以外を間に含めない．辺を用いると，都市数 n の巡回路は n 本の辺により表現される．非対称 TSP の場合，$e(a,b)$ は都市 a から都市 b へ向かうことを意味している．しかし，対象 TSP の場合，方向は関係ないので，$e(a,b)$ は都市 a と都市 b を直接移動することを表す．つまり，$e(a,b) = e(b,a)$ が成り立つ．2 つの巡回路の辺の一致率 $S(\boldsymbol{a},\boldsymbol{b})$ を (4.2) 式に示す．

$$S(\boldsymbol{a},\boldsymbol{b}) = \frac{|E_{\boldsymbol{a}} \cap E_{\boldsymbol{b}}|}{n} \tag{4.2}$$

ここで，$E_{\boldsymbol{x}}$ は巡回路 $\boldsymbol{x}$ の辺集合，n は都市数を表す．$S(\boldsymbol{a},\boldsymbol{b})$ は $\boldsymbol{a}$ と $\boldsymbol{b}$ が一致する場合に 1 となり，すべて異なる辺で構成されている場合 0 となる．2 つの巡回路の辺の一致率

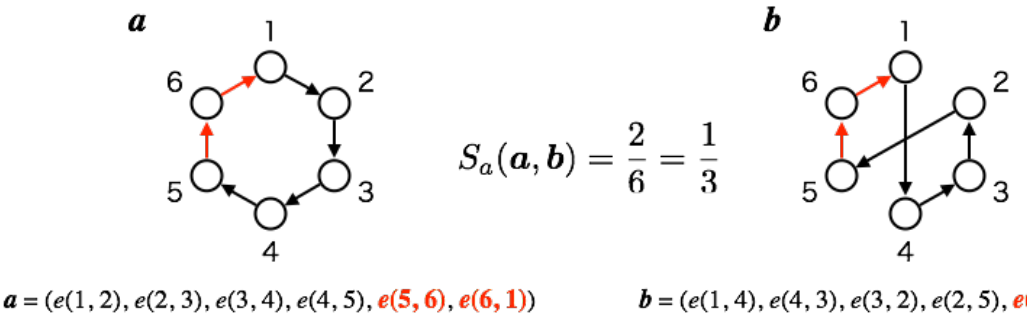


図 4.1　非対称 TSP における巡回路 $\boldsymbol{a}$，$\boldsymbol{b}$ の類似度 $S_a(\boldsymbol{a}, \boldsymbol{b})$．非対称 TSP においては移動方向も意味があるため，向きも一致する辺のみ対象となる．

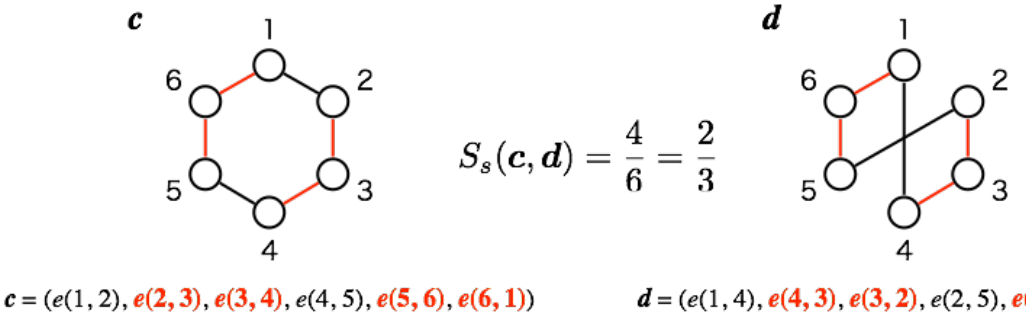


図 4.2　対称 TSP における巡回路 $\boldsymbol{c}$，$\boldsymbol{d}$ の類似度 $S_s(\boldsymbol{c}, \boldsymbol{d})$．対称 TSP では辺の向きは関係ない．向きの一致している辺に加え，逆向きの辺も対象となる．

の計算例を図 4.1，4.2 に示す．図 4.1，4.2 の巡回路 $\boldsymbol{a}$ と巡回路 $\boldsymbol{c}$，巡回路 $\boldsymbol{b}$ と巡回路 $\boldsymbol{d}$ は辺集合表記上では一致しているが，それぞれ非対称 TSP と対称 TSP の巡回路であるため，一致率の計算は異なることに注意されたい．

4.4　粒子群最適化からの変更点

提案手法 IPSO の PSO からの変更点を以下にまとめる．

- **粒子が持つ解の構造を実数値ベクトルから巡回路に変更**

  この変更により，粒子が巡回路を扱えるようになる．

- **解の更新を部分経路挿入により行うよう変更**

  解を巡回路表現にしたことにより，従来の方法では解の更新を行えなくなる．本

手法では，PSO の解の更新の戦略は“各粒子に *pbest* と *gbest* への引力を働かせることで，良い解の周りを集中的に探索し，短時間で高精度な解を得ること”と考える．これを巡回路で実現する最も単純な方法は，*pbest* と *gbest* の部分経路を解に挿入することである．

4.3 節で定義したように，2 つの巡回路の距離はそれらの辺の一致率の逆数で表現できるとすると，粒子の解を *pbest* や *gbest* に近づけるためには，*pbest* や *gbest* に含まれる辺を解に加えればよい．この場合，*pbest* や *gbest* からの辺の選び方は様々な方法があるが，TSP では 1 辺 1 辺よりパスが意味を成すことが多いことや，操作する辺が 2 辺だけで良いので計算コストが小さくて済むことを考慮すると部分経路挿入のメリットが大きいと考えられる．部分経路挿入の詳細については次節で述べる．

- **速度 v_i の廃止**

  PSO において速度とは，(3.2) 式から，$\boldsymbol{v}_i(t) = \boldsymbol{x}_i(t+1) - \boldsymbol{x}_i(t)$ より，現在と 1 単位時間前の位置の差である．PSO では解空間が n 次元の実数値ベクトル空間であるため，速度の持つ物理的な意味がわかりやすいが，解を巡回路に変更した IPSO では速度 v_i にあたる要素を定義することはできない．

4.5　部分経路の挿入

部分経路の挿入方法について，図 4.3 の例を用いて説明する．ここでは，図 4.3 のように都市 $1,2,3,4,5,6,7,8,9$ から成る 9 都市の対称 TSP を解いており，図の左上の解 $\boldsymbol{x}_i$ に対して，同じ図中の $\boldsymbol{p}_i$ と $\boldsymbol{g}$ の部分経路を挿入することを考える．まず，提案手法で用いる解表現について説明する．解 $\boldsymbol{x}_i = (1,2,3,4,5,6,7,8,9)$ は $1,2,3,4,5,6,7,8,9,1$ の順に都市を巡る実行可能解（巡回路）である．TSP では始点や終点となる都市の区別がないので，$(2,3,4,5,6,7,8,9,1)$ としても同じ巡回路を表す．さらに，この例では対称 TSP と呼ばれる，都市 a から都市 b までの距離と都市 b から都市 a までの距離が等しい TSP について考えているので，$\boldsymbol{x}_i$ の逆順である $(9,8,7,6,5,4,3,2,1)$ も同じ巡回路である．また，$e(a,b)$ を都市 a から都市 b までの他の都市を経由しない最短な道（辺）とすると，$\boldsymbol{x}_i^e = \{e(1,2), e(2,3), e(3,4), e(4,5), e(5,6), e(6,7), e(7,8), e(8,9), e(9,1)\}$ のように辺の集合としても表現できる．

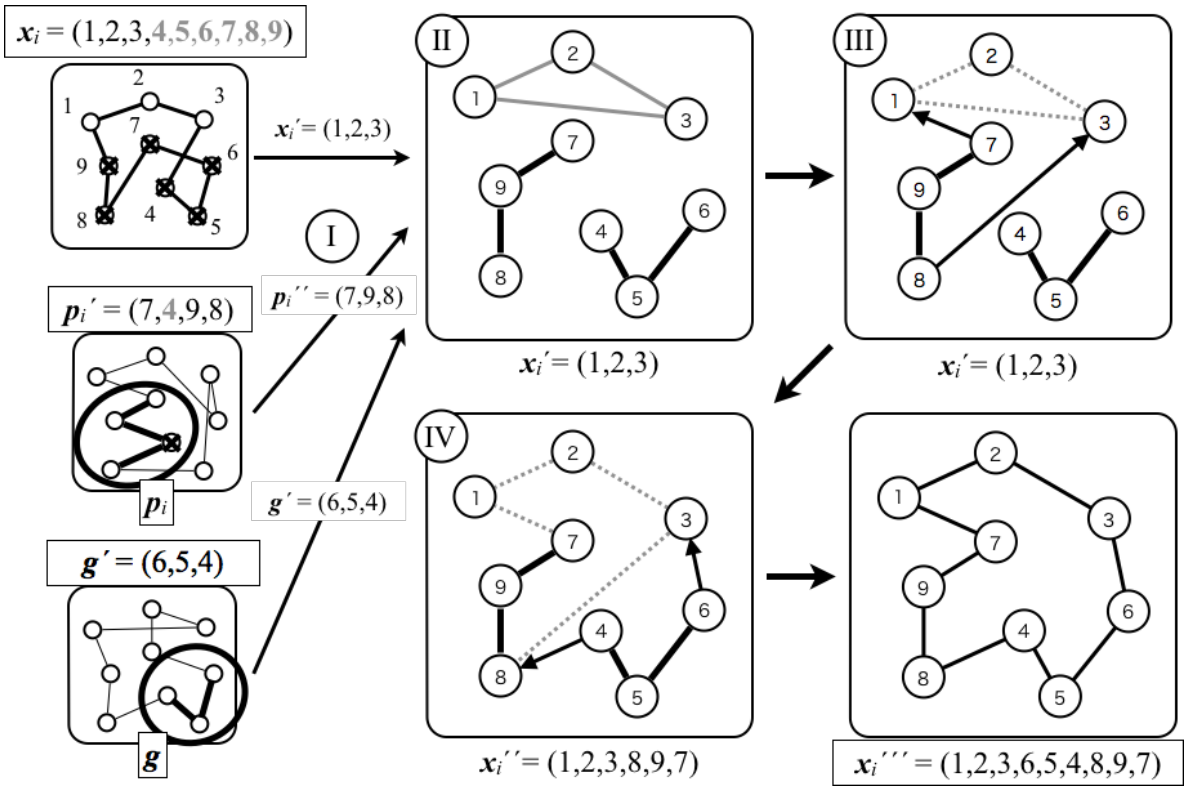


図 4.3　部分経路挿入の例

次に，部分経路の定義について説明する．部分経路とは巡回路中の連続した辺である．連続な辺とは，$\boldsymbol{x}_i$ を例に説明すると，$\{e(1,2), e(2,3), e(3,4)\}$ のような，部分辺集合内の都市が辺を介して互いに接続されている元の巡回路の辺集合 G の部分集合である．例えば，$\{e(1,2), e(4,5), e(7,8)\}$ は連続な辺ではないので部分経路ではない．これ以降，部分経路 $\{e(1,2), e(2,3), e(3,4)\}$ は $(1,2,3,4)$ のように訪問する都市の順序で表記する．この場合，巡回路と違い，辺 $e(4,1)$ は含まれない．

次に，選択された部分集合 $\boldsymbol{p}_i'$ と $\boldsymbol{g}'$ の挿入方法について説明する．巡回路 $\boldsymbol{a}$ に部分経路 $\boldsymbol{b}'$ を挿入するとき，$\boldsymbol{a}$ と $\boldsymbol{b}'$ の両方に含まれる都市が存在する場合，その都市を 2 度訪れることとなり，挿入後の巡回路は実行可能解とはならない．そこで，部分経路 $\boldsymbol{b}'$ に含まれる都市を巡回路 $\boldsymbol{a}$ から除いた閉路 $\boldsymbol{a}'$ に部分経路 $\boldsymbol{b}'$ を挿入することとする．例えば，巡回路 $\boldsymbol{a} = (1,2,3,4,5)$，部分経路 $\boldsymbol{b}' = (2,4)$ の場合，閉路 $\boldsymbol{a}' = (1,3,5)$ となる．辺で考えると，$e(1,2), e(2,3), e(3,4), e(4,5)$ が除去され，$e(1,3), e(3,5)$ が追加される．提案手法では，現在の粒子 i の解 $\boldsymbol{x}_i$ に，部分経路 $\boldsymbol{p}_i'$ と部分経路 $\boldsymbol{g}'$ を挿入する．2 つの部分

経路を挿入するが, $\boldsymbol{g}'$ に含まれる都市が $\boldsymbol{p}'_i$ にもある場合, $\boldsymbol{p}'_i$ から該当の都市を除去する．図 4.3 の例では，$\boldsymbol{p}'_i = (7, 4, 9, 8)$，$\boldsymbol{g}' = (6, 5, 4)$ であり都市 4 が共通して含まれているため，$\boldsymbol{p}'_i$ から都市 4 を除去し, $\boldsymbol{p}''_i = (7, 9, 8)$ としている．

最後に，部分経路挿入法のアルゴリズムを以下に示す．

1. $\boldsymbol{p}'_i$ から $\boldsymbol{g}'$ に含まれる都市を除いた部分経路 $\boldsymbol{p}''_i$ を生成する．(図中 I)
2. $\boldsymbol{x}_i$ から $\boldsymbol{p}''_i$ または $\boldsymbol{g}'$ に含まれる都市を除いた閉路 $\boldsymbol{x}'_i$ を生成する．(図中 II)
3. $\boldsymbol{x}'_i$ のすべての辺の中で，その辺の両端の都市の間に $\boldsymbol{p}''_i$ を挿入したときに最も総経路長が短くなるような辺を選択し，その辺に対して $\boldsymbol{p}''_i$ を挿入する．この操作により生成された閉路を $\boldsymbol{x}''_i$ とする．(図中 III)
4. $\boldsymbol{x}''_i$ 辺の中で $\boldsymbol{p}''_i$ に含まれていない辺に対して，その辺の両端の都市の間に $\boldsymbol{g}'$ を挿入したときに最も総経路長が短くなるような辺を選択し，その辺に対して $\boldsymbol{g}'$ を挿入する．(図中 IV)

対称 TSP において，手順 3 と 4 で部分経路を挿入するときに接続の仕方は 2 通りある．具体的には，選択された辺を $e(v_1, v_2)$，挿入する部分経路の両端の都市を v_3，v_4 とすると，$e(v_1, v_3)$，$e(v_4, v_2)$ と接続する場合と，$e(v_1, v_4)$，$e(v_3, v_2)$ と接続する場合である．本手法では，これら 2 通りのうち，総経路長が短くなる方のみを考慮している．

4.6　IPSO のアルゴリズム

部分経路の挿入を導入した PSO 戦略アルゴリズムを以下に示す．各粒子 i は巡回路（解）$\boldsymbol{x}_i$ と，粒子 i の *pbest* $\boldsymbol{p}_i$ をもつ．また，群全体で粒子 i の *gbest* $\boldsymbol{g}$ を共有している．n は都市数，c_1 および c_2 はそれぞれ $0 \leq c_1 \leq 1$，$0 \leq c_2 \leq 1$ を満たす定数，r_1，r_2 はそれぞれ $0 \leq c_1 \leq 1$，$0 \leq c_2 \leq 1$ の i に対して独立な乱数である．$[a]$ は実数 a の整数部を返す記号である．$C(\boldsymbol{x})$ は巡回路 $\boldsymbol{x}$ の総経路長を返す関数である．また，巡回路 $\boldsymbol{a}$ を巡回路 $\boldsymbol{b}$ に置き換える場合は代入記号を用いて $\boldsymbol{a} = \boldsymbol{b}$ と記述する．

1. すべての粒子 i の解 $\boldsymbol{x}_i$ に適当な巡回路を設定し，$\boldsymbol{p}_i = \boldsymbol{x}_i$ とする．$\boldsymbol{g}$ はすべての粒子の *pbest* の中で最も総経路長が小さい巡回路とする．数式では (4.3) 式のように定義される．

$$\boldsymbol{g} = \arg\min_{\boldsymbol{x} \in \boldsymbol{P}} C(\boldsymbol{x}) \tag{4.3}$$

ここで，m は粒子数であり，*pbest* 集合 $\boldsymbol{P}$ は $\boldsymbol{P} = \{\boldsymbol{p}_1, \boldsymbol{p}_2, \cdots, \boldsymbol{p}_m\}$ である．

2. すべての粒子 i について，次を適用する．
   (a) $\boldsymbol{p}_i$ から長さ $[c_1 r_1(n+1)]$ の連結成分をランダムに抽出し，部分経路 $\boldsymbol{p}'_i$ とする．
   (b) $\boldsymbol{l}_i$ から長さ $[c_2 r_2(n+1)]$ の連結成分をランダムに抽出し，部分経路 $\boldsymbol{g}'$ とする．
   (c) $\boldsymbol{x}_i$ に $\boldsymbol{p}'_i$ と $\boldsymbol{g}'$ を部分経路挿入法により挿入し，次の $\boldsymbol{x}_i$ とする．
   (d) $C(\boldsymbol{x}_i) < C(\boldsymbol{p}_i)$ ならば，$\boldsymbol{p}_i = \boldsymbol{x}_i$ とする．
3. (4.3) 式に従って $\boldsymbol{g}$ を更新し，それを暫定解 $\boldsymbol{s}$ とする．
4. 終了条件を満たしていなければ 2. へ戻る．
5. 暫定解 $\boldsymbol{s}$ を最終的に得られた解として返す．

アルゴリズム中で，2.，を 1 度実行することを 1 ステップとする．IPSO の N ステップ目とは，すべての $\boldsymbol{x}_i$，$\boldsymbol{p}_i$，$\boldsymbol{g}$ の更新を $N-1$ 回試みた状態を指す．アルゴリズムの終了条件は，N ステップ経過や制限時間経過などを設定する．

4.7　数値計算実験：IPSO のパラメータによる性能変化

4.7.1　本実験の目的

提案手法 IPSO には，粒子数 m，*pbest* の重み c_1，*gbest* の重み c_2 の 3 つの探索性能を決めるパラメータがある．本実験では，良い解を得るために設定すべき m，c_1，c_2 の値について調査する．

4.7.2　本実験における評価について

本実験では代表的なベンチマーク問題を扱っている TSPLIB[29, 30] からいくつかの TSP を使用する．TSPLIB に掲載されている TSP は最適解が既知であり，そのコストが公開されている．これらの TSP に提案手法 IPSO を適用し，得られた解のコストと最適解のコストを比較し，その誤差から提案手法の探索性能について議論する．誤差 $e_{\boldsymbol{s}}$ は次式で定義される．

$$e_{\boldsymbol{s}} = \frac{C(\boldsymbol{s}) - C(\boldsymbol{s}_o)}{C(\boldsymbol{s}_o)} \tag{4.4}$$

ここで，$\boldsymbol{s}$ は IPSO で得られた解，$\boldsymbol{s}_o$ は厳密解である．

4.7.3　IPSO の内部状態を議論するための特徴量

提案手法の内部状態を表現するための特徴量を定義する．

粒子の密度 $D_{\boldsymbol{x}}$

あるステップでの粒子の解の類似度の平均を粒子の密度 $D_{\boldsymbol{x}}$ とする．密度 $D_{\boldsymbol{x}}$ は次式で定義される．

$$D_{\boldsymbol{x}} = \frac{2}{m(m-1)} \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} S(\boldsymbol{x}_i, \boldsymbol{x}_j) \tag{4.5}$$

ここで，S は類似度関数であり，(4.2) 式で定義したものを使用する．$D_{\boldsymbol{x}}$ はすべての粒子の解が一致しているときに 1，1 つも共通する辺がないときに 0 となる．

$pbest$ の密度 $D_{\boldsymbol{p}}$

あるステップでの粒子の $pbest$ の類似度の平均を粒子の密度 $D_{\boldsymbol{p}}$ とする．密度 $D_{\boldsymbol{p}}$ は次式で定義される．

$$D_{\boldsymbol{p}} = \frac{2}{m(m-1)} \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} S(\boldsymbol{p}_i, \boldsymbol{p}_j) \tag{4.6}$$

$D_{\boldsymbol{p}}$ は，$D_{\boldsymbol{x}}$ と同様に，すべての粒子の $pbest$ が一致しているときに 1，1 つも共通する辺がないときに 0 となる．

粒子と $pbest$ の類似度 $S_{\boldsymbol{xp}}$

あるステップでの粒子の解と $pbest$ の類似度の平均を粒子と $pbest$ の類似度 $S_{\boldsymbol{xp}}$ とする．$S_{\boldsymbol{xp}}$ は次式で定義される．

$$S_{\boldsymbol{xp}} = \frac{1}{m} \sum_{i=1}^{m} S(\boldsymbol{x}_i, \boldsymbol{p}_i) \tag{4.7}$$

$S_{\boldsymbol{xp}}$ はすべての粒子の解がその粒子の $pbest$ と一致していれば 1，粒子の解が $pbest$ と 1 つも共通する辺がないとき 0 となる．

表 4.1　実験環境

CPU	Intel Xeon E5520(2.26GHz) × 2
OS	Fedora 20
使用言語	Scala 2.10.4, Java 1.7.0

粒子と *gbest* の類似度 $S_{\boldsymbol{xg}}$

あるステップでの粒子と *gbest* の類似度の平均を粒子と *gbest* の類似度 $S_{\boldsymbol{xg}}$ とする．$S_{\boldsymbol{xg}}$ は次式で定義される．

$$S_{\boldsymbol{xg}} = \frac{1}{m}\sum_{i=1}^{m} S(\boldsymbol{x}_i, \boldsymbol{g}) \tag{4.8}$$

$S_{\boldsymbol{xg}}$ はすべての粒子の粒子が *gbest* と一致していれば 1，粒子の解が *gbest* と 1 つも共通する辺がないとき 0 となる．

***pbest* と *gbest* の類似度 $S_{\boldsymbol{pg}}$**

あるステップでの *pbest* と *gbest* の類似度の平均を *pbest* と *gbest* の類似度 $S_{\boldsymbol{pg}}$ とする．$S_{\boldsymbol{pg}}$ は次式で定義される．

$$S_{\boldsymbol{pg}} = \frac{1}{m}\sum_{i=1}^{m} S(\boldsymbol{p}_i, \boldsymbol{g}) \tag{4.9}$$

$S_{\boldsymbol{pg}}$ はすべての粒子の *pbest* が *gbest* と一致していれば 1，*gbest* と一致している *pbest* が 1 つで，それ以外の粒子の *pbest* が *gbest* と 1 つも共通する辺がないとき $1/m$ となる．

4.7.4　実験環境

これ以降の実験で共通する環境について表 4.1 に示す．メモリは記載していないが，使用するすべての TSP においてコスト行列をメモリ上に確保できる容量を用いている．また，CPU を 2 個搭載したコンピュータを用いており，8 コア 16 スレッドであるが，本実験では 1 つの IPSO は 1 つの CPU のみを使って動作する．

4.7.5　予備実験：IPSO の特徴量と解の収束の関係性の調査

4.7.2 節において，IPSO の特徴量として，粒子の密度 $D_{\boldsymbol{x}}$，*pbest* の密度 $D_{\boldsymbol{p}}$，粒子と *pbest* の類似度 $S_{\boldsymbol{xp}}$，粒子と *gbest* の類似度 $S_{\boldsymbol{xg}}$，*pbest* と *gbest* の類似度 $S_{\boldsymbol{pg}}$ を定義した．本実験では，これらの特徴量が解の収束とどのように関係しているか調査する．

本実験の実験設定を示す．TSP は 100 都市問題である kroA100 を使用する．パラメータについては本実験以降で詳細に調査するので，ここでの吟味は行わない．本実験では $(c_1, c_2) = (0.3, 0.3)$ を用いる．得られた解の誤差や特徴量については，同実験を 100 試行し，その平均値を用いる．また，探索初期と探索全体の両方について比較したいので，終了ステップを 1000 とした実験と 40000 とした実験の 2 つを行った．これらの実験は独立に 100 試行ずつ行う．終了が 1000 ステップの実験では 50 ステップごとに現在の解精度と特徴量を記録し，40000 ステップの実験では 2000 ステップごとに記録する．

実験結果を図 4.4，4.5 に示す．図 4.4 は得られた解の平均誤差 $\bar{e_{\boldsymbol{s}}}$ の推移である．左図が 1000 ステップまでの結果を 50 ステップごとに記録した結果で，右図が 40000 ステップまでの結果を 2000 ステップごとに記録した結果である．横軸がステップ数，縦軸が平均誤差を示す．なお，縦軸は見やすさの観点から対数軸で表示している．図 4.5 は，IPSO の特徴量である，粒子の密度 $D_{\boldsymbol{x}}$，*pbest* の密度 $D_{\boldsymbol{p}}$，粒子と *pbest* の類似度 $S_{\boldsymbol{xp}}$，粒子と *gbest* の類似度 $S_{\boldsymbol{xg}}$，*pbest* と *gbest* の類似度 $S_{\boldsymbol{pg}}$ のそれぞれの平均値の推移である．

まず，図 4.4 の解の平均誤差 $\bar{e_{\boldsymbol{s}}}$ の推移について確認する．グラフの概形から，探索初期に高速に解の誤差が小さくなり，400 ステップ以降はゆっくりと少しずつ解が更新されている．また，平均的にではあるが，誤差 10% の解が約 250 ステップ，誤差 1% の解が約 18000 ステップで得られている．

次に，図 4.5 の IPSO の特徴量の推移について確認する．まず，初期状態について説明する．IPSO の各粒子の解はランダムに初期化されているため，粒子の密度 $D_{\boldsymbol{x}}$ はほぼ 0 となる．初期状態の *pbest* はその粒子の初期解に一致するので，粒子と *pbest* の類似度 $S_{\boldsymbol{xp}}$ は 1 となり，*pbest* の密度 $D_{\boldsymbol{p}}$ は $D_{\boldsymbol{p}} = D_{\boldsymbol{x}}$ となる．粒子と *gbest* の類似度 $S_{\boldsymbol{xg}}$ と，*pbest* と *gbest* の類似度 $S_{\boldsymbol{pg}}$ は，上記の理由から，$S_{\boldsymbol{xg}} = S_{\boldsymbol{pg}}$ となる．*pbest* は少なくとも 1 つは *gbest* と一致するため，$S_{\boldsymbol{xg}}$ と $S_{\boldsymbol{pg}}$ は $D_{\boldsymbol{p}}$ と $D_{\boldsymbol{x}}$ より少し高い値となる．次に，図 4.5 の左図について説明する．各特徴量は徐々に高い値に収束していくことがわかる．$S_{\boldsymbol{xp}}$ は初期状態では 1 であったが，その後，解が部分経路挿入によって更新されたため，一旦小さくなる．しかし，他の特徴量と同様に高い値となる．それぞれの特徴量を比較す

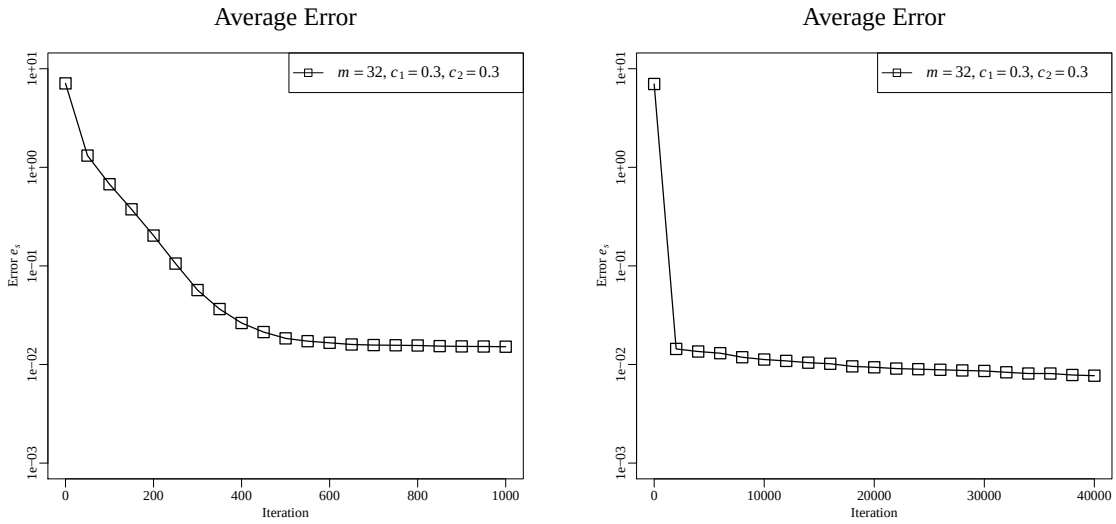


図 4.4　IPSO の解精度の推移

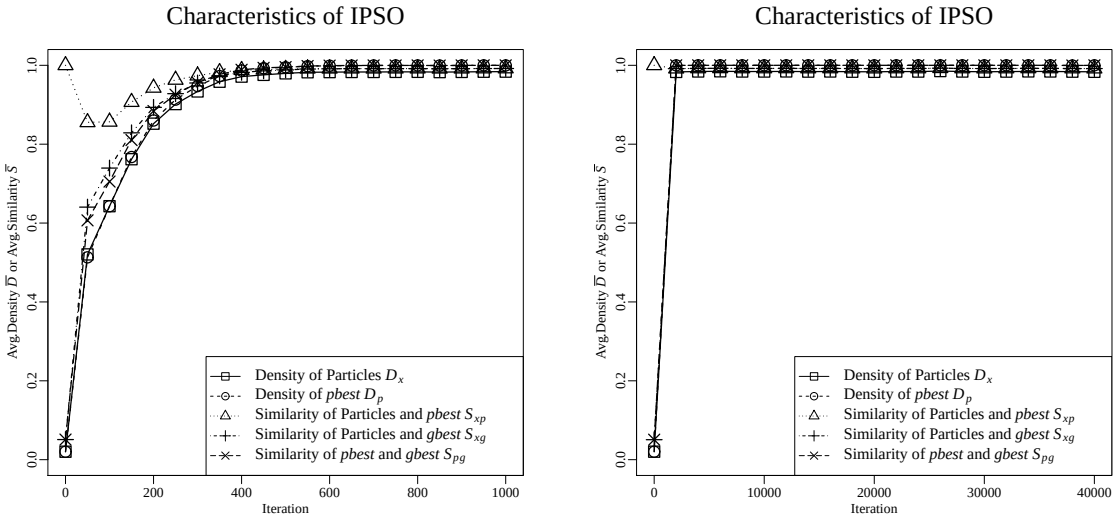


図 4.5　IPSO の特徴量の推移

ると，S_{xp} が最も高く，次に S_{xg}，S_{pg} と続く．D_p と D_x は，序盤は同じぐらいの値だが，探索が進むと次第に D_p が高くなる．これは，図 4.5 の左図の一部を拡大した，図 4.6 からよりはっきりわかる．また，図 4.5 の右図から，さらに探索を進めても，特徴量はほぼ 1 のままであることがわかる．

図 4.4 の左図と図 4.5 の左図を比較すると，各特徴量がほぼ 1 となったところで，解の

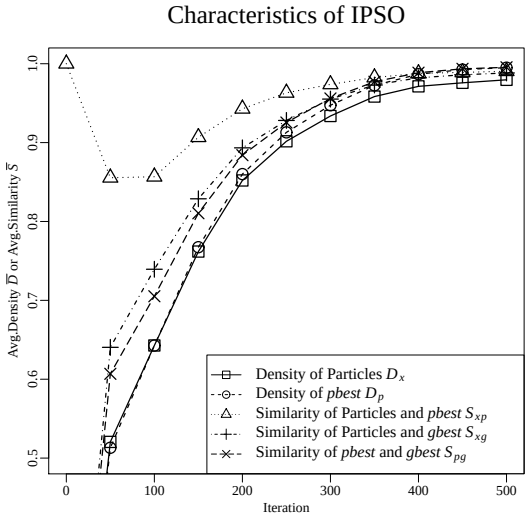


図 4.6　図 4.5 左図の拡大図．ステップが 0 から 500，特徴量の値は 0.5 から 1.0 の範囲で描画している．

更新速度が緩やかになっていることがわかる．また，図 4.4 の右図と図 4.5 の右図の比較から，特徴量はほぼ 1 になっても，解は緩やかに更新されていることも示されている．

予備実験の結果のまとめと考察

予備実験の結果を簡単にまとめた上で考察を行う．予備実験から次の 2 点が明らかになった．

- 探索が進むにつれて，IPSO の特徴量は 1 に収束していく．
- IPSO の特徴量がほぼ 1 になると，解の更新が緩やかになる．

これらの結果は，IPSO のアルゴリズムから予想できる．まず，探索が進むにつれて，IPSO の特徴量は 1 に収束していく現象について考える．IPSO は *pbest* や *gbest* からの部分経路挿入により解を更新していく．この部分経路挿入によって，$S_{\boldsymbol{xp}}$ と $S_{\boldsymbol{xg}}$ は多くの場合上昇する．また，この解の更新により，*pbest* や *gbest* が更新されると，その粒子の解と *pbest* や *gbest* の類似度は 1 となる．一方，粒子と *pbest* の類似度が下がるケースは，粒子に *pbest* が挿入されず，*gbest* の辺のうち，*pbest* に含まれない辺が解に挿入されるときである．これは自身の *pbest* が *gbest* であるときには起こらない現象である．また，

粒子と *gbest* の類似度が下がるケースは，上記の逆の場合か，もしくは *gbest* が他の粒子によって更新されたときである．つまり，S_{xp} と S_{xg} を比較的低く保てているときは，*gbest* が様々な粒子から更新されていて，なおかつ，D_x が低い場合であるとわかる．各粒子をランダムな巡回路で初期化した場合，探索初期はこの条件を満たしやすいことは明らかである．しかし，*gbest* の更新の頻度が小さくなってくると，類似度を下がる操作が行われにくくなるので，S_{xp} と S_{xg} は 1 に収束していく．また，*gbest* の更新の頻度が小さくなってくると，すべての粒子が *gbest* に近づく．これは，すべての粒子が *gbest* の部分経路を各ステップで必ず行うからである．このことにより，D_x や D_p が高くなってくる．上記の考察は図 4.5，4.6 に示した結果と合致する．

次に，IPSO の特徴量がほぼ 1 になると，解の更新が緩やかになる現象について考える．そもそも，すべての特徴量が 1 の状態とは，すべての粒子の解と *pbest* と *gbest* が一致している状態である．この場合，部分経路挿入は自身の部分経路の再挿入となる．例を図 4.7 に示す．この例では，自身の部分経路 1 つを再挿入する場合について扱う．アルゴリズム的には，*pbest* か *gbest* のどちらか一方のみ挿入される場合と考えられる．この例では，赤で示した部分経路が選択され，一旦，それ以外の都市で部分巡回路を構築してから，コストが最も小さくなるように赤の部分経路を挿入している．このとき，コストが最小化されたのは赤で示した部分経路の両端の 2 辺で，実際に変更された辺は 3 箇所である．つまり，限定された 3-opt 操作と呼ぶことができる．また，*pbest* と *gbest* の両方が部分経路挿入される場合，最大で 6-opt となる．このように，仮にすべての特徴量が 1 になった場合，遷移する状態の種類は限定されるが，探索が停止することはない．ただし，このような更新を繰り返したとしても，*pbest* や *gbest* を更新できなければ，同じ巡回路に引き寄せられていることには変わりないので，*pbest* や *gbest* から構造が大きく違う解は発見されにくい．

予備実験の結論

本実験では，IPSO の特徴量と解の収束の関係について実験を行った．その結果，探索が進むにつれて特徴量は 1 に収束していき，特徴量が 1 付近になると解が収束することがわかった．これは，特徴量がほぼ 1 のときは，現在の最良解から大きく構造が異なる解を発見できにくいアルゴリズムの特徴が理由である．以上の結果から，IPSO の収束は特徴量の値からも観測できる．

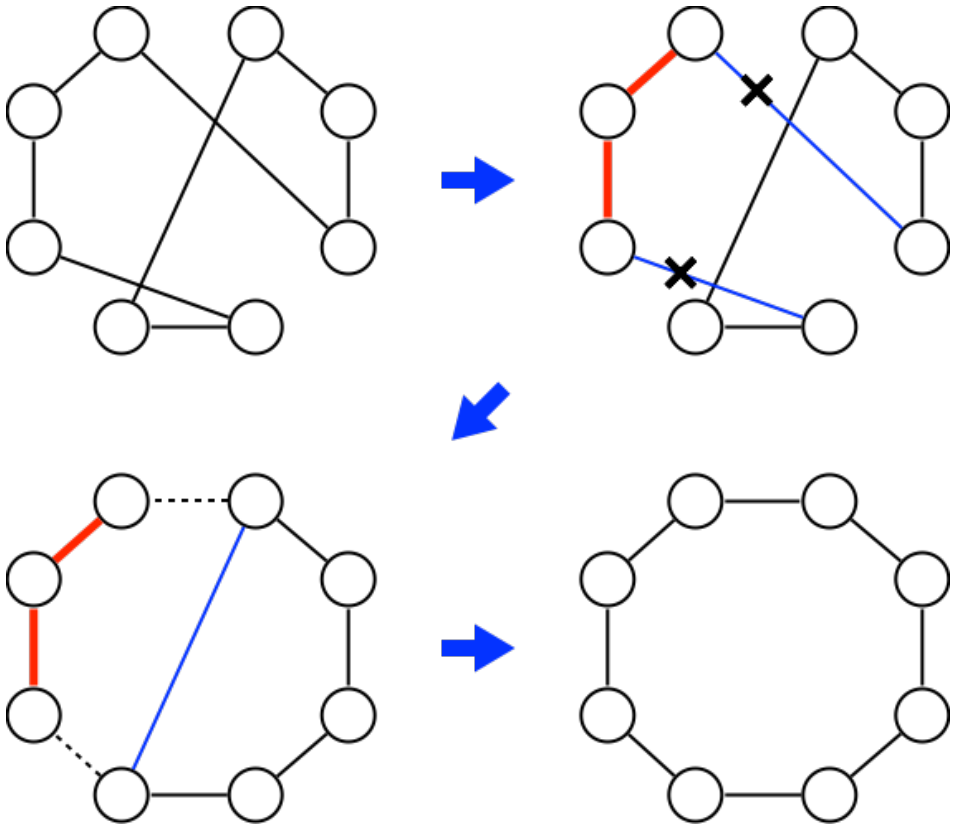


図 4.7　IPSO のすべての特徴量が 1 である場合の部分経路挿入の動作例（自身の部分経路 1 つを再挿入する場合）.

表 4.2　パラメータ (c_1, c_2) が探索に及ぼす影響の調査で用いるパラメータ

粒子数 m	32		
pbest の重み c_1	$\{0, 0.05, 0.1, 0.15, 0.2, \cdots, 1.0\}$		
gbest の重み c_2	$\{0, 0.05, 0.1, 0.15, 0.2, \cdots, 1.0\}$		
使用する TSP	kroA100	kroA200	pr439
終了ステップ数	$\{2000, 20000\}$	60000	100000
パラメータセットあたりの試行回数	100		

4.7.6　パラメータ (c_1, c_2) が探索に及ぼす影響の調査

まず，c_1 と c_2 が探索に及ぼす影響について調査する．実験設定を表 4.2 に示す．パラ

メータ (c_1, c_2) は実験に使用する c_1 と c_2 のすべての組合せについて行う．つまり，c_1 の集合を C_1，c_2 の集合を C_2 とすると，$(c_1, c_2) \in C_1 \times C_2$ である．なお，これ以降，単に (x, y) と記述したときは，$c_1 = x$，$c_2 = y$ という設定であることを意味する．使用する TSP は 100 都市問題（kroA100），200 都市問題（kroA200），439 都市問題（pr439）を用い，問題規模に応じて有効なパラメータが異なるか調査する．IPSO の探索は一定ステップ探索を終えるまで行い，kroA100 では 20000 ステップ，kroA200 は 60000 ステップ，pr439 では 100000 ステップ探索を行う．このとき解は十分に収束することを確認している．なお，kroA100 に関しては，探索初期と探索全体の両方について議論するためである．終了が 2000 ステップの実験も行う．探索の推移を議論するために，実験中は一定ステップごとにその時点の解精度と特徴量を記録する．具体的には，終了が 2000 ステップの実験では 100 ステップごと，20000 ステップの実験では 1000 ステップごと，60000 ステップの実験では 3000 ステップごと，100000 ステップの実験では 5000 ステップごとに記録する．得られた解の誤差の議論は，パラメータセットごとに 100 試行し，その平均値，最小値，最大値について行う．また，100 試行の内で厳密解が得られた試行の割合についても調査する．

最終的に得られた解について

まず，kroA100 に IPSO を 20000 ステップ適用し得られた解の平均誤差 $\bar{e_{\boldsymbol{x}}}$ を図 4.8，kroA200 に IPSO を 60000 ステップ適用し得られた解の平均誤差 $\bar{e_{\boldsymbol{x}}}$ を図 4.9，pr439 に IPSO を 100000 ステップ適用し得られた解の平均誤差 $\bar{e_{\boldsymbol{x}}}$ を図 4.10 に示す． 図 4.8 から図 4.10 は各パラメータセットごとに最終的に得られた解の平均誤差 $\bar{e_{\boldsymbol{s}}}$ をヒートマップで表示している．横軸が *pbest* の重み c_1 の値，縦軸が *gbest* の重み c_2 の値を示している．色の濃淡は $\bar{e_{\boldsymbol{s}}}$ の高低を表しており，見やすさの観点から常用対数をとって表示している．得られた解が厳密解に近いほど白となり，解が悪いと黒になる．図 4.8 から，解の精度が高かったパラメータは $0.4 \leq c_1 \leq 0.9$，$c_2 = 0.05$ であるとわかる．c_2 が大きな値の場合は精度はあまり良くならず，なおかつ，c_1 によらず，ほぼ一定である．一方，図 4.9，図 4.10 も図 4.8 と同様に，$0.4 \leq c_1 \leq 0.9$，$c_2 = 0.05$ の範囲のパラメータの場合に良い解が得られることが示された．この結果から，解が十分に収束するまで探索を行う場合，解の平均誤差 $\bar{e_{\boldsymbol{x}}}$ を小さくするために有効なパラメータ値は問題規模に影響されないことがわかる．さらに調査した結果，平均誤差 $\bar{e_{\boldsymbol{x}}}$ 以外の結果についても同様であることがわかったため，以下の実験では 3 種類の TSP のうち，kroA100 に関してのみ議論する．

次に，得られた解の最小値と最大値について図 4.11 に示す．図 4.11 の左図が $e_{\boldsymbol{s}}$ の

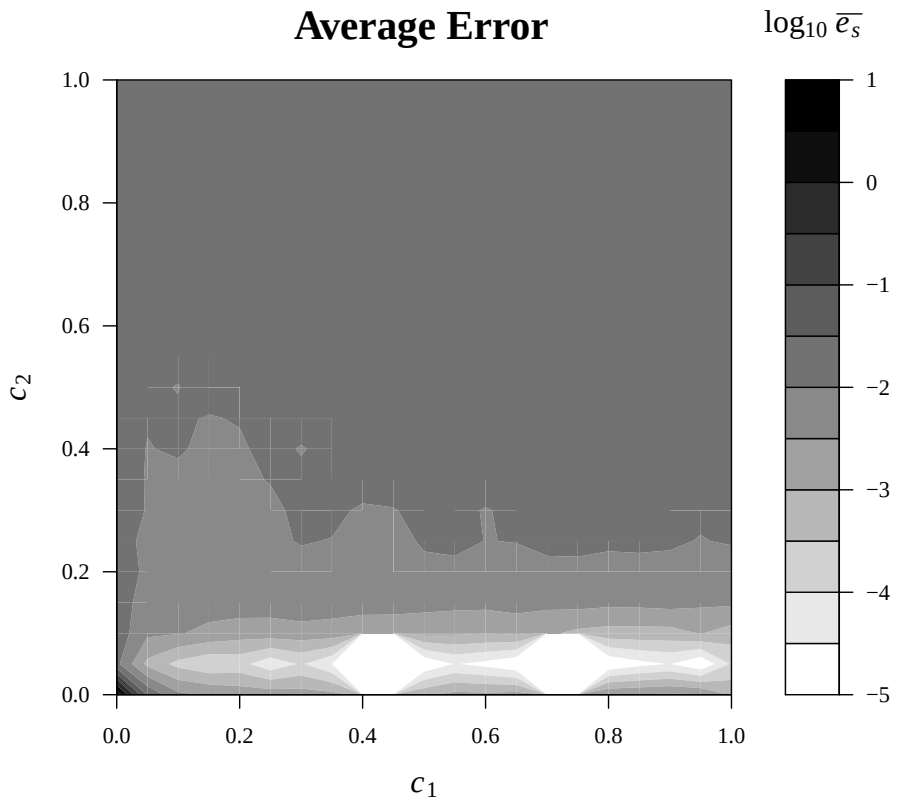


図 4.8　kroA100 に対し IPSO を 20000 ステップ適用し得られた解の平均誤差 $\overline{e_{\boldsymbol{x}}}$.

最小値，右図が $e_{\boldsymbol{s}}$ の最大値を示している．図 4.11 の左図から，ほぼすべてパラメータにおいて $e_{\boldsymbol{s}}$ の最小値はとても小さいことがわかる．データを調査すると，パラメータセット (c_1, c_2) が $(0.0, 0.0)$ と $(0.05, 0.0)$ 以外のものは，少なくとも 1 度は最適解が見つかっている．一方，最大値を示している図 4.11 の右図は，概形が図 4.8 に似ており，$0.4 \leq c_1 \leq 0.9$，$c_2 = 0.05$ において値が小さい．

次に，100 試行の中で最適解が得られた割合を図 4.12 に示す．色の濃度が厳密解との一致率を示しており，一致率が 1 に近いほど黒く，0 に近いほど白い．各試行における一致率 S_o は，20000 ステップ適用後に得られた解を $\boldsymbol{s}$，厳密解を $\boldsymbol{s}_o$ とすると，次式で定義される．

$$S_o = \begin{cases} 1 & if\ S(\boldsymbol{s}, \boldsymbol{s}_o) = 1 \\ 0 & else \end{cases} \tag{4.10}$$

つまり，巡回路中の辺の一致率ではなく，巡回路が完全一致した場合のみ 1 を返す．図

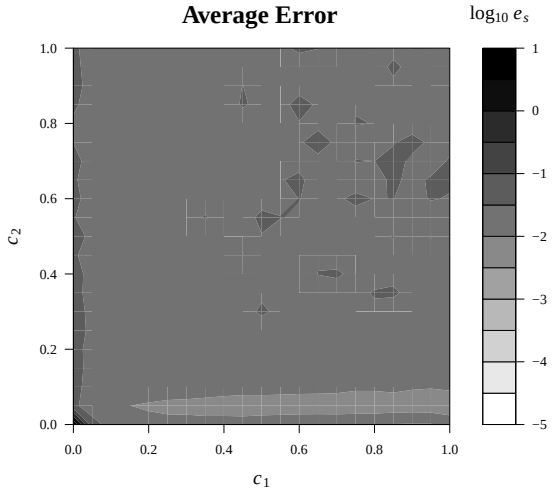


図 4.9　kroA200 に対し IPSO を 60000 ステップ適用し得られた解の平均誤差 $\bar{e_{\boldsymbol{x}}}$.

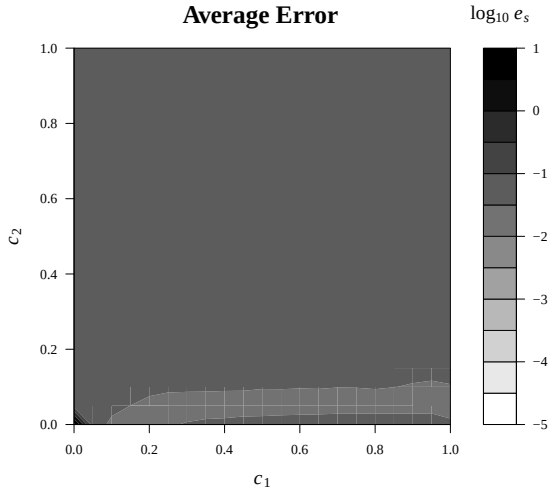
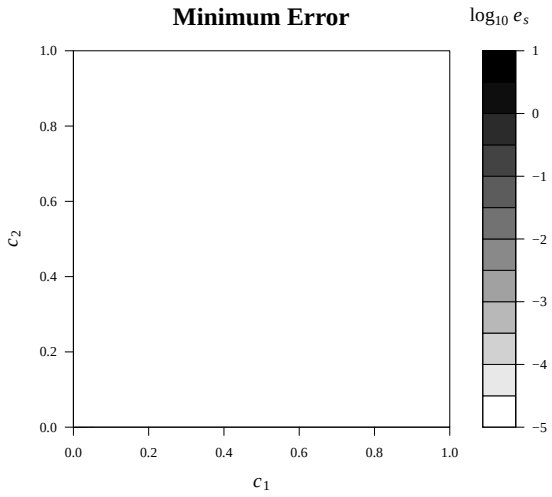


図 4.10　pr439 に対し IPSO を 100000 ステップ適用し得られた解の平均誤差 $\bar{e_{\boldsymbol{x}}}$.



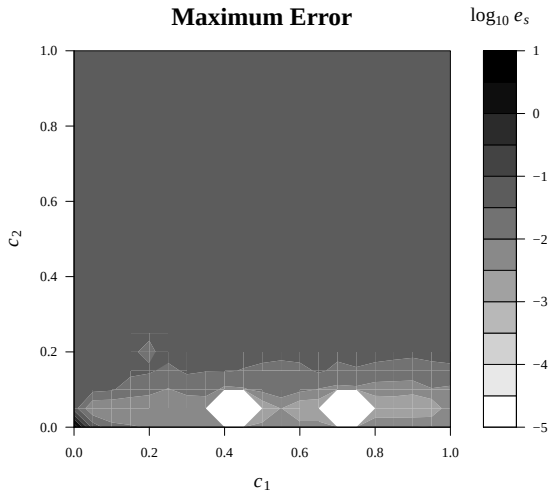


図 4.11　kroA100 に対し IPSO を 20000 ステップ適用し得られた解の誤差 $e_{\boldsymbol{x}}$ の最小値（左図）と最大値（右図）.

4.12 の結果から，$c_2 = 0.05$ のときに一致率が高いことがわかる．その他のパラメータについては，これまでの結果と同様に，一致率は低い．

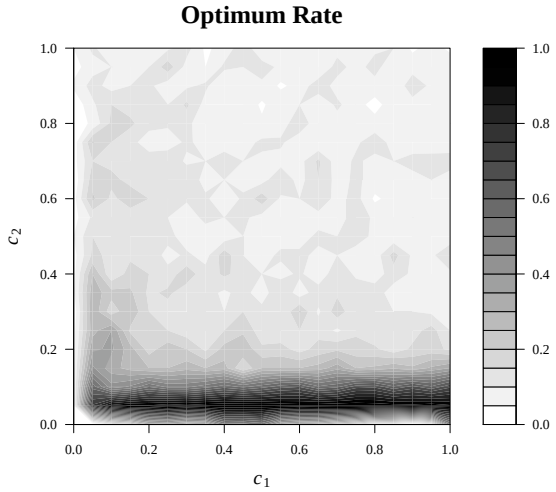


図 4.12　kroA100 に対し IPSO を 20000 ステップ適用し得られた解が厳密解と一致した割合.

探索の推移について

上記の結果から，$0.4 \leq c_1 \leq 0.9$，$c_2 = 0.05$ の場合に，つまり，c_1 が大きく，c_2 が小さい値のときに良い解が得られることがわかった．次に，そのパラメータ値とそれ以外の値で探索がどのように違うか検証する．調査するパラメータセット (c_1, c_2) は，上記の結果で最も良かった (0.7,0.05) の他に，c_1 と c_2 の両方が小さい (0.05,0.05)，c_1 と c_2 の両方が大きい (0.7,0.7)，c_1 が小さく，c_2 が大きい (0.05,0.7) の 4 つとする．

実験結果を図 4.13 から図 4.18 に示す．図 4.13 は平均誤差 $\bar{e_s}$ の推移を示している．図 4.13 の左図は 2000 ステップまでの $\bar{e_s}$，右図は 20000 ステップまでの $\bar{e_s}$ を示している．右図の (0,7,0.05) の結果が 17000 ステップ以降表示されていないが，これは 17000 以降で厳密解が得られているために，$\bar{e_s} = 0$ となり，対数軸表示できなくなっているためである．20000 ステップ終了時の $\bar{e_s} = 0$ は，図 4.8 にある通り，(0.7,0.05) が最も小さく，次に (0.05,0.05)，(0.05,0.07)，(0.7,0.7) の順に小さい．この順番は 2000 ステップ以降は変わっていないが，1000 ステップ以前は異なることが示されている．左図に示されている，探索初期における $\bar{e_s}$ に着目すると，(0.05,0.7) が最も初期の探索性能がよいことがわかる．例えば，平均誤差が 10% ，つまり，$\bar{e_s} = 0.1$ となるために必要なステップ数は，(0.05,0.7) が最も小さく，次に (0.7,0.7) が小さく，その次に同じステップで (0.05,0.05)，(0.7,0.05) となっている．以上から，終了ステップ数において，最適な (c_1, c_2) は異なる

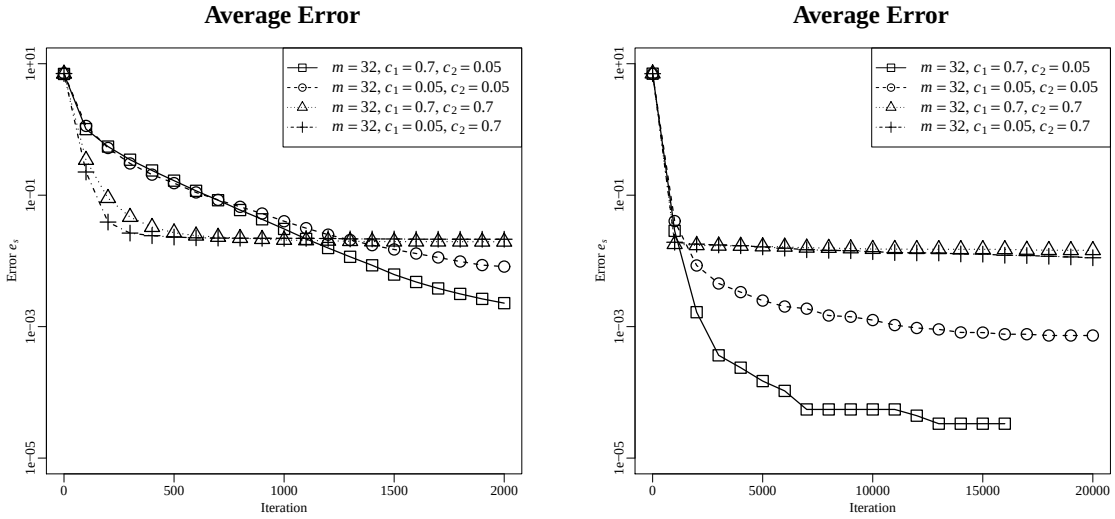


図 4.13　kroA100 に対し IPSO を適用し得られた解の平均誤差 $\bar{e_{\boldsymbol{x}}}$．左図は 2000 ステップまで．右図は 20000 ステップまで．

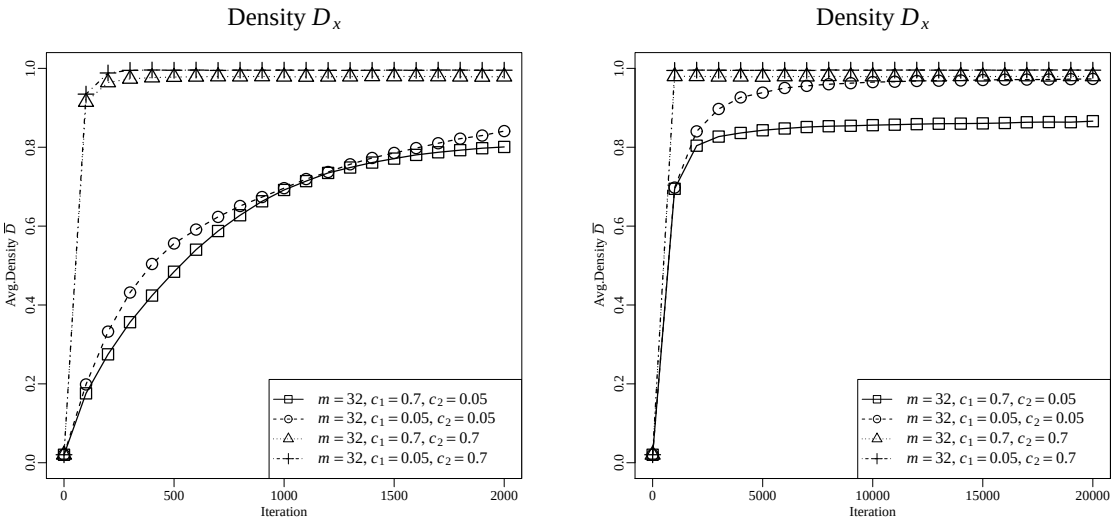


図 4.14　kroA100 に対し IPSO を適用したときの粒子の密度 $D_{\boldsymbol{x}}$．左図は 2000 ステップまで．右図は 20000 ステップまで．

ことが示された．

図 4.14 から図 4.18 までは，IPSO の特徴量の推移を示している．それぞれの左図は

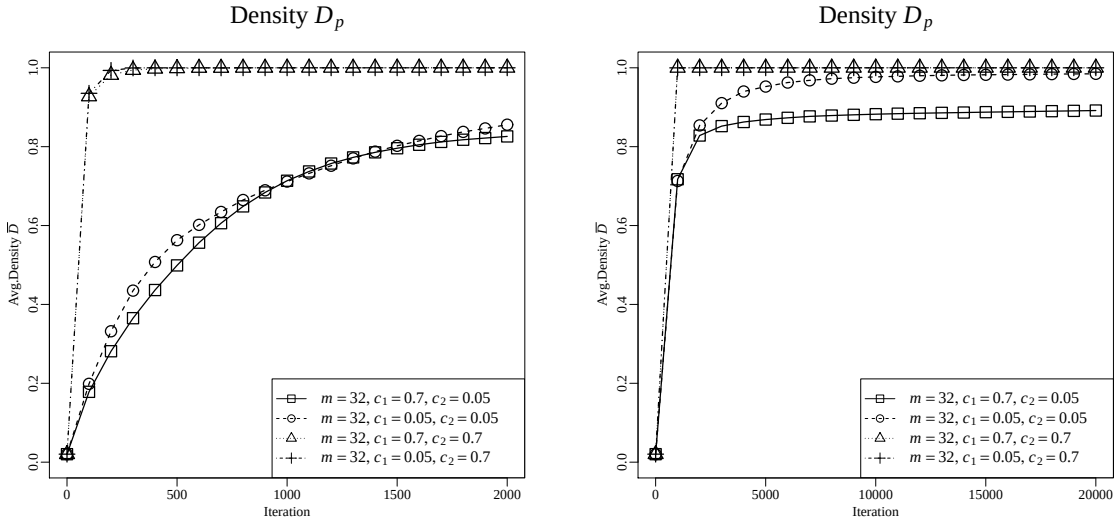


図 4.15　kroA100 に対し IPSO を適用したときの *pbest* の密度 $D_{\boldsymbol{p}}$．左図は 2000 ステップまで．右図は 20000 ステップまで．

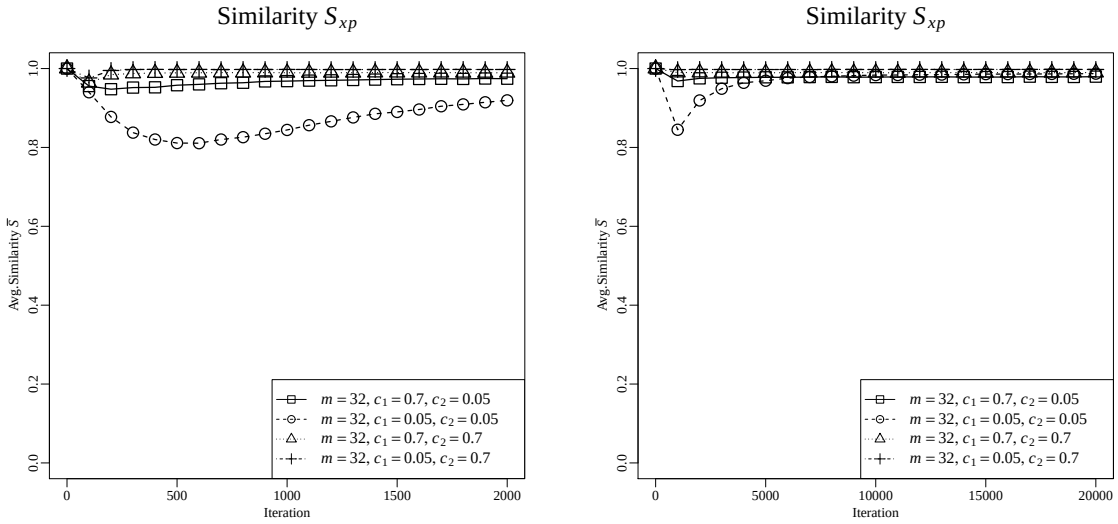


図 4.16　kroA100 に対し IPSO を適用したときの粒子と *pbest* の類似度 $S_{\boldsymbol{xp}}$．左図は 2000 ステップまで．右図は 20000 ステップまで．

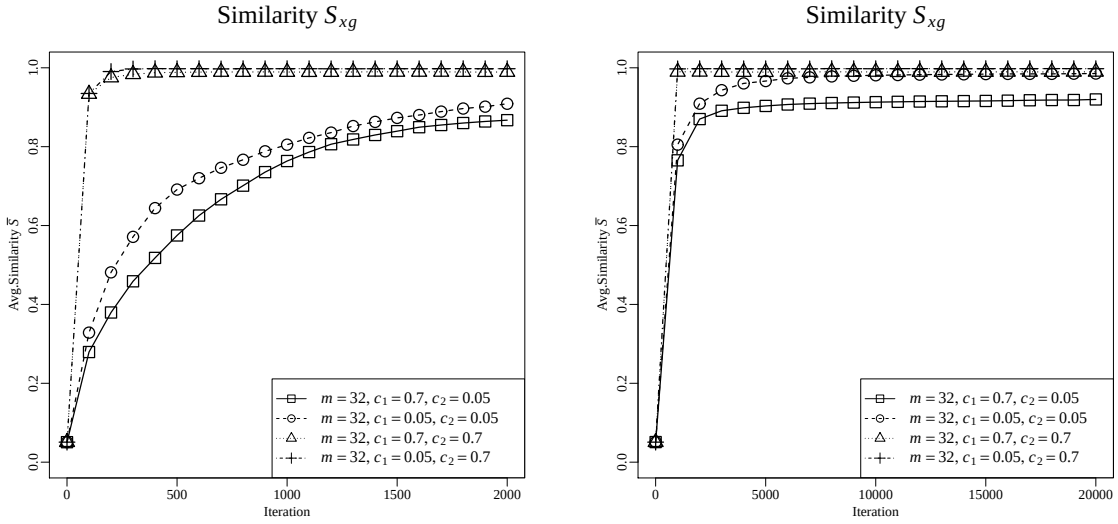


図 4.17　kroA100 に対し IPSO を適用したときの粒子と *gbest* の類似度 S_{xg}．左図は 2000 ステップまで．右図は 20000 ステップまで．

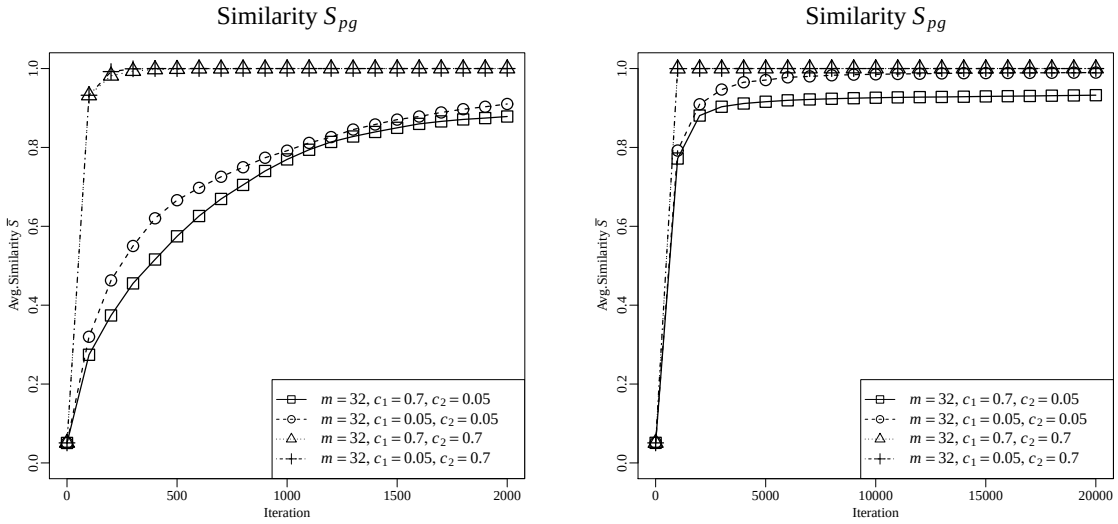


図 4.18　kroA100 に対し IPSO を適用したときの *pbest* と *gbest* の類似度 S_{pg}．左図は 2000 ステップまで．右図は 20000 ステップまで．

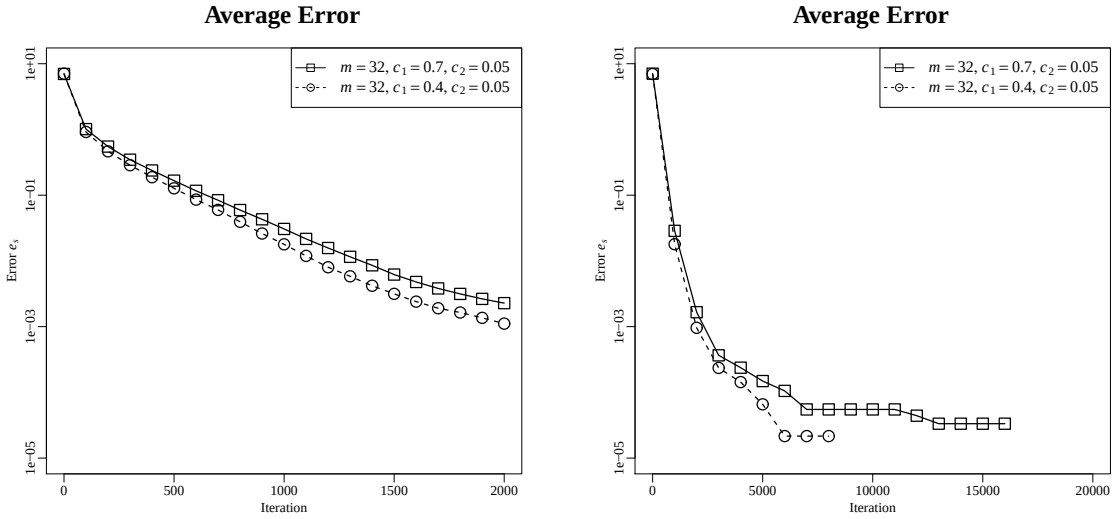


図 4.19　kroA100 に対し IPSO を適用し得られた解の平均誤差 $\bar{e_{\boldsymbol{x}}}$．左図は 2000 ステップまで．右図は 20000 ステップまで．

2000 ステップまでの $\bar{e_{\boldsymbol{s}}}$，右図は 20000 ステップまでの $\bar{e_{\boldsymbol{s}}}$ を示している．まず，$D_{\boldsymbol{x}}$，$D_{\boldsymbol{p}}$，$S_{\boldsymbol{xg}}$，$S_{\boldsymbol{pg}}$ に着目する．これらの特徴量は探索過程のすべてにおいて (0.7,0.05) のときに最も小さい．その次に (0.05,0.05)，(0.7,0.7)，(0.05, 0.7) の順番で小さい．また，2000 ステップまでは (0.7,0.05) と (0.05,0.05) の結果が似ていて，(0.7,0.7) と (0.05, 0.7) の結果が似ている．一方，2000 ステップより後は (0.05,0.05)，(0.7,0.7)，(0.05, 0.7) の結果がほぼ 1 なのに対し，(0.7,0.05) の結果がだけ少し小さい．次に，$S_{\boldsymbol{xp}}$ に着目する．図 4.16 から，(0.05,0.05) の $S_{\boldsymbol{xp}}$ が最も小さく，次に (0.7,0.05)，(0.7,0.7)，(0.05,0.7) の順番で $S_{\boldsymbol{xp}}$ が小さい．また，3000 ステップ以降は，すべてのパラメータでほぼ $S_{\boldsymbol{xp}} = 1$ となる．

(0.7,0.05) と (0.4,0.05) の比較

図 4.13 から図 4.18 にパラメータ c_1 と c_2 の大小を変えた 4 種類のパラメータセットに関して詳細な結果を示した．その結果，c_1 を大きく，c_2 を小さく設定したときに良い解が得られることが示された．しかし，図 4.8 から，良い解が得られる c_1 の範囲は $[0.4, 0.9]$ であった．そこで，(0.7,0.05) と (0.4,0.05) の探索過程の比較を行い，c_1 と解精度の関係性をさらに調べる．

実験結果を図 4.19 から図 4.21 に示す．図 4.19 の左図は 2000 ステップまでの $\bar{e_{\boldsymbol{s}}}$，右

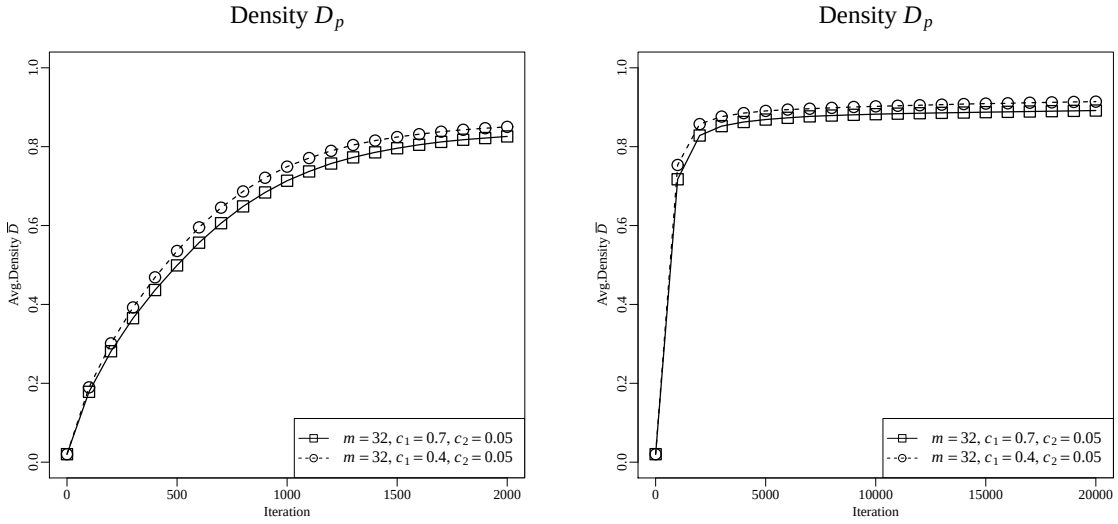


図 4.20　kroA100 に対し IPSO を適用したときの *pbest* の密度 $D_{\boldsymbol{p}}$．左図は 2000 ステップまで．右図は 20000 ステップまで．

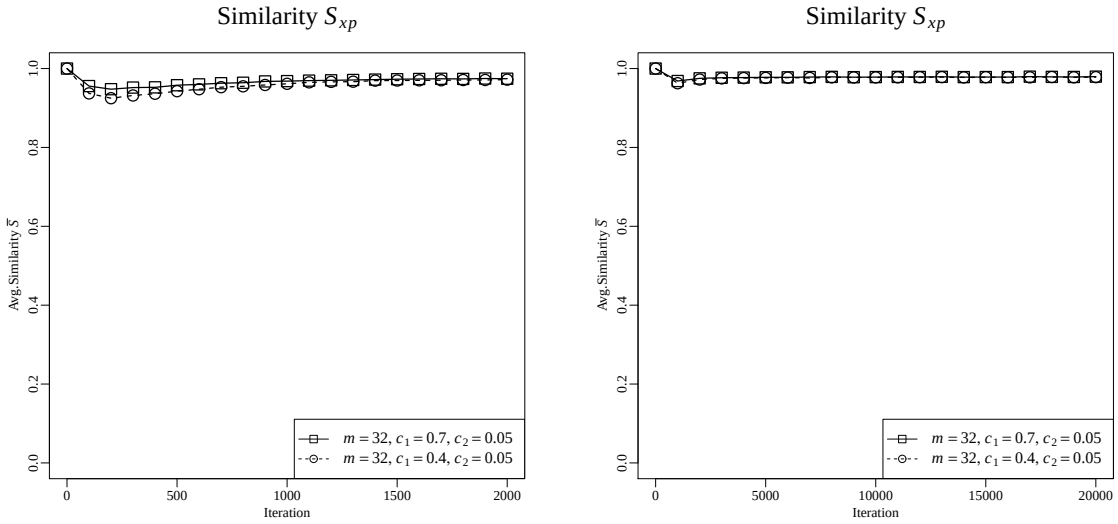


図 4.21　kroA100 に対し IPSO を適用したときの粒子と *pbest* の類似度 $S_{\boldsymbol{xp}}$．左図は 2000 ステップまで．右図は 20000 ステップまで．

図は 20000 ステップまでの $\bar{e_s}$ を示している．右図の結果が，(0,7,0.05) は 17000 ステップ以降，(0.4,0.05) は 9500 ステップ以降表示されていないが，これはこのステップ以降で厳密解が得られているために，$\bar{e_s} = 0$ となり，対数軸表示できなくなっているためである．図 4.19 から，(0.4,0.05) の方がより高速に良い解を発見していることがわかる．図 4.20 と図 4.16 は，$D_{\boldsymbol{p}}$ と $S_{\boldsymbol{xp}}$ の推移を示している．$D_{\boldsymbol{x}}$，$S_{\boldsymbol{xg}}$，$S_{\boldsymbol{pg}}$ は $D_{\boldsymbol{p}}$ と傾向が同じであったため省略する．図 4.20 から，$D_{\boldsymbol{p}}$ は (0.7,0.05) より (0.4,0.05) のほうがわずかに大きいことがわかる．また，図 4.21 から，$S_{\boldsymbol{xp}}$ は (0.7,0.05) より (0.4,0.05) のほうがわずかに小さいとわかる．つまり，(0.4,0.05) の方が特徴量の収束性が強く，その結果，解の収束も早い．しかし，(0.7,0.05) より収束までの時間が短いとしても，kroA100 に対しては厳密解を求めることができるだけの性能があったということである．また，この結果から，kroA100 より難しい問題を解く上では，(0.4,0.05) では解の誤差が小さくなりきる前に解が収束してしまう可能性があることが示された．そのような問題では，よりゆっくりと解が収束する，(0.7,0.05) のような c_1 が大きな値を設定すべきである．

本実験のまとめと考察

本実験の結果を簡単にまとめた上で考察を行う．本実験から次の点が明らかになった．

- 終了ステップ数によって最適なパラメータが変化．
  - IPSO を十分に長い時間適用する場合，$0.4 \leq c_1 \leq 0.9$，$c_2 = 0.05$ がよい．
  - 制限時間が短い場合，$c_1 = 0.05$，$c_2 = 0.7$ がよい．
- c_1 や c_2 の値を変更すると，特徴量が 1 へ収束する速さが変わる．
  - $D_{\boldsymbol{x}}$，$D_{\boldsymbol{p}}$，$S_{\boldsymbol{xg}}$，$S_{\boldsymbol{pg}}$：(0.05, 0.7)，(0.7,0.7)，(0.05,0.05)，(0.7,0.05) の順番に短いステップで収束する．
  - $S_{\boldsymbol{xp}}$：(0.05, 0.7)，(0.7,0.7)，(0.7,0.05)，(0.05,0.05) の順番に短いステップで収束する．

まず，(c_1, c_2) の違いによって得られる解の平均誤差に差が生じる理由であるが，(c_1, c_2) が変わると特徴量の収束速度が変化するからである．予備実験でも示したが，IPSO の特徴量が 1 になった場合，すべての粒子は現在の *gbest* の周辺を制限された 3-opt によって探索する．この探索範囲の制限は $D_{\boldsymbol{p}}$ や $S_{\boldsymbol{pg}}$ が 1 に近づくほどに厳しくなる．(c_1, c_2) によって，このように探索範囲の制限が変化するために，最終的に得られる平均誤差も変わる．c_1 が大きく，c_2 が小さい場合，特徴量の上昇が緩やかであるため，解空間の広い範囲を探索し，誤差の小さい解を得ることができる．c_1 が小さく，c_2 が大きい場合，特徴

量の上昇が急激であり，*gbest* の中心を集中的に探索することで，短い時間である程度良い解を発見できる．

次に，それぞれのパラメータにより，どうしてこのような特徴量の推移になったか考察する．まず，$S_{\boldsymbol{xg}}$，$S_{\boldsymbol{xp}}$ に関しては基本的に理由はとても単純である．$S_{\boldsymbol{xg}}$ は c_2 の値が大きいほど，$S_{\boldsymbol{xp}}$ は c_1 の値が大きいほど，短いステップで 1 に収束する．c_1 と c_2 の両方共に大きな値である (0.7,0.7) の場合は，(0.05,0.7) に近い振る舞いをしている．この理由は，部分経路挿入法のアルゴリズムにある．パラメータ (0.7,0.7) の場合，*pbest* から最大 $[0.7n]$ の長さの部分経路 $\boldsymbol{p}'$ と *gbest* から最大 $[0.7n]$ の長さの部分経路 $\boldsymbol{g}'$ とが選択されるが，このとき，$\boldsymbol{p}'$ と $\boldsymbol{g}'$ に共通する都市が含まれている場合，$\boldsymbol{g}'$ を優先し，$\boldsymbol{p}'$ から該当都市を削除する．最終的に挿入される *pbest* の部分経路 $\boldsymbol{p}''$ は $\boldsymbol{p}'$ より短くなる．つまり，c_2 が十分に大きい場合，c_1 の値の大小の影響力は小さくなる．次に，$S_{\boldsymbol{pg}}$ について考える．$S_{\boldsymbol{pg}}$ は c_2 の値が大きいほど短いステップで 1 に収束する．これは，粒子の解 $\boldsymbol{x}$ に挿入する *gbest* の部分経路が長い場合，それによって更新される *pbest* も *gbest* の部分経路が多く残るからと考えられる．最期に $D_{\boldsymbol{x}}$ と $D_{\boldsymbol{p}}$ について考察する．これらの特徴量も c_2 が大きいときに短いステップで 1 に収束するので，*gbest* の影響が強いと考えられる．しかし，c_2 が十分に小さい場合には，これらの特徴量は 1 にならないことも考えられる．実際に，今回の実験でも (0.7,0.05) は多くの特徴量が約 0.9 で収束している．これは，例えば，与えられた TSP の中に，コストが同じぐらい小さい局所最適解が離れた位置に複数あり，粒子群が分かれてその周辺を探索している場合，*gbest* が小さいことにより，*gbest* の周辺までジャンプできないため，特徴量が 1 にならずに膠着状態になると考えられる．

ここで，(0.05,0.05) に関して考える．(0.05,0.05) は，部分経路挿入による *pbest* や *gbest* への遷移が小さいため，特徴量が上昇しにくいように思える．しかし，実験結果から，$S_{\boldsymbol{xp}}$ においては最も小さいが，それ以外においては (0.7,0.05) のほうが特徴量の値が小さい．これは，*pbest* と *gbest* の引力が同程度なため，より良い巡回路である *gbest* の部分経路が *pbest* にも蓄積されたのではないかと考えられる．一方，(0.7,0.05) は *pbest* の重みが大きいため，*pbest* 周辺に解を留めることができたため，$D_{\boldsymbol{p}}$ を低く保ち，結果的に有望そうな複数の解を探索できたと考えられる．

また，5 種類の特徴量 $D_{\boldsymbol{x}}$，$D_{\boldsymbol{p}}$，$S_{\boldsymbol{xp}}$，$S_{\boldsymbol{xg}}$，$S_{\boldsymbol{pg}}$ のうち，$D_{\boldsymbol{x}}$，$D_{\boldsymbol{p}}$，$S_{\boldsymbol{xg}}$，$S_{\boldsymbol{pg}}$ については傾向がほぼ同じであると示された．その理由は上記の考察で示したとおり，各粒子の解への *gbest* の伝搬によるところが大きいと考えられる．さらに，$S_{\boldsymbol{xp}}$ に関しても，c_1 の大きさに素直に比例していることも示された．以上より，今後は $D_{\boldsymbol{p}}$ の値のみを使って収束性に関して議論をすすめる．$D_{\boldsymbol{p}}$ を使用する理由は，他の特徴量に比べて $D_{\boldsymbol{p}}$ は収束が

表 4.3　パラメータ (c_1, c_2) が探索に及ぼす影響の調査で用いるパラメータ

粒子数 m		$\{16, 32, 64, 512\}$
pbest の重み c_1		0.7
gbest の重み c_2		0.05
使用する TSP		kroA100
パラメータセットあたりの試行回数		100
終了条件	ステップ数	$\{2000, 40000\}$
	時間 (秒)	$\{2, 20\}$

遅いことである．収束の遅さだけなら $D_{\boldsymbol{p}}$ より $D_{\boldsymbol{x}}$ が遅いが，$D_{\boldsymbol{p}}$ はそれ以降のステップの探索範囲にも影響しており，$D_{\boldsymbol{x}}$ より得られる情報が多いため，$D_{\boldsymbol{p}}$ を使用する．

4.7.6 節の結論

本実験では，c_1 と c_2 の値によって IPSO の探索性能がどのように変化するか調査した．実験結果から，与える制限時間によって最適な (c_1, c_2) が異なることがわかった．具体的には，c_1 が大きく，c_2 が小さい場合に，十分時間をかけると最も誤差の小さい解を発見でき，逆に，c_1 が小さく，c_2 が大きい場合に，短時間である程度良い解を見つけることができる．ただし，$c_1 = 0$ や $c_2 = 0$ として，*pbest* や *gbest* を用いない場合は得られる解は悪くなる．また，c_1 と c_2 の値を変えたときに IPSO がどのような探索をしているかが，$D_{\boldsymbol{p}}$ の推移から明らかになった．c_1 が大きく，c_2 が小さい場合には，各粒子は解に *pbest* を強く残そうとするため，$D_{\boldsymbol{p}}$ の上昇が緩やかになった．つまり，各粒子は比較的に距離を保ちながら探索し，その結果，様々な有望そうな解の周囲を探索している．一方，c_1 が小さく，c_2 が大きい場合には，各粒子は解に *gbest* を積極的に取り込むため，$D_{\boldsymbol{p}}$ の上昇が急激になった．言い換えると，各粒子は *gbest* の周囲を集中的に探索する．

4.7.7　パラメータ m が探索に及ぼす影響の調査

次に，m が探索に及ぼす影響について調査する．実験設定を表 4.3 に示す．(c_1, c_2) は 4.7.6 節の実験で十分に時間をかけたときに最も解の誤差が小さくなった (0.7,0.05) を使用する．IPSO の終了条件はステップと時間についてそれぞれ行う．ステップでの議論は IPSO の基本的な性能を調査する上で欠かせない．しかし，IPSO をシングルスレッドで

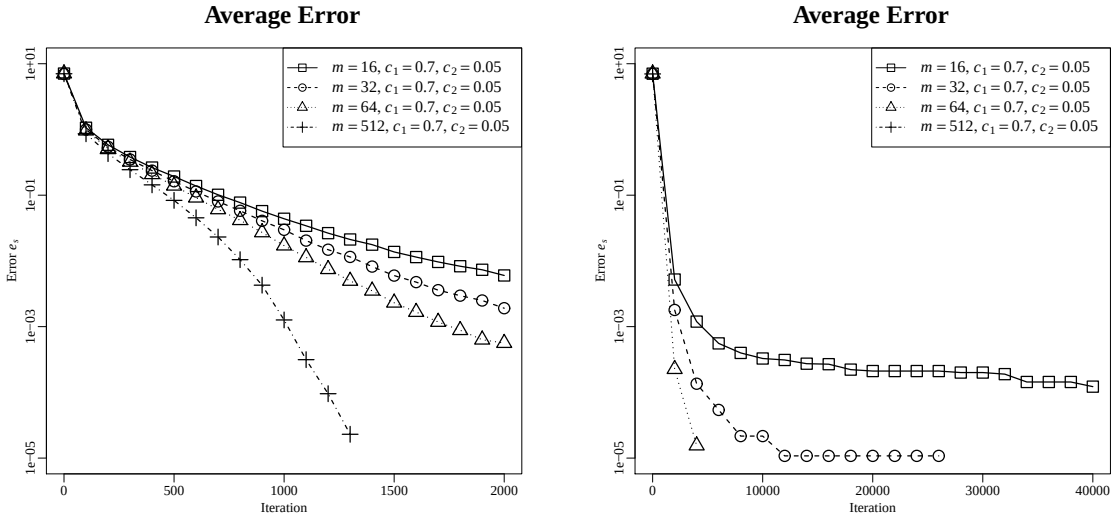


図 4.22　kroA100 に対し IPSO を適用し得られた解の平均誤差 $\bar{e_{\boldsymbol{x}}}$．左図は 2000 ステップまで．右図は 40000 ステップまで．

実行する場合，粒子数が多いほど 1 ステップあたりにかかる計算時間が多くなる．このため，終了条件に処理時間を設定した IPSO において比較することも必要である．終了ステップは，粒子数を変化させると収束までに必要なステップ数が変化する可能性があるので，長い方を 40000 ステップとする．その他の設定については 4.7.6 節と同様である．

終了条件をステップ数とした実験結果と考察

終了条件をステップ数にした実験結果を図 4.22，4.23 に示す．図 4.22 は平均誤差 $\bar{e_{\boldsymbol{x}}}$ の推移を示している．$m = 512$ の 1400 ステップ以降と $m = 64$ の 6000 ステップ以降，$m = 32$ の 28000 ステップ以降は最適解が得られていることを表している．この結果は，粒子は多ければ多いほど，同じステップあたりで得られる解が良くなることを示している．図 4.23 は *pbest* の密度 $D_{\boldsymbol{p}}$ と *pbest* と *gbest* の類似度 $S_{\boldsymbol{pg}}$ を示している．この結果から，(c_1, c_2) を固定した場合，ステップあたりの $D_{\boldsymbol{p}}$ と $S_{\boldsymbol{pg}}$ はほぼ変化しないことがわかる．粒子数が大きくなっても $D_{\boldsymbol{p}}$ や $S_{\boldsymbol{pg}}$ に変化がないことから，次の 2 つのことが導かれる．1 つ目は，kroA100 という問題は 512 の粒子を散りばめても余りある解空間であるということである．仮に，解空間がもっと狭いか粒子がとても多い場合，$D_{\boldsymbol{p}}$ は上昇してしまうからである．2 つ目は，粒子が多いほど探索する領域が広いことである．これは

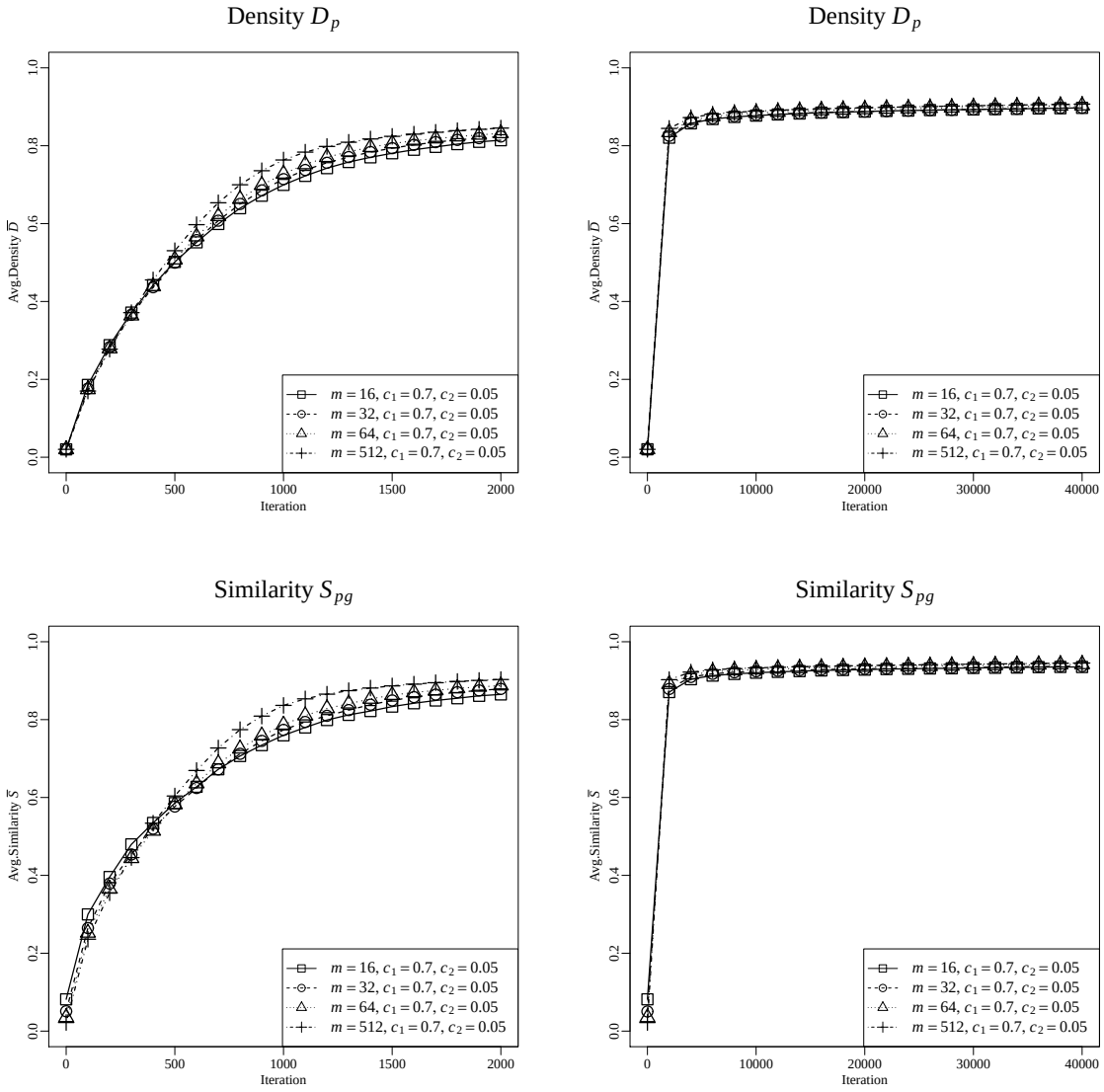


図 4.23　kroA100 に対し IPSO を適用したときの *pbest* の密度 $D_{\boldsymbol{p}}$（上図）と *pbest* と *gbest* の類似度 $S_{\boldsymbol{pg}}$（下図）．それぞれ左図は 2000 ステップまで．右図は 40000 ステップまで．

探索範囲を球に例えるとわかりやすい．図 4.24 に例を示す．各色の点はそれぞれの粒子の *pbest* を表しており，点の周りの球はその粒子の探索範囲を示している．左図は 3 つの粒子，右図は 4 つの粒子が存在しているが，各 *pbest* の類似度はすべて同一であり，左右の $D_{\boldsymbol{p}}$ は同じ値になる．このように，$D_{\boldsymbol{p}}$ が同じ IPSO においては，粒子が多いほど広い

図 4.24　粒子数が異なるが *pbest* の密度 $D_{\boldsymbol{p}}$ が変わらない例.

範囲を探索している．また，左右の図の × は *gbest* の位置を示しており，すべての *pbest* から等距離にある．つまり，$D_{\boldsymbol{p}}$ や $S_{\boldsymbol{pg}}$ に変化がないということは，粒子を増やしたとき，*gbest* を中心に，探索範囲が被らないように粒子が配置されていることを示している．粒子数を多くしたときに少ないステップでよい解が見つかった理由は，このように，1 ステップあたりに探索する範囲が広いからと考えられる．

終了条件を実時間とした実験結果と考察

終了条件を実時間にした実験結果を図 4.25，4.26 に示す．図 4.25 は平均誤差 $\bar{e_{\boldsymbol{x}}}$ の推移を示している．$m = 512$ の 8 秒以降と $m = 64$ の 10 秒以降は最適解が得られていることを表している．この結果から，制限時間の 20 秒経過時では粒子数が大きいほど良い解が得られているが，制限時間によって有効な粒子数が異なることがわかる．特に，粒子数が小さいほど探索初期に得られる解の誤差が小さいことが示されている．

図 4.25 は平均誤差 $\bar{e_{\boldsymbol{x}}}$ の推移を，図 4.26 は *pbest* の密度 $D_{\boldsymbol{p}}$ と *pbest* と *gbest* の類似度 $S_{\boldsymbol{pg}}$ を示している．図 4.25 については，$m = 512$ の 8 秒以降と $m = 64$ の 10 秒以降は最適解が得られているためプロットがない．これらの図の結果から，制限時間の 20 秒経過時では粒子数が大きいほど良い解が得られているが，制限時間によって有効な粒子数が異なることがわかる．特に，粒子数が小さいほど探索初期に得られる解の誤差が小さいことが示されている．また，$D_{\boldsymbol{p}}$ や $S_{\boldsymbol{pg}}$ も収束が早い．これは，終了条件をステップにした結果と合わせて考えると，粒子数 m が小さい場合はすべての粒子を 1 ステップ処理する時間が短いので，結果として多くのステップを処理できる m を小さく設定したときに収束も早くなると考えられる．

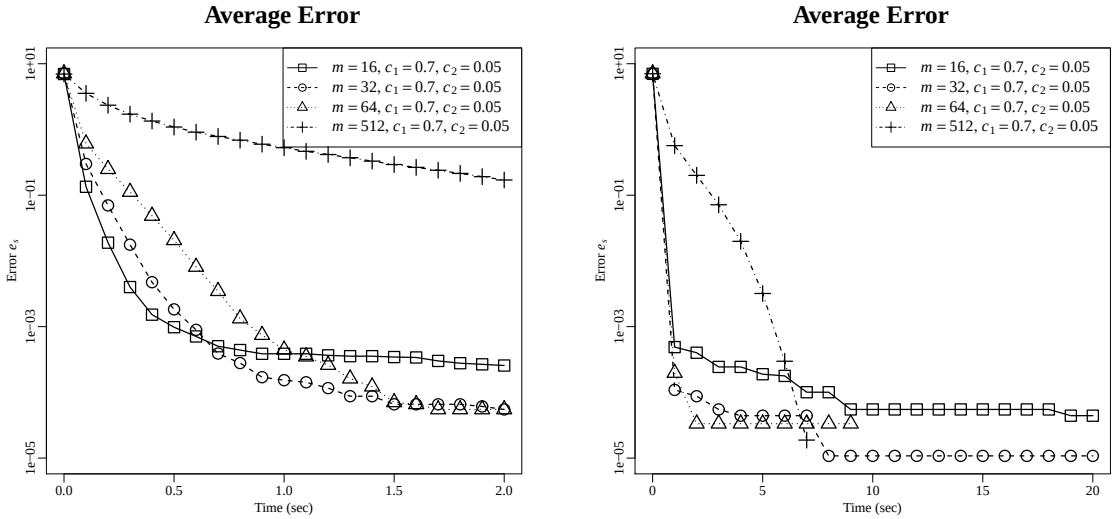


図 4.25　kroA100 に対し IPSO を適用し得られた解の平均誤差 $\bar{e_{\boldsymbol{x}}}$．左図は制限時間が 2 秒．右図は 20 秒．

4.7.7 節の結論

本実験では，(c_1, c_2) を固定した場合，m の値によって IPSO の探索性能がどのように変化するか調査した．実験結果から，粒子数 m を大きくすると，*gbest* の周囲をより広い範囲で探索することがわかり，十分に時間を書ける場合は m を大きな値に設定した方がより誤差 $e_{\boldsymbol{x}}$ の小さい解を見つけられることが示された．しかし，設定する制限時間によって最適な m が異なることがわかった．m が増加すると $e_{\boldsymbol{x}}$ の収束が遅くなるため，制限時間が短い場合は m を小さく設定すべきであることが示された．

4.7.8　4.7 節の結論

本節では，IPSO のパラメータである，粒子数 m，*pbest* の重み c_1，*gbest* の重み c_2 を変化させることで，探索にどのような違いが生じるか調査した．その結果，(c_1, c_2) と m について以下のことが判明した．

- 設定する探索時間によって最適な (c_1, c_2) は異なる．短時間である程度良い解を求めたい場合は c_1 を小さく，c_2 を大きく設定するとよい．逆に，時間がかかっても

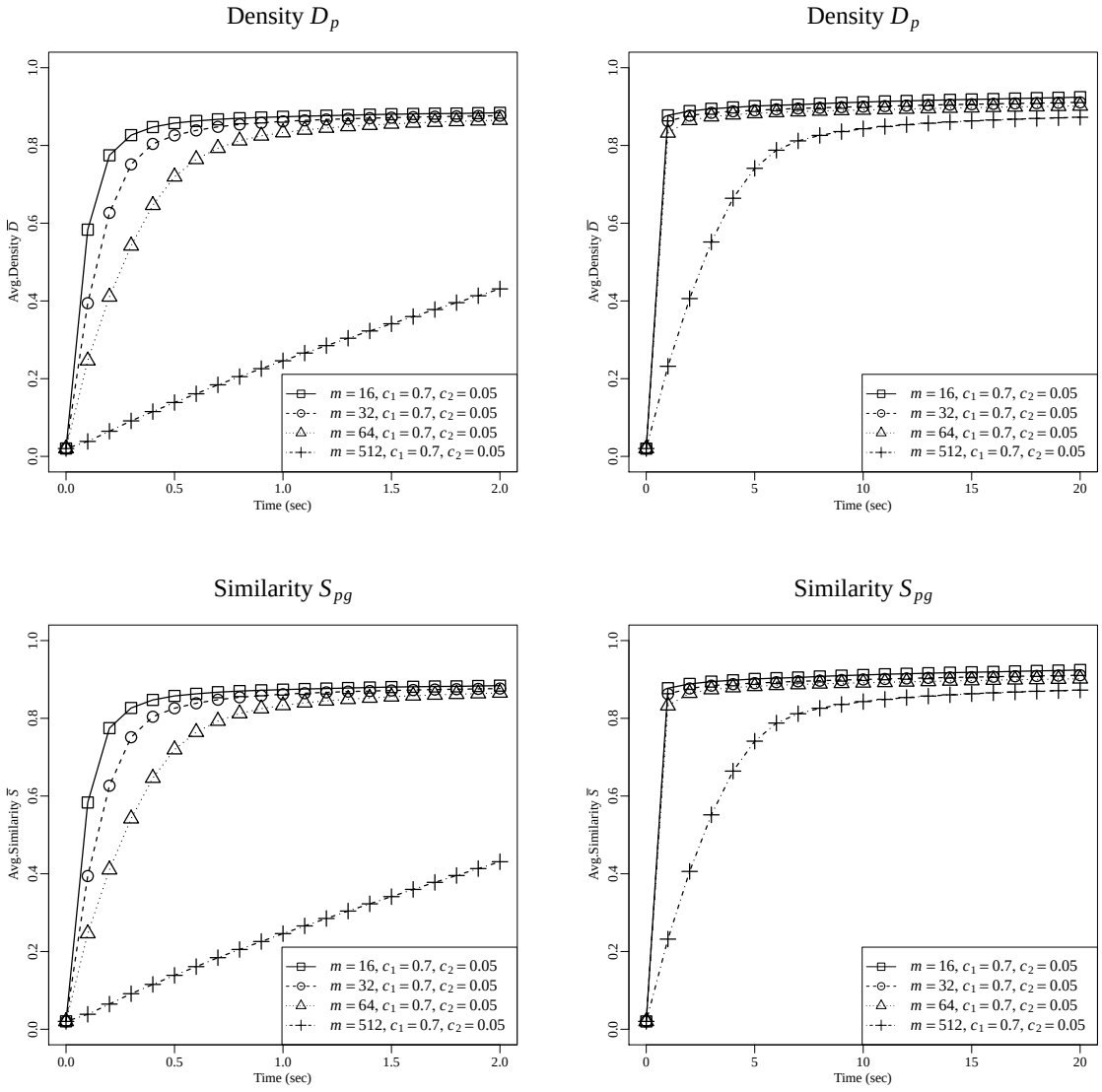


図 4.26　kroA100 に対し IPSO を適用したときの *pbest* の密度 $D_{\boldsymbol{p}}$（上図）と *pbest* と *gbest* の類似度 $S_{\boldsymbol{pg}}$（下図）．それぞれ左図は制限時間が 2 秒．右図は 20 秒．

誤差を最小化したい場合は c_1 を大きく，c_2 を小さく設定すればよい．

- m も設定する探索時間によって最適な値が異なる．m が小さいほど初期の解の収束がよく，m が大きいほど最終的に得られる解の誤差が小さい．

(c_1, c_2) は部分経路挿入に用いる *pbest* と *gbest* の部分経路の長さに関するパラメータで，粒子の解は c_1 が大きいほど *pbest* に，c_2 が大きいほど *gbest* に近づく．*gbest* はすべての粒子が共有する解であるため，c_2 が大きい場合，粒子はどうしても *gbest* 周辺に集中してしまう．このために，上記のような特性が現れると考えられ，数値計算実験の結果もこれを裏付けるものであった．

m を大きく設定した場合に最終的により小さい誤差の解が見つかる理由は，粒子が追加されたことが原因で生じる探索範囲の重複が小さいために，粒子を多くすればするほど IPSO はより広い解空間を探索するからである．一方，m を大きくした場合に探索初期に解があまり良くならない原因は，粒子数を増やしたことにより，単位時間あたりに計算できるステップ数が減ったことが理由である．

4.8　数値計算実験：代表的な従来手法との比較実験

提案手法 IPSO の有効性を示すために，いくつかの既存手法との性能比較を行う．本実験では，焼きなまし法（SA）と遺伝的アルゴリズム（GA）を比較手法として用いる．それぞれの方法で解の生成方法や評価方法がかなり異なるため，解の評価回数などで比較することは適切ではないと考え，実行時間に基づいた比較を行うこととする．ただし，それぞれの方法でデータ構造や実装上の工夫をすることでなるべく実行時間がかからないよう工夫し，各パラメータも十分にチューニングすることとした．実験では，数種類のベンチマーク問題に対して制限時間（秒）内で得られた解の誤差を比較する．

4.8.1　実験設定

本実験で用いる SA と GA の詳細設定と，その他の実験設定について説明する．

SA

初期解 $\boldsymbol{s}_0$ はランダムな実行可能解とする．近傍解は現在の解 $\boldsymbol{s}$ からランダムに 2 辺を選択してそれらを繋ぎ変えた巡回路とする．近傍解 $\boldsymbol{s}'$ が改悪解であった場合の受理率 p_w は 4.11 式により決定する．

$$p_w = \exp\left(-\frac{C(\boldsymbol{s}') - C(\boldsymbol{s})}{T}\right) \tag{4.11}$$

ここで，C はコスト関数，T は温度パラメータとする．T は探索が進むにつれて減少させる．ステップ t における温度は 4.12 式に従う．

$$T(t) = T_0 \alpha^{\left[\frac{t}{R}\right]} \tag{4.12}$$

T_0 は初期温度，α は減少率，R は各温度での反復回数，$[a]$ は実数 a の整数部を返す記号である．パラメータ T_0，α，R は問題ごとに予備実験を行い決定する．SA のパラメータ調整については，初期温度については探索開始時の受理率が 0.5 から 0.8 程度になる値に，減少率は 0.9 から 0.9999 などの 1 に非常に近い値に設定するのが一般的である [13]．初期温度，減少率についてはこれらの範囲内で調整し，各温度での反復回数 R は制限時間を考慮しつつ決定する．

GA

m 個の個体に対し，選択，交叉，突然変異を終了条件が満たされるまで繰り返し適用する．選択にはトーナメント選択を使用する．トーナメント選択は，個体群中の 2 個体の総経路長を比較し，優秀な個体のみを次世代に残す手法である．選択後の個体数を m にするために，各個体は 2 度ずつ自身以外の別々の個体と比較される．交叉には提案方法で使用した挿入法を使用する．個体 A と個体 B が交叉のペアとして選ばれた場合，個体 A に個体 B の部分経路を，個体 B に個体 A の部分経路を挿入し，次世代の個体 A’，B’ とする．部分経路長 l_p は $2 \leq l_p \leq n$ の範囲でその都度ランダムに決定する．一世代ごとにランダムに選択された m_c 個，$m_c/2$ 組の個体に交叉を適用する．突然変異は，選択された個体の解 $\boldsymbol{s} = \{v_1, v_2, \cdots, v_n\}$ からランダムに都市をいくつか選択し，それを別の都市間に挿入する操作とする．移動する都市数は 1 つか 2 つとし，その都度ランダムに決定する．一世代ごとにランダムに選択された m_m 個の個体に適用する．これ以外に，エリート保存を適用する．エリート保存とは，個体群内で総経路長が小さいいくつかの個体を次世代に残す戦略である．本実験に用いる GA では，個体群内で総経路長が最も小さい個体を交叉および突然変異の対象から除外する．また，トーナメント選択を採用しているため，この個体は次世代にも確実に残される．パラメータ m，m_c，m_m は問題ごとに予備実験を行い決定する．

その他の実験設定

その他の実験設定を表 4.4 に示す．比較には, 上記に示す都市数が 51 から 1002 までの 10 個のベンチマーク問題を使用する．また，IPSO，SA，GA は与えるパラメータによっ

表 4.4　IPSO と従来手法の比較実験の実験条件

使用する TSP	eil51，kroA100，kroA200，tsp225，pr299，pr439，rat575，d657，u724，pr1002
各手法の制限時間	60 秒
試行回数	100
実験機（CPU）	Intel Xeon E5520
OS	Fedora 20
実装言語	Scala 2.10.4，Java 1.7.0

て探索性能が大きく変わるため，実装した IPSO，SA，GA を用いて事前実験を行い，各ベンチマーク問題に対して制限時間 60 秒で得られる解の誤差が最小になるパラメータセットを求め，それを用いる．各 TSP に対してそれぞれのパラメータセットごとに得られた誤差の最小値，平均値，最大値を比較する．

4.8.2　性能比較実験結果

本実験で得られた結果を表 3 に示す．誤差 $e_{\boldsymbol{x}}$ の最小値，平均値，最大値はそれぞれ百分率で小数点第 3 位まで表示する．$e_{\boldsymbol{x}}$ の標本標準偏差は指数表記で有効桁数 3 桁で示す．表 4.5 より，使用したすべてのベンチマーク問題において，最小値，平均値，最大値のいずれも IPSO が最も小さいことがわかる．IPSO と SA について，それぞれのベンチマーク問題に対して 2 群の平均値の差の検定を Welch の t 検定により行った結果，すべての問題で 1% 水準で有意差が認められた．また，IPSO と GA についても同様に検定を行った結果，すべての問題で 1% 水準で有意差が認められた．これらのことから，すべての手法に同一の制限時間を設定する場合，IPSO は SA や GA より誤差 $e_{\boldsymbol{s}}$ が小さい解を得られる可能性が高いといえる．

表 4.5　性能比較実験結果

TSP	IPSO				SA				GA			
	Error (%)			S.D.	Error (%)			S.D.	Error (%)			S.D.
	Min	Average	Max		Min	Average	Max		Min	Average	Max	
eil51	0.000	0.000	0.000	0.00E+00	0.000	0.134	0.469	1.30E-03	0.000	1.687	3.756	8.33E-03
kroA100	0.000	0.000	0.000	0.00E+00	0.000	0.527	1.917	4.11E-03	0.000	1.817	6.390	1.50E-02
kroA200	0.000	0.254	0.739	1.83E-03	0.143	1.647	4.450	7.66E-03	0.678	3.176	6.453	1.26E-02
tsp225	0.409	1.290	2.298	4.05E-03	0.715	2.911	4.750	8.02E-03	1.175	3.717	6.793	1.07E-02
pr299	0.145	0.766	1.801	3.62E-03	0.488	2.206	7.557	1.01E-02	0.726	3.683	7.454	1.33E-02
pr439	0.471	2.450	5.369	1.07E-02	1.243	4.506	8.317	1.51E-02	1.523	5.042	8.575	1.43E-02
rat575	3.617	4.850	6.644	6.32E-03	5.832	7.724	10.320	7.89E-03	4.370	6.252	8.460	7.04E-03
d657	2.605	4.290	6.504	8.06E-03	5.457	7.882	10.744	1.02E-02	4.807	7.190	10.145	1.05E-02
u724	3.276	4.564	6.531	6.55E-03	6.998	9.514	15.614	1.06E-02	7.559	10.251	12.971	9.26E-03
pr1002	3.413	5.234	7.036	7.50E-03	10.254	12.870	16.705	1.33E-02	11.305	29.299	34.692	3.75E-02

表 4.6　性能比較実験で使用した IPSO のパラメータ (m, c_1, c_2)

TSP	m	c_1	c_2	TSP	m	c_1	c_2
eil51	128	0.7	0.05	pr439	8	0.7	0.05
kroA100	64	0.7	0.05	rat575	16	0.05	0.9
kroA200	64	0.7	0.05	d657	4	0.05	0.9
tsp225	32	0.7	0.05	u724	4	0.05	0.95
pr299	32	0.7	0.05	pr1002	4	0.05	0.95

4.8.3　実験考察

本実験では各ベンチマーク問題ごとに IPSO のパラメータ調整を行った．4.7 節の知見から，(c_1, c_2) を $\{(0.7, 0.05), (0.4, 0.05), (0.05, 0.9), (0.05, 0.95)\}$ の 4 組の中から，m を $\{4, 8, 16, 32, 64, 128\}$ の 6 つの中から選択した．これらの組み合わせにより，探索初期の収束性を強めたり，逆に，解精度重視にしたりできる．予備実験として，問題ごとにこれらのパラメータの組を数個選択し，本実験と同様の制限時間で探索を行い，最も誤差 $e_{\boldsymbol{s}}$ が小さくなるパラメータ値を採用した．表 4.6 に問題ごとの最適パラメータを示す．問題ごとに最適なパラメータは異なり，その傾向は 4.7 において示したものに一致した．eil51 から pr439 までは，$(c_1, c_2) = (0.7, 0.05)$ で，m が問題規模が大きくなるにつれ小さくなっている．この範囲では，粒子は *pbest* の密度を低く保つパラメータが選択されており，収束性は粒子数 m で調整されている．rat575 では，$(m, c_1, c_2) = (16, 0.05, 0.9)$ となり，以降は (c_1, c_2) が粒子の *gbest* への収束を強めるパラメータが最適となっている．d657 以上の問題では $m = 4$ となり，パラメータ調整に使用した範囲の最小の粒子数となった．u724 と pr1002 では，*gbest* への収束性をさらに高めるために $c_2 = 0.95$ が最適となった．

以上の結果から，制限時間を設定する場合の IPSO のパラメータ調整について次の指針が導ける．まずはじめに，c_1 に大きな値，c_2 に小さな値を設定し，粒子数 m をできるだけ大きな値を設定する．これは，粒子の探索範囲を出来るだけ大きくし，最適解を探索する戦略である．次に，このパラメータで解の収束曲線を記録し，制限時間内に収束していなければ，m を減少させる．この調整により IPSO の探索範囲は狭まるが，同じ探索時間で粒子の更新をよりたくさん行える．m を小さくしても収束しないようであれば，次は c_1 に小さな値，c_2 に大きな値を設定し，m を少し大きくする．(c_1, c_2) の変更により，

粒子は *gbest* への収束を強め，*gbest* の周囲を集中的に探索し，解の収束が大きく早まる．それは，m を少し大きくしても余裕があることが多い．これらの変更により，局所最適解に陥るリスクはあるが，短時間である程度の精度の解を発見できる．それでも解が収束しない場合は，m を出来る限り小さくする．

4.9　結論

本章では，既存のメタヒューリスティクス，特に遺伝的アルゴリズムより高速に良い解を求めることができる，TSP 向けのヒューリスティクスの開発を目的とし，実数値最適化手法の一種である PSO が高速に高精度な解を発見であることに着目して，PSO の探索の戦略をベースとした TSP ソルバである IPSO を構築した．PSO で組合せ最適化問題である TSP を直接扱うことは難しいため，IPSO では，解表現を巡回路とし，粒子の解の更新を *pbest* や *gbest* の部分経路を自身に取り込む部分経路挿入法を用いて行うように変更した．

次に，IPSO のパラメータによる探索性能の変化について調査するために数値計算実験を行った．その結果，IPSO の最適なパラメータは問題の規模や制限時間によって変わることが示された．IPSO には，粒子数 m，*pbest* の重み c_1，*gbest* の重み c_2 の 3 つのパラメータがあるが，(c_1, c_2) に関しては，c_1 を小さく，c_2 を大きく設定すると短時間である程度良い解が得られ，逆に，c_1 を大きく，c_2 を小さく設定すると，収束までの時間は長くなるがより小さい誤差の解が得られることがわかった．(c_1, c_2) は部分経路挿入に用いる *pbest* と *gbest* の部分経路の長さに関するパラメータで，粒子の解は c_1 が大きいほど *pbest* に，c_2 が大きいほど *gbest* に近づく．よって，c_2 の比率が大きいほど，探索は *gbest* の周囲を集中的に探索するため，局所最適解に陥るリスクはあるものの，短時間で良い解が見つかる．m に関しては，m が小さいほど初期の解の収束がよく，m が大きいほど最終的に得られる解の誤差が小さいことが示された．m を大きくすればするほど，解空間のより広い範囲を探索するため精度の良い解を発見しやすいが，単位時間あたりに処理できる粒子数が減少するために，探索初期の解の精度はあまり良くない．

次に，IPSO の有効性を示すために，SA と GA との性能比較を行った．問題規模の違う 10 種類の TSP に対して，同一の制限時間を設定した IPSO，SA，GA を適用し，得られる解の誤差を比較した結果，すべての TSP に対して IPSO が最も優れていることが示された．制限時間を固定したことにより，規模の大きい問題ほど解くことが難しくなっているが，IPSO はそのような TSP に対してもパラメータをチューニングすることである

程度精度の良い解を発見できていた．また，この実験から，IPSO のパラメータチューニングに関する指針が示された．まずはじめに，c_1 に大きな値，c_2 に小さな値，粒子数 m に大きな値を設定する．これは十分時間がある場合に誤差を最小化できるパラメータセットである．もし，このパラメータセットで制限時間内に解が収束しない場合は，m を減らす．m を減らしても収束しない場合には c_1 に小さく，c_2 を大きくする．このようにパラメータを調整すると，その制限時間で得られる解の精度を最もよくできる．

第 5 章

粒子間通信を制限した挿入操作 PSO 戦略

4 章では，IPSO を提案し，数値計算実験から高速に高精度な解を発見可能であることを示した．IPSO の短いステップで良い解を発見できる特性は，*gbest*，つまり，群全体で発見した解の中で最もコストが小さい解をすべての粒子が共有することにより実現されている．しかし，それが理由で，*gbest* の重み c_2 を小さく設定しても，各粒子の解 $\boldsymbol{x}_i$ や *pbest* $\boldsymbol{p}_i$ は短いステップで *gbest* $\boldsymbol{g}$ に収束してしまい，局所最適解に陥ることもある．本章では，より局所最適解に陥りにくくすることを目的として，IPSO をより長いステップにわたり *pbest* のバリエーションを保つために粒子間通信を制限した IPSO を提案し，数値計算実験からその有効性に関して調査する．

5.1 IPSO が局所最適解に陥る要因とその解決策

4 章において，IPSO の解の収束と *pbest* の密度 $D_{\boldsymbol{p}}$ の関係を調査し，$D_{\boldsymbol{p}} = 1$ であるか，または $D_{\boldsymbol{p}}$ が比較的大きな値に収束している場合，解もそれ以上更新されないことが示された．また，このとき，*pbest* と *gbest* の類似度 $S_{\boldsymbol{pg}}$ も $S_{\boldsymbol{pg}} \geq D_{\boldsymbol{p}}$ であり，すべての *pbest* は *gbest* の周囲に存在していることが明らかになった．上記の結果から，*gbest* をすべての粒子で共有していることが *pbest* のバリエーションを減らしており，そのために解の収束が早いと考えられる．そこで，*gbest* を廃止し，粒子群に通信ネットワークを適用してリンクで接続された粒子間でのみ情報のやり取りが行えるように変更することで，*pbest* が急激に収束することを防ぐ．

5.2　適用する粒子間ネットワーク構造

5.2.1　様々なネットワークとその平均距離

ネットワークは古くから研究されており，最も単純で基本的な理論はグラフ理論としてまとめられている．また，その古典的なグラフ構造としては，完全グラフや n 次元格子，サイクル，木などが有名である．IPSO は *gbest* をすべての粒子で共有しているが，これは，すべての粒子が相互に接続されており，各々の粒子が群全体を走査して最良な解を *gbest* としていると解釈できる．つまり，IPSO は粒子間ネットワークとして完全グラフを持っているといえる．

近年では，複雑ネットワークと呼ばれる分野に発展している．複雑ネットワークでは，グラフ理論の数学的な部分を基本としつつ，現実世界のネットワークを対象としている．例えば，人間関係ネットワークやインターネット，食物網，神経系と脳，電力網，航空網などの交通系のネットワークなどがある [36]．複雑ネットワークのネットワークモデルとしては，ランダムグラフ [37] や Watts-Strogatz (WS) モデル [38]，Barabási-Albert（BA）[39] モデルが有名である．

ネットワーク上の情報伝搬の速さについて考えるとき，重要な要素のひとつにネットワークの平均距離がある．ネットワークにおける 2 頂点 v_i と v_j の距離 $d(v_i, v_j)$ は，v_i から v_j までに通らなければならない最小の枝数と定義される．通常，ネットワークは無向なので，$d(v_i, v_j) = d(v_j, v_i)$ である．平均距離 L はすべての頂点対の距離の平均であり，次式で定義される．

$$L = \frac{2}{N(N-1)} \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} d(v_i, v_j) \tag{5.1}$$

ここで，N は頂点数である．上記に例として紹介したネットワークモデルの平均距離 L の概算を表 5.1 に示す [36, 37, 40, 41, 42]．ただし，WS モデルに関しては，一般的にスモールワールド性が認められるネットワークにおける L の推定式である．$\langle k \rangle$ は平均次数である．次数とは頂点に接続されている辺の数を表す．p は WS モデル内の枝張替えに関するパラメータである．

表 5.1　各ネットワークモデルの平均距離 L

ネットワークモデル	平均距離 L
完全グラフ	1
n 次元格子	$O(N^{\frac{1}{n}})$
拡張サイクル	$O(N)$
木	$O(\log N)$
ランダムグラフ	$O\left(\frac{\log N}{\log\langle k\rangle}\right)$
Watts-Strogatz モデル	$O\left(\frac{\log(N\langle k\rangle p)}{\langle k\rangle^2 p}\right)$
Barabási-Albert モデル	$O(\log N)$

5.2.2　IPSO に用いるネットワーク

IPSO の粒子間ネットワークに求める条件は，情報伝播ができるだけゆっくりと進むことである．これは，平均距離 L で考えると，できるだけ L が大きい方が適している．5.1 から，これらの代表的なネットワークの中で最も L が大きくなるものは，1 次元格子か拡張サイクルである．このうち，どの頂点も均等である拡張サイクルを使用する．

拡張サイクル

図 5.1 に拡張サイクルと類似のネットワークを示す．図 5.1 の左上図は 1 次元格子である．図にある通り，1 次元格子は頂点が 1 次元上に並んでおり，それぞれの頂点は自身の左右の頂点のみと接続されている．1 次元格子には末端の頂点が 2 つあり，それらはそれぞれ 1 つしか辺を持たない．図 5.1 の左下図はサイクルである．サイクルは 1 次元格子の末端の頂点を接続させたネットワークである．このようなことを周期的境界条件という．周期的境界条件を課すことにより，1 次元格子は末端がなくなり，ネットワークから辺構造の偏りをなくすことができる．図 5.1 の右上図は拡張 1 次元格子である．拡張 1 次元格子は 1 次元格子の各頂点が，隣接頂点だけでなく，d 個隣の頂点まで辺で接続されたネットワークである．拡張 1 次元格子においても末端の頂点が存在し，その頂点はその他の頂点と比べて接続される辺の数が半分になる．図 5.1 の右下図は拡張サイクルである．拡張サイクルは拡張 1 次元格子に周期的境界条件を課したネットワークである．すべての頂点が左右 d 個の頂点と接続されている．

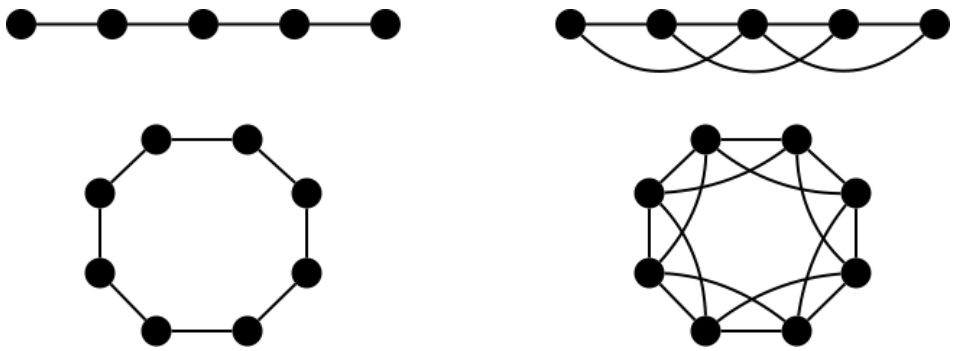


図 5.1　1 次元格子（左上図），サイクル（左下図），拡張 1 次元格子（右上図），拡張サイクル（右下図）．

拡張サイクルの平均距離を感覚的に考える．まず，サイクルの平均距離については簡単に求めることができる．頂点数 n が奇数のサイクルの場合，ある頂点は，距離 1 の頂点を 2 つ，距離 2 の頂点を 2 つ $\cdots$ のように，左右に等距離の頂点を持つ．また，その頂点の左右にはそれぞれ $(n-1)/2$ の頂点があり，最も遠い頂点は $(n-1)/2$ の距離がある．つまり，ある頂点 i から他の頂点の平均距離 $\bar{l_i}$ は

$$\bar{l_i} = \frac{1}{n-1} \sum_{l=1}^{(n-1)/2} 2l = \frac{n+1}{4} \tag{5.2}$$

となる．一方，頂点数 n が偶数の場合，奇数とは異なり，ある頂点は最も遠い頂点が 1 つだけであり，距離は $n/2$ である．よって平均距離 $\bar{l_i}$ は

$$\bar{l_i} = \frac{1}{n-1} \left(\frac{n}{2} + \sum_{l=1}^{n/2-1} 2l \right) = \frac{n^2}{4(n-1)} \tag{5.3}$$

となる．上記はある 1 つの頂点の平均距離であるが，対称性からサイクルの平均距離と一致する．以上から，サイクルの平均距離は約 $n/4$ といえる．拡張サイクルは，d 個先まで接続するため，ある頂点から別の頂点までの距離は，サイクルのときの約 $1/d$ となる．よって，拡張サイクルの平均距離は L は約 $n/4d$ となる．次数 k を用いると，$d = k/2$ から

$$L \approx \frac{n}{2k} \tag{5.4}$$

である．以上より，頂点数 n が次数 k より十分に大きい場合，平均距離 L は頂点数 n に比例することがわかる．

拡張サイクルは local version と呼ばれる粒子間通信を制限した PSO[34] でも用いられているネットワーク構造である．*pbest* の収束を遅らせる目的でネットワーク構造を導入するのであれば，上記の理由から拡張サイクルを適用することが最善と考えられる．

5.3　4 章で提案した IPSO からの変更点

4 章で提案した IPSO からの変更点を以下にまとめる

- **粒子にネットワーク構造を導入**

  粒子群に拡張サイクルによるネットワークを導入する．5.1 の例のように，それぞれの粒子を自分以外の左右 d 個の粒子と接続する．粒子 i と辺で接続された粒子の小群を，粒子 i の近傍と呼び，A_i で表す．

- ***gbest* の廃止，*lbest* の導入**

  まず，粒子全体で共有していた *gbest* を使わないように変更する．その代わりに，各粒子 i は近傍 A_i と自身がこれまでに探索した中で最もコストが小さい解を保持する．この巡回路を *lbest* と呼ぶ．*lbest* はその粒子自身の探索履歴も含むため，近傍粒子が発見した解より自身が発見した解の方が優れているならば，*lbest* は *pbest* に一致する．

5.4　粒子間通信を制限した IPSO のアルゴリズム

粒子間通信を制限した IPSO のアルゴリズムを以下に示す．各粒子 i は巡回路（解）$\boldsymbol{x}_i$ と，粒子 i の *pbest* $\boldsymbol{p}_i$ と，粒子 i の *lbest* $\boldsymbol{l}_i$ をもつ．粒子 i の近傍 A_i は $A_i = \{i-d, \cdots, i-2, i-1, i+1, i+2, \cdots, i+d\}$ とする．d は 1 以上の整数である．なお，$i-d \leq 0$ の場合は $i-d+m$ とし，$i+d > m$ の場合は $i+d-m$ とする．n は都市数，c_1 および c_2 はそれぞれ $0 \leq c_1 \leq 1$，$0 \leq c_2 \leq 1$ を満たす定数，r_1，r_2 はそれぞれ $0 \leq c_1 \leq 1$，$0 \leq c_2 \leq 1$ の i に対して独立な乱数である．$[a]$ は実数 a の整数部を返す記号である．$C(\boldsymbol{x})$ は巡回路 $\boldsymbol{x}$ の総経路長を返す関数である．また，巡回路 $\boldsymbol{a}$ を巡回路 $\boldsymbol{b}$ に置き換える場合は代入記号を用いて $\boldsymbol{a} = \boldsymbol{b}$ と記述する．

1. すべての粒子 i の解 $\boldsymbol{x}_i$ に適当な巡回路を設定し，$\boldsymbol{p}_i = \boldsymbol{x}_i$ とする．また，自身の *pbest* と近傍 A_i の *pbest* からなる集合 A_i^p の中で最も総経路長が小さい巡回路を $\boldsymbol{l}_i$ とする．数式では (5.5) 式のように定義される．

$$\boldsymbol{l}_i = \arg\min_{\boldsymbol{p}_i \in \boldsymbol{A}_i^{\boldsymbol{p}}} C(\boldsymbol{p}_i) \tag{5.5}$$

2. すべての粒子 i について，次を適用する．
   (a) $\boldsymbol{p}_i$ から長さ $[c_1 r_1(n+1)]$ の連結成分をランダムに抽出し，部分経路 $\boldsymbol{p}_i'$ とする．
   (b) $\boldsymbol{l}_i$ から長さ $[c_2 r_2(n+1)]$ の連結成分をランダムに抽出し，部分経路 $\boldsymbol{l}_i'$ とする．
   (c) $\boldsymbol{x}_i$ に $\boldsymbol{p}_i'$ と $\boldsymbol{l}_i'$ を部分経路挿入法により挿入し，次の $\boldsymbol{x}_i$ とする．
   (d) $C(\boldsymbol{x}_i) < C(\boldsymbol{p}_i)$ ならば，$\boldsymbol{p}_i = \boldsymbol{x}_i$ とする．
3. すべての粒子 i について，次を適用する．
   (a) (5.5) 式に従って $\boldsymbol{l}_i$ を更新する．
4. すべての $\boldsymbol{p}_i$ の中で最もコストの小さい巡回路を暫定解 $\boldsymbol{s}$ とする．
5. 終了条件を満たしていなければ 2. へ戻る．
6. 暫定解 $\boldsymbol{s}$ を最終的に得られた解として返す．

アルゴリズム中で，2.，3. を 1 度実行することを 1 ステップとする．アルゴリズムの終了条件は，N ステップ経過や制限時間経過などを設定する．

上記のアルゴリズムでは，粒子の次数 k は $k = 2d$ であり，粒子数を m とすると，k は $2 \leq k < m$ を満たす偶数となることがわかる．しかし，特別な例として，$k = m - 1$ の場合のみ k が奇数になったとしても拡張サイクルの条件が満たされる．このとき，自分以外の粒子を近傍と設定していることとなり，*gbest* を用いる IPSO と同じ動作となる．これ以降，4 章で提案した IPSO は，$k = m - 1$ を用いる粒子間通信を制限した IPSO として扱い，粒子間通信を制限した IPSO を単に IPSO と呼ぶ．また，これ以降 4 章と 5 章の IPSO を明確に差別化したい場合には，4 章の IPSO を大域的 IPSO，5 章の IPSO を局所的 IPSO と呼ぶ．

5.5　数値計算実験:局所的 IPSO のパラメータによる性能変化

局所的 IPSO には，粒子数 m，*pbest* の重み c_1，*lbest* の重み c_2，各粒子の近傍数 k の 4 つのパラメータがある．そのうちの 3 つのパラメータ (m, c_1, c_2) は大域的 IPSO と共通しているが，近傍数 k の導入により，最適なパラメータが異なる可能性もあるので，4

表 5.2　実験 5.5.1 で用いるパラメータ

粒子数 m	$\{16, 32, 64\}$
pbest の重み c_1	$\{0, 0.1, 0.2, \cdots, 1.0\}$
gbest の重み c_2	$\{0, 0.1, 0.2, \cdots, 1.0\}$
粒子の近傍数 k	$\{2, 4, 6, m-1\}$
終了ステップ数	20000
使用する TSP	kroA100
パラメータセットあたりの試行回数	100

つともに調査を行う．パラメータの組ごとに 100 回試行し，平均誤差 $\bar{e_s}$ と pbest の平均密度 $\bar{D_p}$ をそれぞれ比較する．

5.5.1　解が十分に収束するまで探索を行う場合

まず，解の更新が起こらなくなるまで十分に探索する場合に，得られる解の誤差を最小にするためのパラメー タについて調査する．

実験条件

本実験に用いる IPSO のパラメータ (m, c_1, c_2, k) を含めた実験条件を表 5.2 に示す．(m, c_1, c_2, k) は表 5.2 に示した値を組合せて使用するが，それぞれの実験において，調査したい項目に関するパラメータに応じて，表 5.2 に示した範囲の中から一部の組合せについてのみ使用する．具体的にどのようなパラメータを固定していくかは，その都度説明する．

実験結果

まず，粒子数 m を 32 に固定し，(c_1, c_2, k) のすべての組合せに対して実験を行う．得られた平均誤差を図 5.2 に示す．それぞれ k を $2, 4, 6, m-1$ と設定したときのグラフであり，横軸が c_1，縦軸が c_2，濃淡は平均誤差 $\bar{e_s}$ の高低を表す．座標点が黒いほどが大きく，白いほどが小さい．濃淡のスケールは結果の見やすさの観点から対数にしている．この条件で最もが小さくなるパラメータの組合せは $(k, c_1, c_2) = (2, 0.7, 0.1)$ である．また，すべての k について，$0.4 \leq c_1 \leq 1.0$，$c_2 = 0.1$ の範囲内で $\bar{e_s}$ が小さくなることがわ

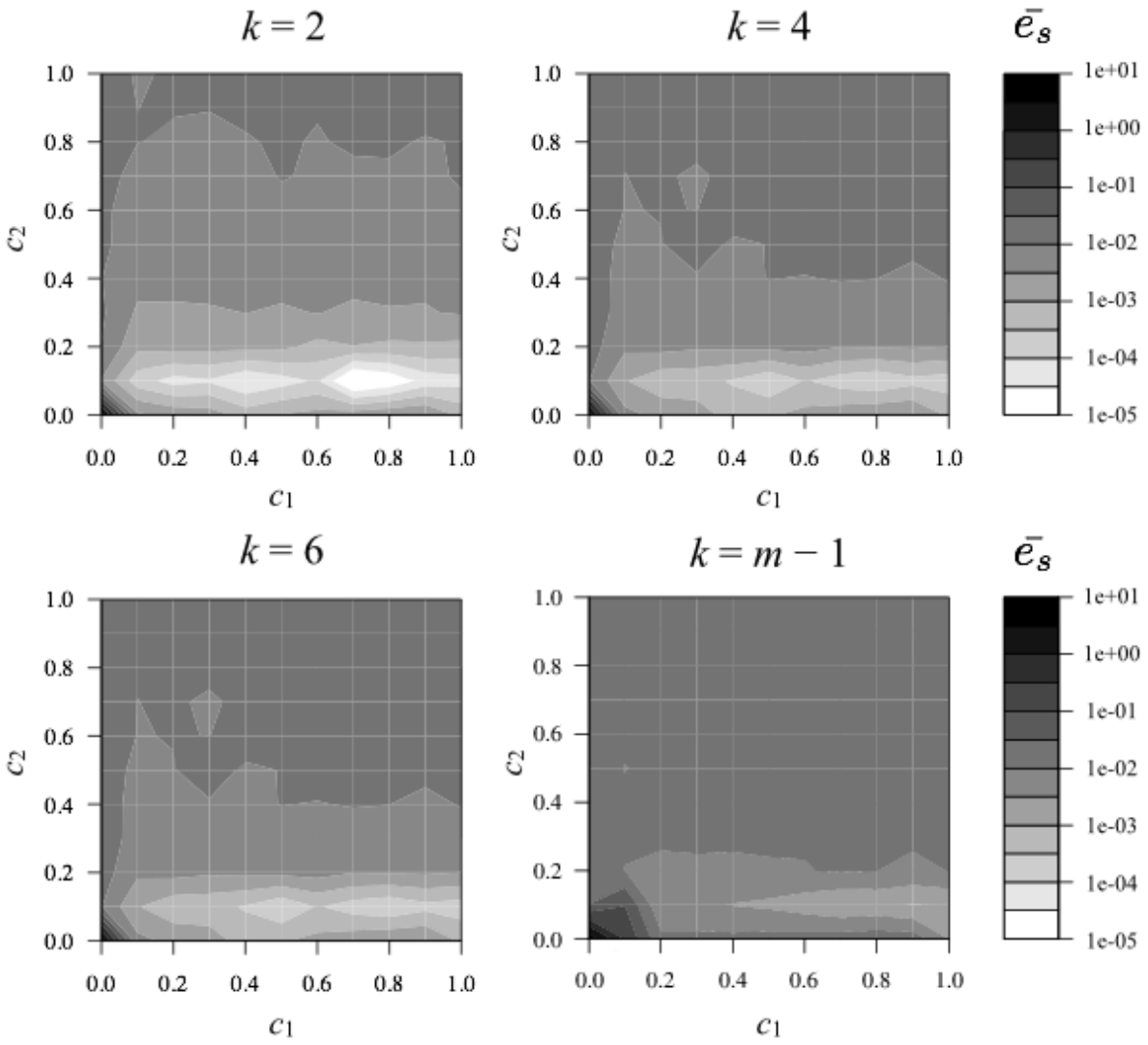


図 5.2　$m = 32$ の場合の平均誤差 $\bar{e_s}$．左上：近傍数 $k = 2$，右上：$k = 4$，左下：$k = 6$，右下：$k = m - 1$

.

かる．

次に，上記の実験の中から，$k = \{2, m-1\}$，$(c_1, c_2) = \{(0.7, 0.1), (0.2, 0.1), (0.7, 0.8)\}$ である，計 6 種のパラメータの組の結果に着目し，それらの平均誤差と *pbest* の平均密度の推移を図 5.3 に示す．横軸はステップ数，縦軸はそれぞれ平均誤差 $\bar{e_s}$ と *pbest* の平均密度 $\bar{D}_p$ を表す．$\bar{e_s}$ と $\bar{D}_p$ は 1000 ステップごとに記録している．$\boldsymbol{s}$ はそのステップが終了したのちに更新されたもの使用する．$\bar{e_s}$ のグラフの縦軸は結果の見やすさの観点から

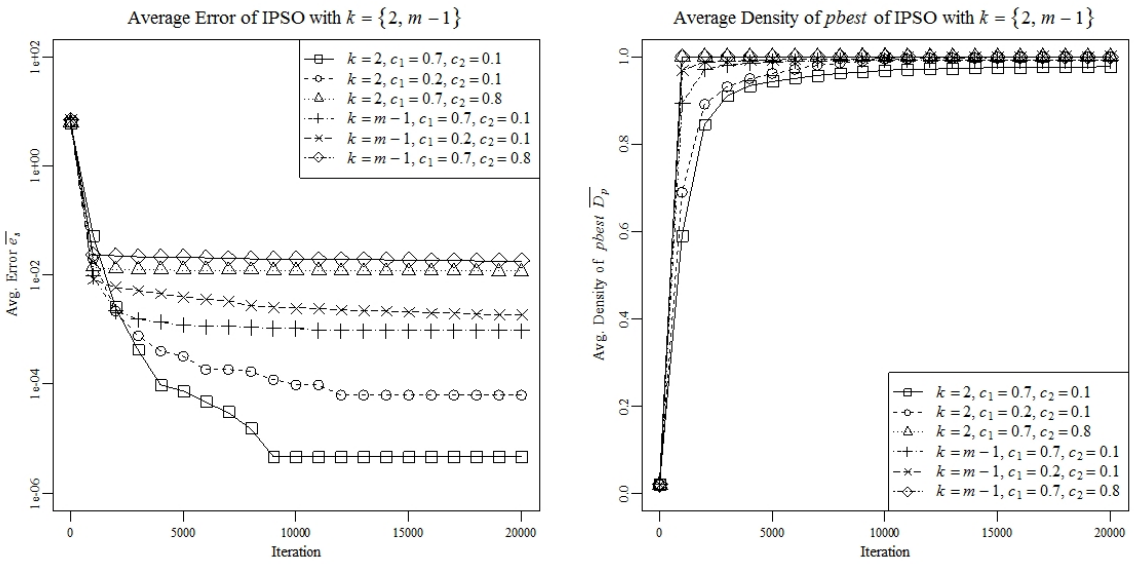


図 5.3　$m = 32$，$k = \{2, m-1\}$ のときの平均誤差 $\bar{e_s}$ の推移（左図）と *pbest* の平均密度 $\bar{D}_p$ の推移（右図）．

対数軸表示している．まず，平均誤差 $\bar{e_s}$ に着目する．$k = 2$ の場合，$(c_1, c_2) = (0.7, 0.1)$ のときに最も $\bar{e_s}$ が小さくなり，その次に $(c_1, c_2) = (0.2, 0.1)$ のときの $\bar{e_s}$ が小さく，$(c_1, c_2) = (0.7, 0.8)$ のときに最も $\bar{e_s}$ が大きい．解が収束するまでのステップ数は $\bar{e_s}$ が大きいパラメータの組ほど小さい．$k = m-1$ の場合についても上記と同様の結果である．また，c_1 と c_2 を固定した場合，$k = m-1$ より $k = 2$ の $\bar{e_s}$ が小さくなることがわかる．次に，pbest の平均密度 $\bar{D}_p$ に着目する．20000 ステップにおいてはすべてのパラメータの組でほぼ 1 に収束するが，はじめの数千ステップにおいて差がみられる．$(c_1, c_2) = (0.7, 0.1)$ の場合に最も収束が遅く，その次に $(c_1, c_2) = (0.2, 0.1)$ の場合の収束が遅く，$(c_1, c_2) = (0.7, 0.8)$ の場合に最も収束が早い．この傾向は $k = 2$ の場合にも $k = m-1$ の場合にも成り立つ．

次に，$(c_1, c_2) = (0.7, 0.1)$ と設定し，k をそれぞれ $2, 4, 6, m-1$ に設定したときの平均誤差 $\bar{e_s}$ と pbest の平均密度の推移 $\bar{D}_p$ を図 5.4 に示す．図 5.4 より，k が小さいほど，最終的に $\bar{e_s}$ が小さくなることと，$\bar{D}_p$ が大きくなりにくいことがわかる．ただし，1000 ステップ時点では k が大きいほど誤差が小さくなっている．

最後に，$(c_1, c_2) = (0.7, 0.1)$ とし，k を $2, m-1$ と m の範囲のすべての組合せに対して実験を行う．この実験で得られた平均誤差 $\bar{e_s}$ と pbest の平均密度 $\bar{D}_p$ の推移を図 5.5 に

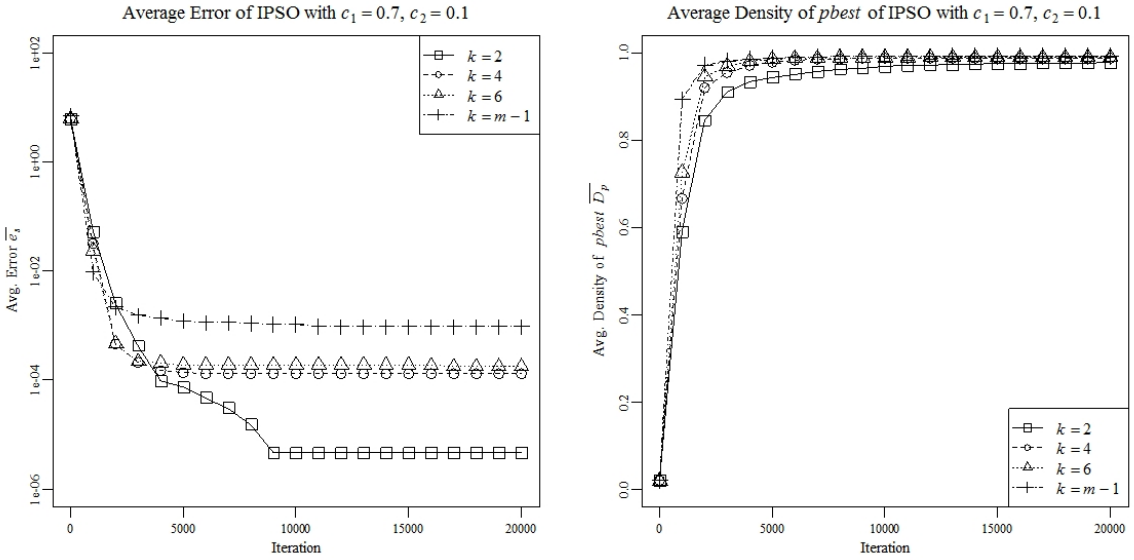


図 5.4　$m = 32, c_1 = 0.7, c_2 = 0.1$ のときの平均誤差 $\bar{e_s}$ の推移（左図）と *pbest* の平均密度 $\bar{D}_p$ の推移（右図）.

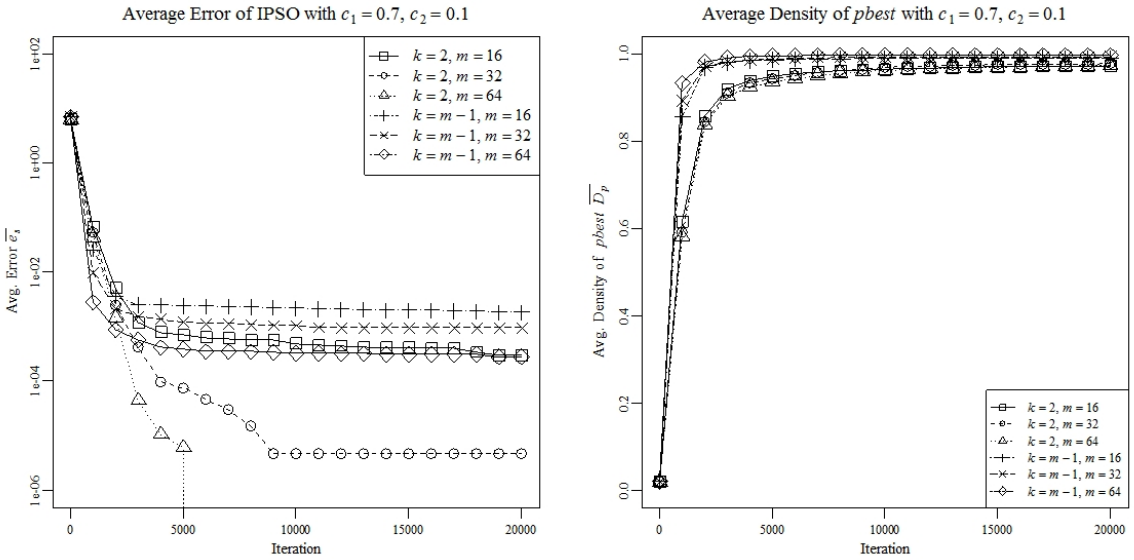


図 5.5　$c_1 = 0.7, c_2 = 0.1$ のときの平均誤差 $\bar{e_s}$ の推移（左図）と *pbest* の平均密度 $\bar{D}_p$ の推移（右図）.

示す．$k = 2, m = 64$ について 6000 ステップ以降のデータが表示されていないが，これ

は 6000 ステップ以降で $\bar{e_s} = 0$ であることを表している．図 5.5 から，$k = 2$，$k = m - 1$ のいずれの場合でも，粒子数 m が大きいほど $\bar{e_s}$ が小さくなることがわかる，$\bar{D_p}$ は，これまでの実験結果と比較して，m による変化は微小である．

実験考察

上記の結果から m, c_1, c_2 に関しては 4 章で得られた実験結果と同様の傾向が見られた．つまり，十分に時間をかけて探索を行う場合，c_1 を大きく，c_2 を小さくすることで，*pbest* の密度 D_p が小さく保たれ，群として広い範囲を探索し，高精度な解を発見することができる．また，粒子数 m を多くすることでも探索範囲が広がり，高精度な解を発見しやすくなる．さらに，上記の結果から，近傍数 k を変更しても m, c_1, c_2 の特性を変えないことも裏付けられた．

一方，新たに導入されたパラメータである近傍数 k については，図 5.4 から，$k = 2$ のときに最も *pbest* の密度の増加が緩やかで，最終的に平均誤差 $\bar{e_s}$ が小さくなることが示された．この理由は，$k = 2$ のときに粒子の近傍関係ネットワークの平均頂点間距離が最も長くなり，*lbest* の拡散にかかる時間が最も長くなるからである．この場合，*pbest* の多様さが比較的長く保たれ，粒子群は局所最適解に陥りにくい．一方，探索初期においては $k = m - 1$ の場合に $\bar{e_s}$ が最も小さい．$k = m - 1$ のときのは *gbest* を用いる PSO となり，すべての粒子が *gbest* の周辺を集中的に探索する．このため，少ないステップで $\bar{e_s}$ が減少しやすいが，$\bar{D_p}$ も増加しやすく，局所最適解に陥りやすいと考えられる．

5.5.2　制限時間が設定されている場合

次に，時間制限が設定されていて十分に探索が行えない場合に有効なパラメータの調査を行う．5.5.1 の実験において，十分に時間をかける場合，最適なパラメータセット $P_b(k, c_1, c_2)$ は $P_b = (2, 0.7, 0.1)$ であることを示した．しかし，探索の初期においては以外のパラメータを用いた方が小さな誤差となる場合がある．そのとき，同時に *pbest* の平均密度も高くなる．4 章で行った実験からも明らかであるが，短時間で誤差の小さな解を得るためには，粒子全体の探索範囲を狭めて集中的に探索したほうが良い．具体的に探索範囲を狭めるパラメータ調整に以下の方法があげられる．

- 近傍数 k を大きくし，同じ *lbest* を利用する粒子を増やす．
- *lbest* の重み c_2 を大きくし，解 $\boldsymbol{x}_i$ を $lbest\boldsymbol{l}_i$ に近づける．

表 5.3　実験 5.5.2 で使用するパラメータ

パラメータセット名 m	k	c_1	c_2	
P_1	32	2	0.7	0.1
P_2	32	$m-1$	0.7	0.1
P_3	32	2	0.1	0.8
P_4	4	2	0.7	0.1
P_5	4	$m-1$	0.1	0.8

- *pbest* の重み c_1 を大きくし，解 $\boldsymbol{x}_i$ を $pbest\boldsymbol{l}_i$ に近づける．
- 粒子数 m を小さくすることにより，*pbest* や *lbest* の種類が減り，生成される解の多様さが小さくなる．また，単位時間での探索なので，m を小さくすると，各粒子の更新回数が増える．

ただし，c_1 と c_2 の両方を大きくしたとしても，部分経路挿入法には *gbest* に優先則が働くため，*pbest* の部分経路は短くなってしまう．解 $\boldsymbol{x}_i$ を *pbest* に近づけるなら c_1 を，*lbest* に近づけるなら c_2 を，どちらか一方だけを大きくするべきである．上記に基づくパラメータ調整の有効性を確かめるために数値計算実験を行い検証する．

実験条件

上記を基に比較するパラメータセットを表 5.3 に示す．P1 は粒子数 m を 32 として，k, c_1, c_2 は十分に時間をかける場合に最適なパラメータである．P2，P3，P4 は P1 からそれぞれ，k，(c_1, c_2)，m を調整したパラメータセット，P5 は P1 からすべてのパラメータに調整を加えたパラメータセットである．

実験には 575 都市問題である rat575 を使用する．この TSP に対して，パラメータを P1 から P5 に設定した 5 種類の IPSO を制限時間 60 秒で 200 試行，制限時間 3600 秒で 30 試行ずつ行い，平均誤差 $\bar{e_{\boldsymbol{s}}}$ と *pbest* の平均密度 $\bar{D}_p$ を比較する．$\bar{e_{\boldsymbol{s}}}$ と $\bar{D}_p$ は，制限時間が 60 秒の実験では 3 秒ごとに，制限時間が 3600 秒の実験では 180 秒ごとに記録する．IPSO の実装は Scala 2.10.4 で行い，実験機の CPU は Intel Xeon E5520（2.26GHz），OS は Fedora 20，Java のバージョンは 1.7.0 を使用する．

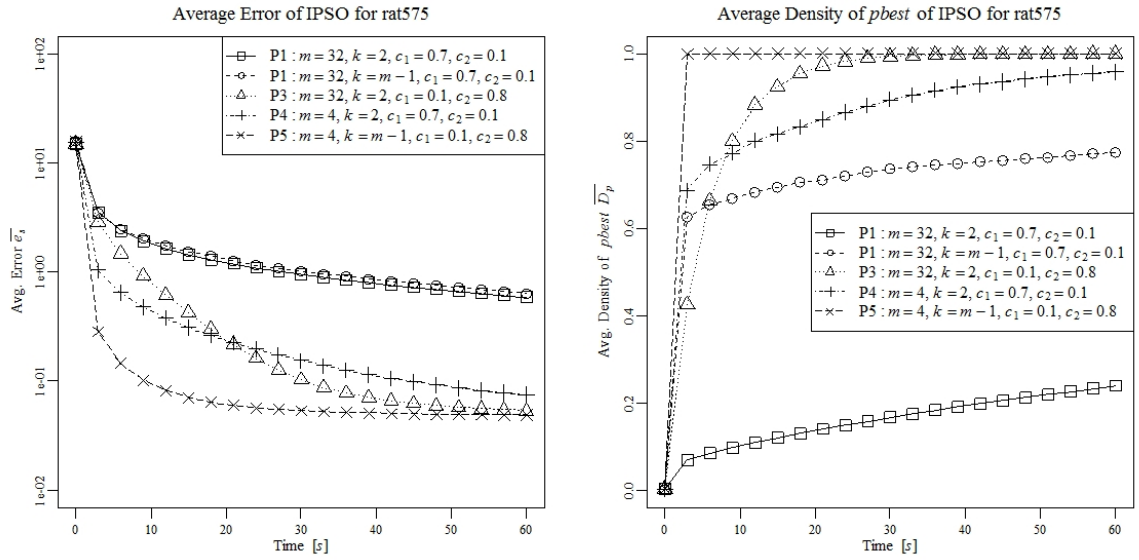


図 5.6　制限時間が 60 秒の平均誤差 $\bar{e_s}$ の推移（左図）と *pbest* の平均密度 $\bar{D}_p$ の推移（右図）.

実験結果

制限時間を 60 秒に設定したときの平均誤差 $\bar{e_s}$ と *pbest* の平均密度 $\bar{D}_p$ を図 5.6 に，制限時間を 3600 秒に設定したときの平均誤差 $\bar{e_s}$ と *pbest* の平均密度 $\bar{D}_p$ を図 5.7 に示す.

これらの結果より，P_1 は 900 秒以降については最も $\bar{e_s}$ が小さく，P_1 の一部のパラメータに変更を加えた P_2 から P_4 の $\bar{e_s}$ は 900 秒以前において一定の期間 P_1 より小さくなることが示された．つまり，時間制限が設定されている場合，P_1 以外のパラメータが最適である可能性がある．また，P_2 から P_4 の結果を比較すると，最も序盤にが小さくなるのは P_4 で，その次に P_3 であった．よって，できるだけ短時間で $\bar{e_s}$ を小さくしたい場合，m を小さくすることが最も効果的で，次に c_1 を小さくしつつ c_2 を大きくすることが効果的である．k を増やすことは m や (c_1, c_2) を調整するより効果が薄い．また，P_5 は 60 秒までは他のすべてのパラメータセットより $\bar{e_s}$ が小さくなったので，より短時間で $\bar{e_s}$ を小さくしたい場合はこれらのパラメータを同時に調整すると効果的であることが示された.

実験考察

上記の結果から m, c_1, c_2 に関しては 4 章で得られた実験結果と同様の傾向が見られた．これより，制限時間がある場合に関しても，近傍数 k の変更は m, c_1, c_2 の特性に影響を

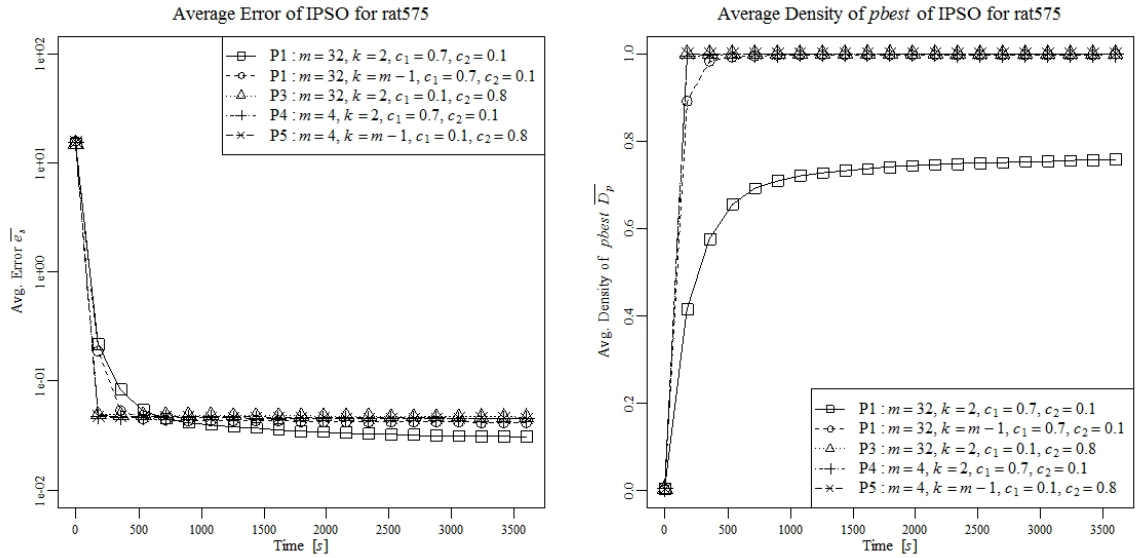


図 5.7　制限時間が 3600 秒の平均誤差 $\bar{e_s}$ の推移（左図）と $pbest$ の平均密度 $\bar{D_p}$ の推移（右図）.

与えないことが示された.

k に関しては，値を小さくすると $pbest$ の平均密度 $\bar{D_p}$ の上昇が緩やかになり，粒子間通信を制限したことによる $pbest$ の同一化の抑制効果が確認できる．$gbest$ 中心の探索に比べて広い範囲の解を探索しているため，探索初期においては精度の良い解が見つかりにくいが，十分に時間をかけて探索を行ったときに $\bar{D_p}$ が 1 より低い値に収束するため解が更新される可能性があり，その結果，平均誤差 $\bar{e_s}$ が小さくなると考えられる.

5.5.3　5.5 節の結論

本実験では，局所的 IPSO の 4 つのパラメータ，粒子数 m，$pbest$ の重み c_1，$lbest$ の重み c_2，各粒子の近傍数 k が探索性能に与える影響について調査を行った．m, c_1, c_2 を固定し，k を変化させた結果，k が大きいときほど探索初期における $pbest$ の収束が強く精度が比較的良い解が得られているが，解が収束してしまうのも早いことが示され，逆に，k が小さいときは $pbest$ の収束は遅く，最終的に誤差の小さい解を発見可能であることが示された．これは粒子群に適用した拡張サイクル型の通信トポロジが狙い通りに機能していることを示している．また，rat575 への適用実験の結果，$k = m - 1$ の場合は $\bar{D_p}$ が 1 に収束してしまうことが多かったが，$k = 2$ の場合は $\bar{D_p}$ は 0.8 付近に収束し，探索終盤の

解の更新可能性が高い状態であることが示され，実際に平均誤差 $\bar{e_s}$ が小さくなることが確認された．一方，粒子数 m，*pbest* の重み c_1，*lbest* の重み c_2 の 3 つのパラメータは，4 章の結果と一致し，m, c_1, c_2 の探索に及ぼす影響は k の値に依存しないことが示された．

5.6　結論

4 章で提案した IPSO を長時間実行した場合，すべての粒子の *pbest* が大きく似てしまうことから，解の探索範囲が狭くなりそれ以上の解の更新が望めない状態に陥ることがわかった．本章では，IPSO がより *pbest* の密度を低く保てるように改良することを目的とし，粒子間通信を制限した IPSO（局所的 IPSO）を構築した．局所的 IPSO は，それぞれの粒子はあらかじめ設定された一部の粒子（近傍粒子）の情報しか得ることができず，自分を含めた近傍粒子の中で最も経路コストが小さい解（*lbest*）を *gbest* の代わりに用いる．粒子の近傍関係を拡張サイクルによって構築することにより，*pbest* に多様性を維持しやすくなり，なおかつ，接続された粒子を通じて良い解を伝搬することができる．

次に，局所的 IPSO のパラメータによる探索性能の変化について調査するために数値計算実験を行った．本実験では，局所的 IPSO の 4 つのパラメータ，粒子数 m，*pbest* の重み c_1，*lbest* の重み c_2，各粒子の近傍数 k が探索性能に与える影響について調査を行った．その結果，k を小さくした場合，*pbest* の平均密度 $\bar{D_p}$ の上昇が緩やかになることが示された．これは，粒子が自身の *pbest* と通信可能な他の粒子の *pbest* の中で最も経路コストが小さい巡回路である *lbest* の部分経路を取り込むことと，群の中で最も良い *lbest* は平均距離が長いネットワークで構成された粒子の通信経路をゆっくりと伝搬するためと考えられる．また，$k = m - 1$ の場合は問題規模が大きくなったときにも $\bar{D_p}$ が 1 に収束してしまうことが多かったが，$k = 2$ の場合は $\bar{D_p}$ は 1 より小さな値に収束し，探索終盤の解の更新可能性が高い状態であることが示され，実際に平均誤差 $\bar{e_s}$ が小さくなることが確認された．一方，粒子数 m，*pbest* の重み c_1，*lbest* の重み c_2 の 3 つのパラメータは，4 章の結果と一致し，m, c_1, c_2 の探索に及ぼす影響は k の値に依存しないことが示された．

第 6 章

挿入操作 PSO 戦略の並列化

大規模な問題を扱うアプローチとして，また，処理を高速化する手法として，並列計算が注目されている．かつての並列計算は，多くのプロセッサをもつスーパーコンピュータや，複数のコンピュータを接続したコンピュータ・クラスターを対象としていた．しかし，近年，マルチコアプロセッサや Graphics Processing Unit(GPU) が一般的になったことで，並列計算技法の開発は重要性が高まった．TSP のような組合せ最適化問題の高速解法にも並列計算が積極的に用いられており，TSP を対象としたマルチコアを用いた並列タブーサーチ (TS)[43] や，２次割当問題を対象とした General-purpose computing on graphics processing units(GPGPU) を用いたアントコロニー最適化 (ACO)+TS[44] などが提案され，いずれも高速化を実現している．本章では，Java スレッドを用いて，4 章で提案した TSP 向け PSO のアルゴリズムを並列化し，並列化前後の性能を比較することで，並列化の有効性を示す．

6.1 対象とするコンピュータの構成

並列計算を行う上で，どのような構成のコンピュータ上で実装するかはとても重要である．表 6.1 に代表的な並列計算手法と装置の構成を示す．マルチスレッドはほぼすべてのコンピュータにおいて実現可能であるが，CPU のコア数は一般向けのコンピュータで 4 程度，サーバ用のコンピュータでも多くて 40 程度なので，並列数はあまり多くできない．GPU を用いた並列計算は近年注目されている方法で，特にディープラーニングの計算時間短縮の手段として有名である．GPU は内部にとても多くのコアを持っているが，基本的にはすべてのコアで同一の処理をするため，大規模行列の数値計算のような，限られた

表 6.1　代表的な並列計算方法と特長

手法	装置	並列数	メモリ	特徴
マルチスレッド	CPU	4〜40	4〜128GB	ハード遅延が少ない
GPU を用いた並列計算	CPU+GPU	2000〜4992	4〜32GB	制約が多い
コンピュータクラスタ 分散コンピューティング	複数台の コンピュータ	任意	任意	データ通信速度が ボトルネック

アプリケーションにしか有効ではない．コンピュータクラスタや分散コンピューティングはそれぞれ複数のコンピュータを LAN などで接続して並列に処理する手法であるが，各コンピュータの通信時間がボトルネックとなり，オブジェクトを頻繁にやり取りするようなアプリケーションには向かない．

本論文では，最も基本的な並列化である，マルチスレッドプログラミングによる並列化を行う．マルチスレッドプログラミングでは，他の並列計算手法ほどハードウェアの構成に気を使う必要がない．並列化したいプログラムにおける排他制御すべき箇所と同期処理すべき箇所が適切に把握できているならば，そのプログラムがどの程度高速化できるかの予想もある程度できる．

6.2　並列挿入操作 PSO 戦略のアプローチ

4 章で示した IPSO に対して，並行アルゴリズム設計モデル [45] に従ってタスク分解を試みると，最も効果的な並列化箇所はアルゴリズム中で大部分を占めている，粒子の更新処理である．例えば，m 個の粒子を l $(l < m)$ 個のコアで処理する場合，最も楽観的に見積もると，元の m/l の時間で処理が完了する．実際には，すべての粒子が共有している *gbest* $\boldsymbol{g}$ を安全に更新する必要があるため，ここまでの時間の短縮はできない．

粒子の更新処理を並列処理するにあたり，どのように *gbest* を更新，共有するかはとても重要な問題である．IPSO のアルゴリズムでは各粒子は常に最新の *gbest* を共有していたが，これを並列計算で行おうとするとそれなりの工夫が必要になる．代表的な方法としては，一定ステップごとに粒子の同期処理を行い，そこで *gbest* を更新する方法と，各粒子が新しい *gbest* を見つけたタイミングで *gbest* の更新を試み，これを排他制御を用いることで *gbest* を安全に更新する方法がある．同期を用いる方法は，毎ステップ同期を処理を行うことで，4 章で提案した IPSO とまったく同じ振る舞いにすることが可能である．しかし，同期処理の待ち時間が発生するため，実行時間があまり高速化しないことも考え

られる．一方，排他制御を用いる方法は，粒子の更新処理を待つ必要はないが，複数の粒子が同時に *gbest* の読み書きを試みた場合，どちらかの粒子に処理待ちが発生するため，*gbest* の更新頻度によっては効率よく並列化できないことが考えられる．

以上を考慮し，本章では同期並列 IPSO と非同期並列 IPSO を Java を用いて実装し，数値計算実験を通してそれぞれの有効性について検証する．

6.3　並列挿入操作 PSO 戦略のアルゴリズム

同期並列 IPSO と非同期並列 IPSO のアルゴリズムについて説明する．

6.3.1　同期並列 IPSO

同期並列 IPSO は，同期処理を行うことで *gbest* を安全に更新する並列 IPSO である．具体的には，粒子の更新処理を複数のスレッドを用いて並列に計算し，それらがすべて終わるのを待ってから *gbest* を更新する．*gbest* の更新処理が完了した後，再び粒子の更新処理を並列に計算する．粒子の更新処理において *gbest* は各粒子から読み込まれるだけなので，各粒子は並列に *gbest* にアクセスすることができる．一方，*gbest* の更新処理中は，それぞれの粒子は更新処理を中断するので，並列処理によって壊れた *gbest* にアクセスされる恐れもない．また，上記の実装では元々独立な処理であった粒子の更新処理を並列計算しているだけであるため，探索それ自体は IPSO と完全に同一であることが保証され，最終的に得られる解も同じである．

しかし，IPSO は粒子に対して比較的単純な更新処理を何度も適用することで解の探索を行う手法であるため，同期並列 IPSO は一度に並列計算する処理が短く，同期回数が大きくなる．同期処理には待ち時間が存在し，同期処理を多く行うアルゴリズムは並列計算を行ってもあまり高速化できないことがある．

そこで，同期並列 IPSO では，同期間隔 t というパラメータを導入し，t ステップ連続で粒子の更新を並列計算し，その後同期処理を行い *gbest* を更新する．同期間隔 t を大きく設定することで，探索中に行われる同期回数を小さくでき，より並列化による高速化が期待できる．しかし，その間 *gbest* は更新されないため，IPSO とは探索過程が異なり，最終的に得られる解の精度も異なると考えられる．$t = 1$ と設定した場合は IPSO と同一な探索過程となる．

同期並列 IPSO のアルゴリズムを以下に示す．IPSO のアルゴリズムと比較して，新し

く追加されたパラメータとしてスレッド数 l と同期間隔 t がある．スレッドは 1 つ以上の粒子の更新処理を行う．同期待ち時間はできるだけ短くしたいので，各スレッドが担当する粒子の数は同じ方がよい．つまり，粒子数 m はスレッド数 l の整数倍が望ましく，そのときに並列化による効果が最も現れる．

1. すべての粒子 i の解 $\boldsymbol{x}_i$ に適当な巡回路を設定し，$\boldsymbol{p}_i = \boldsymbol{x}_i$ とする．$\boldsymbol{g}$ はすべての粒子の *pbest* の中で最も総経路長が小さい巡回路とする．数式では (6.1) 式のように定義される．
$$\boldsymbol{g} = \arg\min_{\boldsymbol{x} \in \boldsymbol{P}} C(\boldsymbol{x}) \tag{6.1}$$
ここで，m は粒子数であり，*pbest* 集合 $\boldsymbol{P}$ は $\boldsymbol{P} = \{\boldsymbol{p}_1, \boldsymbol{p}_2, \cdots, \boldsymbol{p}_m\}$ である．
2. l 個のスレッドに粒子を均等に割り当てる．
3. すべてのスレッドが，次の処理を同時に開始し，t 回繰り返し適用する．
   (a) 自身に割り当てられているすべての粒子 i について，次を t 回適用する．
      i. $\boldsymbol{p}_i$ から長さ $[c_1 r_1 (n+1)]$ の連結成分をランダムに抽出し，部分経路 $\boldsymbol{p}'_i$ とする．
      ii. $\boldsymbol{g}$ から長さ $[c_2 r_2 (n+1)]$ の連結成分をランダムに抽出し，部分経路 $\boldsymbol{g}'$ とする．
      iii. $\boldsymbol{x}_i$ に $\boldsymbol{p}'_i$ と $\boldsymbol{g}'$ を部分経路挿入法により挿入し，次の $\boldsymbol{x}_i$ とする．
      iv. $C(\boldsymbol{x}_i) < C(\boldsymbol{p}_i)$ ならば，$\boldsymbol{p}_i = \boldsymbol{x}_i$ とする．
4. すべてのスレッドの処理が完了するまで待つ．
5. (6.1) 式に従って $\boldsymbol{g}$ を更新し，それを暫定解 $\boldsymbol{s}$ とする．
6. 終了条件を満たしていなければ 3. へ戻る．
7. 暫定解 $\boldsymbol{s}$ を最終的に得られた解として返す．

アルゴリズム中で，各スレッドが担当する粒子の更新を 1 度実行することを 1 ステップとする．並列計算しているため，あるタイミングですべてのスレッドが同じステップを計算している保証はないが，t ステップごとには同期処理が行われ，そのタイミングではすべての粒子は t ステップの処理がされている．

同期並列 IPSO は 2 種類の異なる機能を持つスレッドが処理を行っている．1 つは粒子の更新を担当するスレッドである．このスレッドは l 個存在する．もう 1 つはアルゴリズム全体を管理しているスレッドである．アルゴリズムの最初の粒子の初期化や，スレッドの同期がこのスレッドの担当である．これらのスレッドの機能の特徴から，前者を子ス

レッド，後者を親スレッドと呼ぶ．

6.3.2 非同期並列 IPSO

非同期並列 IPSO は，*gbest* の更新処理も含めた粒子の更新処理を並列計算する並列 IPSO である．明示的な同期処理を行わないため常に粒子の更新処理が行われる．このため，同期並列 IPSO より高速であることが予想される．しかし，ある粒子が *gbest* を読み込んでいる途中で別の粒子が *gbest* を更新しようとした場合，読み込まれた *gbest* は更新前の *gbest* であることも更新後の *gbest* であることもある．さらには，それらが混ざった不完全なものが読み込まれることもある．このようなことを避けるため，粒子全体で共有している *gbest* は排他制御により安全に読み書きされる必要がある．

排他制御は使用するマルチスレッドライブラリでサポートされていることがほとんどである．1 つの資源を複数のスレッドで共有する場合は，ミューテックス（MUTual EXclusion, Mutex）と呼ばれる機構がよく用いられる．資源にアクセスしたスレッドは資源に鍵をかける（ロックする）．資源に鍵がかけれられている間は，鍵をかけたスレッドが独占的に資源にアクセスでき，他のスレッドはロックが解除されるまで待つ．このように，1 つの資源にアクセスすることができるスレッドを 1 つに限定することにより，資源を安全に読み書きできる．

以下に非同期並列 IPSO のアルゴリズムを示す．

1. すべての粒子 i の解 $\boldsymbol{x}_i$ に適当な巡回路を設定し，$\boldsymbol{p}_i = \boldsymbol{x}_i$ とする．$\boldsymbol{g}$ はすべての粒子の *pbest* の中で最も総経路長が小さい巡回路とする．数式では (6.2) 式のように定義される．
$$\boldsymbol{g} = \arg\min_{\boldsymbol{x} \in \boldsymbol{P}} C(\boldsymbol{x}) \tag{6.2}$$
ここで，m は粒子数であり，*pbest* 集合 $\boldsymbol{P}$ は $\boldsymbol{P} = \{\boldsymbol{p}_1, \boldsymbol{p}_2, \cdots, \boldsymbol{p}_m\}$ である．
2. l 個のスレッドに粒子を均等に割り当てる．
3. すべてのスレッドが，次の処理を同時に開始する．
   (a) 排他制御を用いて *gbest* $\boldsymbol{g}$ にアクセスし，現在の最新の *gbest* のコピー $\boldsymbol{g_c}$ を作る．
   (b) 自身に割り当てられているすべての粒子 i について，次を適用する．
      i. $\boldsymbol{p}_i$ から長さ $[c_1 r_1 (n+1)]$ の連結成分をランダムに抽出し，部分経路 $\boldsymbol{p}'_i$ とする．

ii. $\boldsymbol{g_c}$ から長さ $[c_2 r_2(n+1)]$ の連結成分をランダムに抽出し，部分経路 $\boldsymbol{g'_c}$ とする．

iii. $\boldsymbol{x}_i$ に $\boldsymbol{p}'_i$ と $\boldsymbol{g}'_{\boldsymbol{c}}$ を部分経路挿入法により挿入し，次の $\boldsymbol{x}_i$ とする．

iv. $C(\boldsymbol{x}_i) < C(\boldsymbol{p}_i)$ ならば，$\boldsymbol{p}_i = \boldsymbol{x}_i$ とする．

(c) 自身に割り当てられているすべての粒子の *pbest* $\boldsymbol{p}_i$ 中で最も良い解 $\boldsymbol{b}$ が $\boldsymbol{g_c}$ より良い解ならば，排他制御を用いて *gbest* $\boldsymbol{g}$ の更新を試み，$C(\boldsymbol{b}) < C(\boldsymbol{g})$ ならば，$\boldsymbol{g} = \boldsymbol{b}$ とする．

(d) 終了条件が満たされていない場合，(a) に戻る．

4. すべてのスレッドの処理が完了するまで待つ．
5. *gbest* $\boldsymbol{g}$ を最終的に得られた解として返す．

非同期並列 IPSO においても，同期並列 IPSO と同様に，2 種類の異なる機能を持つスレッドが処理を行っている．1 つは粒子の更新を担当するスレッドであり，もう 1 つはアルゴリズム全体を管理しているスレッドである．前者を子スレッド，後者を親スレッドと呼ぶ．

6.4　数値計算実験：並列 IPSO の性能調査

並列 IPSO と IPSO をいくつかの異なるパラメータを用いて比較し，それぞれの手法の特徴を調査する．

6.4.1　並列 IPSO の実装

本実験ではプログラミング言語 Scala を用いて並列 IPSO と IPSO を実装し，数値計算実験を行う．Scala は Java 仮想マシン上で動作し，Java のプログラムと容易に連携できる言語である．マルチスレッドプログラミングをするにあたり，Java のスレッドライブラリを利用する．また，アルゴリズムの実装は過度な工夫を行わずにデザインパターンを用いる．用いるデザインパターンについてはその都度示す．以下に実装した並列 IPSO について示す．

同期並列 IPSO（Synchronous Parallel IPSO, SPIPSO）

SPIPSO は 6.3.1 節で示した同期並列 IPSO を実装した TSP ソルバである．SPIPSO では，粒子の更新処理を担当する複数の子スレッドと，子スレッドの結果を集約して *gbest*

の更新を行う親スレッドが必要であるが，これらは Future パターン [46, 47] と Worker Thread パターン [47] を用いて実装する．具体的には，親スレッドが子スレッドの結果（*gbest* の更新有無）を受け取るために Future パターンを使用し，子スレッドを実行するときに既存スレッドを再利用できるように Worker Thread パターンを利用する．

Read-Write Lock パターンを用いた非同期並列 IPSO（Asynchronous Parallel IPSO using the Read-Write Lock pattern, APIPSO(RWL)）

APIPSO(RWL) は 6.3.2 節で示した非同期並列 IPSO を Read-Write Lock パターン [48, 47, 45] を用いて実装した TSP ソルバである．Read-Write Lock パターンは，書き込み処理より読み込み処理の頻度が高いプログラムで効果的な排他制御のデザインパターンである．共有オブジェクトである *gbest* は毎ステップ必ずすべての粒子からアクセスされるため，Read-Write Lock パターンによって排他制御することによって高速化が見込める．

Producer-Consumer パターンを用いた非同期並列 IPSO（Asynchronous Parallel IPSO using the Producer-Consumer pattern, APIPSO(PC)）

APIPSO(PC) は 6.3.2 節で示した非同期並列 IPSO を Producer-Consumer パターン [48, 47, 45] を用いて実装した TSP ソルバである．Producer-Consumer パターンは，一般的にはデータを生産するスレッド（Producer）とデータを消費するスレッド（Consumer）に速度差があるときに有効なデザインパターンである．APIPSO に対しては，新たな *gbest* を発見する子スレッドを Producer，各子スレッドの *gbest* の更新要求に対して実際に更新を行うスレッドを Consumer と切り分けることにより適用できる．APIPSO(PC) の Producer は新しい *gbest* を発見した場合，他の Producer が *gbest* を読み込み中であったとしても，Producer と Consumer の連絡役である Channel にデータを送信するだけで良いので，Read-Write Lock パターンで発生する書き込み待ちが起こらない．また，*gbest* の書き換えを行うスレッドが Consumer スレッドのみのため，書き込みの衝突を考慮する必要がなく，書き込み処理がアトミックであるか，書き込み途中に読み込み処理が発生したとしても安全なデータ構造であれば，*gbest* を排他制御する必要がなくなるメリットもある．その代わりに，Consumer に 1 スレッド割り当てなければならないことと，Consumer がすぐに *gbest* の更新処理を行う保証がないという特徴がある．

6.4.2　実験設定

数値計算実験には，子スレッド数 l が $\{1, 2, 3, 4, 6, 8, 12\}$ の 5 種類の並列 IPSO を用いる．子スレッドとはそれぞれの並列 IPSO において粒子の更新を担当するスレッドであり，IPSO の処理の大部分を占める．すべての並列 IPSO は子スレッドの他に 1 つの親スレッドを持つ．親スレッドとは，*gbest* や子スレッドの初期化や子スレッドの処理状態の監視を行うスレッドである．子スレッドが実行中は待機しているため，CPU リソースをほとんど奪わない．SPIPSO の親スレッドに関しては，この他に子スレッドの同期後に *gbest* を更新する役目がある．その他に，APIPSO(PC) では 1 つの Consumer スレッドが動作している．Consumer スレッドは通常は待機状態にあり，子スレッドからの *gbest* の更新通知を受け取ると，*gbest* の更新を行う．親スレッドと比較して，Consumer スレッドは子スレッドと CPU リソースを奪い合うことが多いが，子スレッドの処理の方が圧倒的に量が多い．本実験では CPU として Intel Xeon E5-2603 v3 × 2 (1.60GHz, 12 コア) を使用するが，子スレッド数にかかわらず，常に 12 コア使用可能な状態である．そのため，実験で使用する子スレッド数が 12 より小さい場合，子スレッド以外のスレッドは子スレッドの CPU リソースを奪うことなく実行されると考えられる．一方，子スレッド数が 12 の場合は子スレッドど子スレッド以外のスレッドが 12 個のコアを奪い合うことがどこかで起こる．

SPIPSO については，*gbest* の更新間隔 t を $\{1, 10, 50, 100\}$ の 4 種類を用いる．各 IPSO について共通するパラメータを表 6.2 に示す．ベンチマーク問題は pr439，pr1002 の 2 種類を使用する．pr439 では $(c_1, c_2) = (0.7, 0.05)$，pr1002 では $(c_1, c_2) = (0.05, 0.95)$ を用い，部分経路挿入法における *pbest* と *gbest* の比率を変えたときに並列化効率が変化するか調査する．

6.4.3　実験結果および考察

並列 IPSO と IPSO を上記の設定で 100 回試行し，平均実行時間と実行時間の標準偏差，平均誤差を表 6.3 から表 6.8 に示す．表には子スレッド数ごとの各 IPSO の結果を示している．同期並列 IPSO に関しては同期間隔 t が $\{1, 10, 50, 100\}$ のそれぞれ場合の結果が示されており，非同期 IPSO に関しては，Read-Write Lock パターン実装と Producer-Consumer パターン実装のそれぞれの結果が示されている．それぞれの行は使

表 6.2　並列 IPSO の性能調査実験において，IPSO と並列 IPSO に共通するパラメータ

使用する TSP	pr439	pr1002
粒子数 m	48	48
$pbest$ の重み c_1	0.7	0.05
$gbest$ の重み c_2	0.05	0.95
終了ステップ	100000	80000
試行回数	100	

用する子スレッド数を表している．IPSO はシングルスレッドのプログラムであるため，子スレッド数 1 の箇所にのみ結果が記述されている．

まず，SPIPSO について説明する．pr439 に対する平均実行時間を示した表 6.3 の結果から，子スレッド数が 2 以上の SPIPSO は IPSO より実行時間が減少しており，並列化によって高速化可能であることが確認できる．しかし，$t = 1$ の場合に IPSO と等価なプログラムであるが，子スレッド数を 6 にしたときに最も実行時間が小さくなり，それ以上の子スレッドに分割した場合は遅くなっている．これは，子スレッドの実行時間の差が大きくなり，同期待ち時間が長くなったためであると考えられる．本実験では，$m = 48$ と粒子数を一定にしているため，子スレッド数が 6 の場合には各子スレッドが扱う粒子は 8 つとなり，子スレッド数が 12 の場合には各子スレッドが扱う粒子は 4 つとなる．粒子の更新法である部分経路挿入法は $[0, 1)$ の範囲の一様乱数によって処理量が変動する．しかし，部分経路挿入法の処理量の期待値は定義可能であり，大数の法則に従って，試行回数を増やすことで平均値は期待値に近づく．このため，子スレッドが扱う粒子数が十分である場合は，各子スレッドごとの処理量が近くなるため，子スレッドの同期待ち時間は小さくなるが，子スレッドが扱う粒子数が小さい場合は，乱数によって各子スレッドの処理量が不均一となり，同期待ち時間が大きくなる．これは，$t \geq 10$ の場合，子スレッド数を 8 や 12 に増やしても平均実行時間は減少していることからも説明できる．また，同期間隔 t については，$t \geq 10$ で結果に大きな違いはなかったため，pr439 に対しては子スレッド 1 つあたり 40 回の部分経路挿入法を実行すれば，同期待ち時間をある程度小さくできると考えられる．しかし，大数の法則を考慮するならば，t は大きいほど同期待ち時間が小さくなることが期待できる．以上の考察は，表 6.6 から，pr1002 についても同様であることがわかる．ここまでで，同期間隔を十分大きくすると SPIPSO の実行時間が減少す

表 6.3　pr439 に対する平均実行時間

子スレッド	IPSO	SPIPSO				APIPSO	
		$t = 1$	$t = 10$	$t = 50$	$t = 100$	RWL	PC
1	234.172	239.640	235.050	235.412	237.310	236.805	235.429
2	-	155.489	162.652	180.981	184.728	168.688	175.230
3	-	101.551	111.515	118.518	123.129	115.395	117.835
4	-	78.917	83.982	89.403	93.939	99.855	98.926
6	-	58.675	53.864	61.403	63.736	63.388	66.026
8	-	73.405	64.799	63.110	60.838	48.466	47.999
12	-	72.858	48.060	49.174	51.980	34.141	34.180

表 6.4　pr439 に対する実行時間の標準偏差

子スレッド	IPSO	SPIPSO				APIPSO	
		$t = 1$	$t = 10$	$t = 50$	$t = 100$	RWL	PC
1	1.512	1.693	1.644	0.992	1.107	1.619	1.718
2	-	2.927	2.723	2.589	2.330	1.638	1.438
3	-	1.461	2.005	1.377	3.076	1.587	2.352
4	-	1.580	1.086	1.702	1.097	1.183	1.449
6	-	2.073	2.151	1.842	2.338	2.114	1.506
8	-	1.211	0.527	0.594	0.436	2.436	1.918
12	-	0.837	0.592	1.129	1.095	1.154	1.154

ることを示したが，同期間隔を大きくした場合，誤差が大きくなることが表 6.5，表 6.8 からわかる．特に *gbest* の重み c_2 を大きく設定した pr1002 に対して $t \geq 50$ とした場合は，IPSO の平均誤差と比較して 10 倍以上悪い誤差である．このため，同期間隔 t の調整では平均実行時間と平均誤差がトレードオフの関係になっており，目的に応じて調整することが必要なことがわかる．

次に，APIPSO について説明する．pr439 に対する平均実行時間を示した表 6.3 の結果から，Read-Write Lock パターン（RWL）と Producer-Consumer パターン（PC）の

表 6.5　pr439 に対する平均誤差（%）

子スレッド	IPSO	SPIPSO				APIPSO	
		$t=1$	$t=10$	$t=50$	$t=100$	RWL	PC
1	1.439	1.217	1.657	2.289	2.840	1.361	1.308
2	-	1.301	1.571	2.186	2.829	1.380	1.339
3	-	1.363	1.745	2.163	2.812	1.309	1.334
4	-	1.246	1.590	2.191	2.799	1.404	1.367
6	-	1.322	1.532	2.004	2.877	1.327	1.287
8	-	1.317	1.790	2.296	2.700	1.411	1.335
12	-	1.359	1.681	2.208	2.918	1.243	1.483

どちらの実装においても，並列化によって高速化可能であることが確認できる．また，RWL と PC の pr439 に対する平均実行時間はほぼ同一の結果であった．RWL と PC の違いは共有オブジェクトの管理方法だけであるので，pr439 規模の問題では *gbest* の書き込み待ちはあまり発生しないと考えられる．平均誤差については，表 6.5 の結果から，RWL と PC の両方で IPSO と同等の誤差の解が得られていることがわかる．この結果から，IPSO は粒子の更新処理を非同期に実行しても，得られる解の精度は悪くならないことが確認できる．一方，pr1002 では RWL と PC に差が現れている．表 6.6 に示されている平均実行時間から，RWL より PC の方がより高速であることがわかる．これは，RWL の方で *gbest* の書き込み待ちや読み込み待ちが発生しているからであると考えられる．pr439 より pr1002 の方が約 2 倍読み書きに時間が必要であり，その分，*gbest* にアクセスするスレッドが待つ機会が多くなりやすい．

最後に，IPSO，SPIPSO，APIPSO を比較する．まず，SPIPSO の $t \geq 50$ は得られる解の誤差が IPSO の結果からかなり悪くなってしまい，実用的ではないので，比較から除外する．残った手法を比較すると，子スレッド数が 6 まではほぼ同一の平均実行時間であり，子スレッドが 8，12 のときは APIPSO の方がより短時間で処理が完了している．さらに，問題規模が大きいときは APIPSO(PC) が最も高速であると期待できる．これは本実験を総合的に考慮すると，子スレッド 1 つあたりに割り当てられる粒子が十分に大きいときは SPIPSO と APIPSO にはそこまで大きな差は現れないと予想でき，逆に，子スレッド 1 つあたりに割り当てられる粒子が少ないときに APIPSO の方が高速になると考

表 6.6　pr1002 に対する平均実行時間

子スレッド	IPSO	SPIPSO				APIPSO	
		$t = 1$	$t = 10$	$t = 50$	$t = 100$	RWL	PC
1	493.666	502.712	502.294	511.736	508.197	495.764	494.875
2	-	330.937	355.375	386.076	384.133	360.460	358.384
3	-	225.241	237.300	257.721	254.121	245.833	244.384
4	-	177.506	182.461	195.895	198.327	205.423	201.921
6	-	127.811	128.447	131.849	132.803	180.401	133.437
8	-	147.828	136.360	123.937	120.217	138.224	99.381
12	-	133.110	99.870	105.852	95.817	94.820	71.421

表 6.7　pr1002 に対する実行時間の標準偏差

子スレッド	IPSO	SPIPSO				APIPSO	
		$t = 1$	$t = 10$	$t = 50$	$t = 100$	RWL	PC
1	3.390	4.003	2.279	1.394	9.157	4.562	3.231
2	-	5.472	7.739	5.569	8.086	1.477	3.715
3	-	4.956	4.976	3.087	3.379	2.101	3.479
4	-	3.286	4.280	2.885	3.240	3.801	3.586
6	-	4.130	3.536	3.202	4.068	10.499	4.751
8	-	1.821	1.526	1.016	1.228	8.622	3.889
12	-	1.088	1.554	2.036	1.397	4.278	2.399

えられる．また，APIPSO(PC) がどのような条件でも安定して高速であると考えられる．

6.5　結論

本章では，IPSO のアルゴリズムの粒子の更新が独立であることに着目し，マルチスレッドプログラミングによる IPSO の並列アルゴリズムである並列 IPSO を提案した．並列 IPSO のアルゴリズムとしては，大きく分けて同期並列 IPSO と非同期並列 IPSO が

表 6.8　pr1002 に対する平均誤差（%）

子スレッド	IPSO	SPIPSO				APIPSO	
		$t = 1$	$t = 10$	$t = 50$	$t = 100$	RWL	PC
1	4.698	4.680	4.984	65.153	1197.974	4.798	4.705
2	-	4.861	4.874	91.237	1182.671	4.876	4.765
3	-	4.667	5.135	81.779	1167.882	4.887	4.756
4	-	4.775	4.878	80.834	1106.810	4.878	4.653
6	-	4.712	4.919	79.351	1081.152	4.799	4.600
8	-	4.784	5.032	69.644	1019.420	4.848	4.641
12	-	4.637	5.055	62.015	1145.407	4.640	4.714

考えられ，それぞれの実装である SPIPSO，APIPSO(RWL)，APIPSO(PC) の 3 種を実装し，数値計算実験を通して性能比較を行った．この実験から，すべての並列 IPSO において並列化による高速化が確認されたが，SPIPSO は子スレッドが扱う粒子数が小さい場合に同期待ち時間が長くなってしまうことや同期間隔が大きいときに平均誤差とても悪くなってしまうこと，APIPSO(RWL) は都市数が大きい問題に場合に同期待ち時間が長くなってしまうことを示し，APIPSO(PC) がどのような条件でも安定して高速であることを示した．具体的には，12 コアの CPU を使用した APIPSO(PC) は，IPSO と比較して，pr439 と pr1002 の両方の問題にたいして，約 7 倍高速に同程度の精度の解を発見可能であることがわかった．

第 7 章

結論

本研究では，GA などの既存の近似解法と比較して，高速に高精度の解を発見可能な近似解法の構築を目的として，PSO を基にしたアルゴリズムである挿入操作 PSO 戦略（IPSO）を提案し，その有効性について検証した．

第 4 章では IPSO について詳細に説明した．PSO は多点探索を行う実数値最適解問題向けの手法であり，遺伝的アルゴリズムより高速に高精度な解を得られることが知られている．本研究では PSO のこのような特徴に着目し，TSP を PSO の探索戦略で解くことで，高速に高精度な解を発見可能ではないかと考えた．PSO で組合せ最適化問題である TSP を直接扱うことは難しいため，IPSO では，解表現を巡回路とし，粒子の解の更新を *pbest* や *gbest* の部分経路を自身に取り込む部分経路挿入法を用いて行うように変更した．この変更により，PSO の *pbest* と *gbest* の周囲を探索するというポリシーを継承しながら，TSP を解くことを可能にした．また，IPSO のパラメータによる探索性能の変化について調査するために数値計算実験を行った．その結果，IPSO の最適なパラメータは問題の規模や制限時間によって変わることが示された．IPSO には，粒子数 m，*pbest* の重み c_1，*gbest* の重み c_2 の 3 つのパラメータがあるが，(c_1, c_2) に関しては，c_1 を小さく，c_2 を大きく設定すると，粒子は *gbest* の周辺を集中的に探索するため，短時間である程度良い解が得られ，逆に，c_1 を大きく，c_2 を小さく設定すると，粒子は各々の *pbest* の周囲を探索し，結果として粒子群は広い範囲の解を探索するので，収束までの時間は長くなるがより小さい誤差の解が得られることがわかった．次に，IPSO の有効性を示すために，SA と GA との性能比較を行った．問題規模の違う 10 種類の TSP に対して，同一の制限時間を設定した IPSO，SA，GA を適用し，得られる解の誤差を比較した結果，すべての TSP に対して IPSO が最も優れていることが示された．また，この実験から，IPSO のパ

ラメータチューニングに関する指針が示された．まずはじめに，c_1 に大きな値，c_2 に小さな値，粒子数 m に大きな値を設定する．これは十分時間がある場合に誤差を最小化できるパラメータセットである．もし，このパラメータセットで制限時間内に解が収束しない場合は，m を減らす．m を減らしても収束しない場合には c_1 に小さく，c_2 を大きくする．このようにパラメータを調整すると，その制限時間で得られる解の精度を最もよくできることがわかった．

第 5 章では 4 章での結果をもとに，より高精度な解が得られるように挿入操作 PSO 戦略を改良した．4 章の実験から，探索が停滞してしまう原因は，すべての粒子の *pbest* の辺の共有率が高くなってしまうことで，各粒子が似たような範囲の解を探索してしまい，探索範囲が狭くなることであったため，*pbest* の密度を低く保つことが解決策と考えた．そこで，*gbest* を廃止して群全体での巡回路の共有をやめる代わりに，各粒子に通信可能な粒子を複数設定し，その小群の中の最良解 *lbest* を利用するように変更した．また，粒子の接続関係を，ネットワークの平均距離が長い拡張サイクルによって構築することにより，良い解の情報が粒子群全体に拡散する時間が長くなることが期待され，*pbest* の密度を低く保つことができると考えた．次に，粒子間通信を制限した IPSO のパラメータによる探索性能の変化について調査するために数値計算実験を行った．その結果，k を小さくした場合，*pbest* の平均密度の上昇が緩やかになることが示され，通信ネットワークの平均距離を長くすればするほど良い解の拡散が遅くなり *pbest* の収束が遅くなることがわかった．また，このとき，*gbest* を用いた IPSO より小さい誤差の解を発見できることが多くなることも示された．

第 6 章では IPSO のアルゴリズムの粒子の更新が独立であることに着目し，マルチスレッドプログラミングによる IPSO の並列アルゴリズムである並列 IPSO を提案した．並列 IPSO のアルゴリズムとしては，大きく分けて同期並列 IPSO と非同期並列 IPSO が考えられ，それぞれの実装である SPIPSO，APIPSO(RWL)，APIPSO(PC) の 3 種を実装し，数値計算実験を通して性能比較を行った．この実験から，すべての並列 IPSO において並列化による高速化が確認され，そのなかでも APIPSO(PC) がどのような条件でも安定して高速であることを示した．具体的には，12 コアの CPU を使用した APIPSO(PC) は，IPSO と比較して，実行時間を最大で約 86% 削減できることを示した．

謝辞

本研究を進めるにあたり，研究内容に関して様々な助言をしていただき，また調査のあり方など細部にわたるご指導を賜りました山本雅人教授，研究とは何か，研究がいかに面白いか，研究者としての姿勢など助言していただき，研究活動のモチベーションの維持を助けていただきました飯塚博幸准教授，北海道大学を退官されたあとも私のことを気にかけていただき，食事に誘っていただき励ましてくださるなど，公私にわたってお心づかいいただきました古川正志前教授に心より感謝いたします．

同級生の皆さんは全員修士で卒業しましたが，その後もお付き合いがあり，時には刺激的な議論を頂き，精神的にも支えられました．本当にありがとうございました．

そして，ご多忙にも関わらず，公私に渡り有益な助言，意見，協力を頂きました自律系工学研究室の皆様に深く感謝いたします．

参考文献

[1] Keld Helsgaun. An effective implementation of the lin–kernighan traveling salesman heuristic. *European Journal of Operational Research*, Vol. 126, No. 1, pp. 106–130, 2000.

[2] Keld Helsgaun. General k-opt submoves for the lin–kernighan tsp heuristic. *Mathematical Programming Computation*, Vol. 1, No. 2-3, pp. 119–163, 2009.

[3] 永田裕一, 小林重信. 巡回セールスマン問題に対する交叉: 枝組み立て交叉の提案と評価. 人工知能学会誌, Vol. 14, No. 5, pp. 848–859, 1999.

[4] Y. Nagata and S. Kobayashi. A powerful genetic algorithm using edge assembly crossover for the traveling salesman problem. *INFORMS Journal on Computing*, Vol. 25, No. 2, pp. 346–363, 2013.

[5] K. Wang, L. Huang, C. Zhou, and W. Pang. Particle swarm optimization for traveling salesman problem. In *Proceedings of International Conference on Machine Learning and Cybernetics*, pp. 1583–1585, 2003.

[6] Wei Pang, Kang Ping Wang, Chun Guang Zhou, and Long Jiang Dong. Fuzzy discrete particle swarm optimization for solving traveling salesman problem. In *Proceedings of the fourth International Conference on Computer and Information Technology, 2004. CIT'04.*, pp. 796–800. IEEE, 2004.

[7] Xiaohu H Shi, Yanchun Chun Liang, Heow Pueh Lee, C Lu, and QX Wang. Particle swarm optimization-based algorithms for tsp and generalized tsp. *Information Processing Letters*, Vol. 103, No. 5, pp. 169–176, 2007.

[8] Michael Sipser. *Introduction to the Theory of Computation*, Vol. 2. Thomson Course Technology Boston, 2006.

[9] Stephen A Cook. The complexity of theorem-proving procedures. In *Proceedings*

of the third annual ACM symposium on Theory of computing, pp. 151–158. ACM, 1971.

[10] Richard M Karp. *Reducibility among combinatorial problems*. Springer, 1972.

[11] 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, 1995.

[12] 久保幹雄, J.P. ペドロソ. メタヒューリスティクスの数理. 共立出版, 2009.

[13] 古川正志, 川上敬, 渡辺美知子, 木下正博, 山本雅人, 鈴木育男. メタヒューリスティクスとナチュラルコンピューティング. コロナ社, 2011.

[14] John H Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.* U Michigan Press, 1975.

[15] David E Goldberg, et al. *Genetic algorithms in search optimization and machine learning*, Vol. 412. Addison-wesley Reading Menlo Park, 1989.

[16] 前川, 玉置, 喜多, 西川. 遺伝的アルゴリズムによる巡回セールスマン問題の一解法. 計測自動制御学会誌論文集, Vol. 31, pp. 598–605, 1995.

[17] H. Braun. On solving travelling salesman problems by genetic algorithms. *Parallel Problem Solving from Nature*, pp. 129–133, 1991.

[18] J.Y. Potvin. Genetic algorithms for the traveling salesman problem. *Annals of Operations Research*, Vol. 63, No. 3, pp. 337–370, 1996.

[19] 永田裕一. 局所的多様性の損失を考慮した評価関数を用いた ga の tsp への適用. 人工知能学会誌, Vol. 21, No. 2, pp. 195–204, 2006.

[20] A. Colorni, M. Dorigo, V. Maniezzo, et al. Distributed optimization by ant colonies. In *Proceedings of the first European conference on artificial life*, Vol. 142, pp. 134–142, 1991.

[21] E. Ridge and D. Kudenko. Tuning the performance of the mmas heuristic. *Lecture Notes in Computer Science*, Vol. 4638, p. 46, 2007.

[22] M. Dorigo and G. Di Caro. The ant colony optimization meta-heuristic, new ideas in optimization, 1999.

[23] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *science*, Vol. 220, No. 4598, p. 671, 1983.

[24] M. Malek, M. Guruswamy, M. Pandya, and H. Owens. Serial and parallel simulated annealing and tabu search algorithms for the traveling salesman problem. *Annals of Operations Research*, Vol. 21, No. 1, pp. 59–84, 1989.

[25] F. Glover, M. Laguna, et al. *Tabu search*. Springer, 1997.

[26] C.N. Fiechter. A parallel tabu search algorithm for large traveling salesman problems. *Discrete Applied Mathematics*, Vol. 51, No. 3, pp. 243–267, 1994.

[27] 古川正志, 渡辺美知子, 松村有祐. 局所クラスタリング組織化法による tsp の解法. 日本機械学会論文集 (C 編), Vol. 71, No. 711, pp. 3189–3195, 2005.

[28] 崔志傑, 松本直文. 巡回セールスマン問題に対する pso-aco ハイブリッド手法の提案 (システム情報工学科). 足利工業大学研究集録, Vol. 42, pp. 53–60, 2008.

[29] Gerhard Reinelt. Tsplib—a traveling salesman problem library. *ORSA journal on computing*, Vol. 3, No. 4, pp. 376–384, 1991.

[30] Tsplib. http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/.

[31] J. Kennedy and R.C. Eberhart. Particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks*, pp. 1942–1948, 1995.

[32] Riccardo Poli, James Kennedy, and Tim Blackwell. Particle swarm optimization. *Swarm Intelligence*, Vol. 1, No. 1, pp. 33–57, 2007.

[33] James Kennedy and William M Spears. Matching algorithms to problems: an experimental test of the particle swarm and some genetic algorithms on the multimodal problem generator. In *Proceedings of the IEEE international conference on evolutionary computation*, pp. 78–83. Citeseer, 1998.

[34] Russ C Eberhart, James Kennedy, et al. A new optimizer using particle swarm theory. In *Proceedings of the sixth international symposium on micro machine and human science*, Vol. 1, pp. 39–43. New York, NY, 1995.

[35] 毛利進太郎. メタ・ヒューリスティックにおける順列間距離の研究. 神戸学院大学経営学論集, Vol. 1, No. 2, pp. 147–154, 2005.

[36] 増田直紀, 今野紀雄. 複雑ネットワーク　基礎から応用まで. Technical report, 近代科学社, 2010.

[37] B. Bollobás. *Random Graphs Second Edition*. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 2001.

[38] Duncan J Watts and Steven H Strogatz. Collective dynamics of 'small-world'networks. *nature*, Vol. 393, No. 6684, pp. 440–442, 1998.

[39] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *science*, Vol. 286, No. 5439, pp. 509–512, 1999.

[40] Mark EJ Newman and Duncan J Watts. Scaling and percolation in the small-

world network model. *Physical Review E*, Vol. 60, No. 6, p. 7332, 1999.

[41] Andrew D Barbour and Gesine Reinert. Small worlds. *arXiv preprint cond-mat/0006001*, 2000.

[42] Béla Bollobás and Oliver Riordan. The diameter of a scale-free random graph. *Combinatorica*, Vol. 24, No. 1, pp. 5–34, 2004.

[43] 榎原博之, 長谷川裕之介, 田中裕也. マルチコアを考慮した並列タブーサーチアルゴリズム. 研究報告数理モデル化と問題解決（MPS）, Vol. 2010, No. 23, pp. 1–7, 2010.

[44] 筒井茂義. Gpu を用いた高速並列進化計算による組合せ最適化問題へのアプローチ. オペレーションズ・リサーチ：経営の科学, Vol. 57, No. 5, pp. 261–269, 2012.

[45] Clay Breshears. *The Art of Concurrency*. O'Reilly Media, 2009.

[46] Henry C Baker Jr and Carl Hewitt. The incremental garbage collection of processes. In *ACM Sigplan Notices*, Vol. 12, pp. 55–59. ACM, 1977.

[47] 結城浩. 増補改訂版 Java 言語で学ぶデザインパターン入門 マルチスレッド編. ソフトバンク クリエイティブ株式会社, 2006.

[48] Mark Grand. *Patterns in Java: a catalog of reusable design patterns illustrated with UML*. John Wiley & Sons, 2003.