ハイパフォーマンス
コンピューティング 82−27
（２０００．８．５）

クラスタリングによる抽象化を用いた巡回セールスマン問題の分散処理解法

渥美 清隆　　新妻 清三郎　　古澤 征平

静岡大学 工学部 システム工学科

概要:

巡回セールスマン問題のような大規模組み合せ最適化問題を扱う場合, 分割統治の戦略を再帰的に適用することが考えられる. 本論文で述べるアルゴリズムは分割統治の戦略を用いず, この問題を帰納的に解く. すなわち, (1) 与えられた問題に対して最も単純な問題を設定し, それを解く. (2)$k-1$番目までの解が選られたならば, それを利用してk番目の解を解き, 徐々に元の問題に近づく. ということである. この戦略では問題を分割する必要がないため, 解の全体像を考慮しながら, 計算することが可能である. 本論文では, 最も抽象度の高い問題から徐々に元の問題に近づく戦略による巡回セールスマン問題の逐次計算機による解法と並列計算機による解法について述べ, その性能を実験的に評価する.

キーワード:

抽象化, 帰納的アルゴリズム, 並列・分散処理, 巡回セールスマン問題, クラスタリング.

A Distributed Algorithm for Traveling Salesman Problem Using Abstraction Generated by a Clustering Technique

Kiyotaka ATSUMI, Seizaburo NIITSUMA and Shohei FURUSAWA

Department of Systems Engineering, Shizuoka University

Abstract:

We will apply recursively divide and conquer strategy to large size combinatorial optimization problem like Traveling Salesman Problem. This paper describe an algorithm not using divide and conquer strategy for Traveling Salesman Problem. This algorithm based on inductive strategy. In other words: (1) We build the simplest problem from original problem, and solve the simplest problem. (2) We approach step by step to original problem by building and solving the kth problem using $k-1$th problem. This paper show experimentally this algorithm's performance on a sequential computer and a parallel computer.

Keywords:

Abstraction, Inductive Algorithm, Parallel/Distributed Processing, Traveling Salesman Problem, Clustering.

1 はじめに

巡回セールスマン問題は都市数が増えると，その問題の難しさが爆発的に増大することで有名である．このように組み合わせ爆発が起こる問題に対して，多くの場合は分割統治を再帰的に行う戦略を取る．分割された部分問題は，通常隣接する他の部分問題とは独立に解かれ，その後，必要な最適化を行なって組み上げていく．

本論文で示すアルゴリズムも $\hat{k}$-means 法 [2] により都市を分割してクラスタにまとめるが，それは分割統治の戦略とは全く異なっている．このアイディアは都市が配置された地図の上に非常に粗いメッシュを重ね，都市が 1 つ以上所属するようなメッシュを新しい問題の都市の 1 つに例える．このようにすると都市の数を大幅に減らすことが出来る．

次にこのメッシュをほんの少しだけ細かくすることにより，少しだけ，都市が増える新しい問題を作ることができる．この新しい問題を粗いメッシュの時に解いた解を用いて求解することが出来れば，徐々にメッシュを細かくすることにより効率的に解を求めることが出来る．この方法は，部分問題が急激に小さくなる分割統治の戦略に比べ，緩やかに問題が複雑になっていくため，解の全体像を考慮しながら求解することが可能である．

本論文では，$\hat{k}$-means 法を用いた巡回セールスマン問題の求解アルゴリズム [2] を用い，他の基本的な求解アルゴリズムと比較実験をすることで評価する．また，並列計算機上で動作するアルゴリズムに変更し，本論文で示すアルゴリズムがどの程度の性能になるのかを実験したので，それを報告する．

2 抽象化と解探索アルゴリズム

2.1 巡回セールスマン問題

巡回セールスマン問題 (Traveling Salesman Problem: TSP) はいくつかの種類に分けることが出来るが，本論文ではユークリッド巡回セールスマン問題 (Euclidean TSP: ETSP) を扱う．ETSP とは次のような問題である．

「ユークリッド空間内に n 個の都市があり，都市間の全ての距離が分かっている．このとき，ある都市から出発してすべての都市を 1 度だけ通り，ふたたび出発した都市に戻る最短の巡回路を求めよ．」

ETSP では，n 個の都市座標が具体的に与えられて，初めて問題が定まる．この具体例をインスタンス I と呼ぶ．I は次のような都市座標の集合から成る．

$$I = \{c_1,\ c_2,\ \cdots,\ c_n\},$$

ここで，$c_i = (x_i,\ y_i)$，$x_i,\ y_i$ は都市 c_i の座標である．各都市の距離はユークリッド距離として定義される．$d(c_i,\ c_j)$，$c_i,\ c_j \in I$ は 2 つの都市 c_i と c_j の距離を返す関数であり，$d(c_i,\ c_j) = d(c_j,\ c_i)$ である．

2.2 $\hat{k}$-means 法による抽象化

都市数の多いインスタンス I は，非常に難しい問題である．そこで，$\hat{k}$-means 法 [2] によりいくつかの都市をまとめ，抽象化した新しいインスタンス $\hat{I}^k$ を作り，そこから，徐々に元のインスタンスに近づきながら，解を構成することを考える．このとき作られるインスタンス $\hat{I}^k$ を k 番目の商インスタンス，元のインスタンスを原インスタンスと呼ぶことにする．商インスタンス $\hat{I}^k$ は次のように構成される．

$$\hat{I}^k = \{rp_{\hat{c}_1^k},\ rp_{\hat{c}_2^k},\cdots,\ rp_{\hat{c}_k^k}\}$$

ここで，$\hat{c}_j^k$ は k 番目の商インスタンスに所属する j 番目のクラスタで，$\bigcup_{j=1}^{k} \hat{c}_j^k = I$，$|\hat{c}_j^k| \geq 1$ となるように原インスタンスの都市は商インスタンスの各クラスターに必ず分配されている．$|\hat{c}_j^k|$ はクラスタ $\hat{c}_j^k$ に分配された都市の数を表す．$rp_{\hat{c}_j^k}$ はクラスタ $\hat{c}_j^k$ の代表点を表す．本論文では代表点は各クラスタに所属する都市の平均ベクトルで表すこととする．

クラスタに分割された各都市はそのクラスタに対する所属度 $b(c_i)$，$c_i \in I$ を保持する．本論文では，所属度 $b(c_i)$ は現在所属しているクラスタの代表点までの距離とし，距離が長い程その所属しているクラスタへの結び付きが弱いと仮定する．

最も抽象度の高い商インスタンス $\hat{I}^1$ は次のように構成される．

$$\hat{I}^1 = \{rp_{\hat{c}_1^1}\}$$

ここで，$\hat{c}_1^1 = I$ で，原インスタンスと同じである．つまり，最初に与えられた原インスタンス全体に対する，たった 1 つの代表点からなる商インスタンスを構成する．

次に商インスタンス $\hat{I}^{k-1}$ が構成された時，商インスタンス $\hat{I}^k$ を構成する方法を述べる．

現在のクラスタリングされた状態の中で，最も所属度が弱い都市 c_{week} を選び，それを新しいクラスタ代

表点 $rp_{\hat{c}_k^{k-1}}$ とする．各都市は $rp_{\hat{c}_1^{k-1}} \cdots rp_{\hat{c}_k^{k-1}}$ までのクラスタ代表点の中で最も近いクラスタ代表点を選び，そのクラスタの所属に入る．最後に各クラスタは更新された都市に基づき，代表点の再計算を行なう．更新された各クラスタの代表点により新しい商インスタンス $\hat{I}^k$ を構成する．

新しい代表点の近傍のクラスタは，そこに所属する都市が大幅に変わる．しかし新しい代表点から遠いクラスタに含まれる都市は，ほとんど変更が無いまま新しい商インスタンスの要素となることが実験的に分っている．そこで，適切な距離を設定し，都市の所属の再計算を行なう範囲を限定することとした．これをスコープと呼ぶこととする．あるクラスタの代表点から，最も所属度の弱い都市までの距離をクラスタ半径とした時，スコープの範囲はクラスタ半径最大のクラスタ半径の 10 倍とした．このスコープの設定により時間計算量は大幅に改善された．時間計算量の証明は次の通りである．

補題 1 スコープを設定しない $\hat{k}$-means による，商インスタンス $\hat{I}^{k-1}$ から $\hat{I}^k$ への再構成に必要な時間計算量は都市数が n の時，平均的に $O(n^2)$ である．

(証明) 各都市はどのクラスタに所属するか，全ての組み合わせについて計算を行なう．クラスタ数は $O(n)$ なので，必要な計算量は $O(n^2)$ である．□

補題 2 クラスタ分割が進み，各クラスタの大きさについて，スコープ内に存在するクラスタ数は平均で $O(1)$ である．

(証明) クラスタ同士は都市を共有することはなく，領域として完全に分離されている．各クラスタ半径が 0 から r であり，その平均が $r/2$ であるとすると，最大クラスタ半径の c 倍の領域内には平均 $4c^2$ 個のクラスタが存在する．しかし，c が定数ならば，スコープ内に存在するクラスタ数も平均的に $O(1)$ である．□

定理 1 スコープを設定した $\hat{k}$-means による，商インスタンス $\hat{I}^{k-1}$ から $\hat{I}^k$ への再構成に必要な時間計算量は都市数が n で，補題 2 が満たされる時，$O(n)$ である．

(証明) 補題 1 と異なる点は，再計算の対象となるクラスタ数である．補題 2 により，スコープ内のクラスタ数は平均的に $O(1)$ なので，再構成に必要な時間計算量は平均的に $O(n)$ である．□

2.3 ETSP に対する解探索アルゴリズム

商インスタンス $\hat{I}^1$ の解は自明である．商インスタンス $\hat{I}^{k-1}$ の各クラスタ代表点の座標とクラスタ代表点 $rp_{\hat{c}_k^k}$ を除く $rp_{\hat{I}^k}$ の各クラスタ代表点の座標はそれほど変更が無いと仮定すると，$\hat{I}^k$ では，$\hat{I}^{k-1}$ で求められた解に対して，新しく出来たクラスタ代表点 $rp_{\hat{c}_k^k}$ を何処に挿入し，解全体をどのように更新するのかという問題になる．ここでは，現在の経路の辺集合を E として，単純に次のような経路上の辺を探索して，そこに挿入することした．

$$min_{(i,j)\in E}\{d(i,rp_{c_k^k}) + d(j,rp_{c_k^k}) - d(i,j)\}$$

全体の解探索アルゴリズムは，都市数を n とすると次のようになる．この方法では，1 つのクラスタの挿入に $O(n)$，全体では $O(n^2)$ の時間計算量がかかる．

アルゴリズム 1

1. 原インスタンスから商インスタンス $\hat{I}^1$ を構成する．
2. 商インスタンス $\hat{I}^1$ の解を構成する．
3. 商インスタンスに含まれるクラスタ代表点の数が n になるまで以下を繰り返す．
   (a) $\hat{k}$-means 法により，商インスタンス $\hat{I}^{k-1}$ から商インスタンス $\hat{I}^k$ を構成する．
   (b) 商インスタンス $\hat{I}^{k-1}$ の解を利用して，商インスタンス $\hat{I}^k$ の解を求める．

2.4 解探索アルゴリズムの分散化

2.3 節で述べたアルゴリズムを並列計算機に実装する場合，いくつかの並列化が考えられる．ここでは，やや精度を犠牲にして，実行速度が速くなる問題分割を行なうこととした．具体的には，商インスタンス $\hat{I}^{100}$ 程度まで，逐次計算機で処理し，そこで構成される解の経路に沿って，負荷が平等になるように各プロセッサにクラスタを分配し，それぞれの計算機で独立に解探索を行なうようにした．

この方法ではプロセッサ間の通信がほとんど無いので，プロセッサ数に比例した速度向上が望める．問題点としては，商インスタンス $\hat{I}^{100}$ の時点での経路を分割してしまうので分割された付近の経路や，隣接したクラスタであるにも拘らず，複数のプロセッサに

表 1: テストセット a

Instance	Size	Instance	Size	Instance	Size
d198	198	u1060	1060	vm1748	1748
lin318	318	pcb1173	1173	rl1889	1889
fl417	417	d1291	1291	u2152	2152
pcb442	442	rl1323	1323	pr2392	2392
u574	574	fl1400	1400	pcb3038	3038
p654	654	u1432	1432	fl3795	3795
rat783	783	fl1577	1577	fnl4461	4461
pr1002	1002	d1655	1655	rl5934	5934

表 2: テストセット b

Instance	Size	Instance	Size
pla7397	7397	d15112	15112
rl11849	11849	d18512	18512
usa13509	13509	pla33810	33810
brd14051	14051	pla85900	85900

跨がってしまう場合, それぞれのクラスタに所属している都市が他のプロセッサのクラスタへ移動出来ないこのによる解精度が悪化が考えられる. この点については, 3 節で明らかにする.

3 実験

実験では, 次の 2 点について検証する.

1. 逐次計算機において, 他の一般的な ETSP の近似解法と比べて, どの程度の計算速度と精度があるのか.
2. 並列計算機において, 逐次アルゴリズムと比べ速度と精度にどの程度の変化があるのか.

アルゴリズムを評価するために, ETSP のベンチマーク問題として公開されている TSPLIB[1] から幾つかのインスタンスを選らぶことにした. TSPLIB の中には 14 都市問題から 85900 都市問題まである. Reinelt[1] は様々な手法を比較するために TSPLIB から表 1 のインスタンスを選び実験している. 本論文でもこれにならい, これらのインスタンスをテストセット a として, 提案するアルゴリズムの評価に用いることとした. また, テストセット a で選ばれたインスタンスよりもさらに都市数の多いインスタンスを表 2 のように選びテストセット b として加えることとした.

この実験に使用した計算機は, 逐次計算機として Intel Celeron/400MHz を用いた. また, 並列計算機としては, Intel Dual Celeron/400MHz を 1 台, AMD K6-2/500MHz を 1 台, AMD K6-2/335MHz を 1 台の計 4 プロセッサで, それぞれを 100Base-TX のネットワークで接続したものを用いた. プログラミング言語は GNU C++ Ver.2.95, 並列計算のための通信ライブラリには LAM/MPI Ver.6.3.2 を用いた.

3.1 逐次アルゴリズムとの比較

まず, 比較対象として選らんだ 2 つのアルゴリズムについて説明する. 1 つは最遠挿入法 (Farthest Insertion: FI)[3], もう 1 つはセービング法 (Saving)[3] である. 最遠挿入法は, 現在経路上にある全ての都市の集合を V, 経路上にない全ての都市の集合を W として, 次の計算

$$\min_{x \in W} \{\max_{y \in V} \{d(x,y)\}\}$$

により経路上にない都市を 1 つ選らび経路に挿入する方法である. このアルゴリズムを単純にインプリメントした時, 時間計算量は $O(n^3)$ である.

セービング法は, 最初に都市 c_1 を除く全ての都市について, 都市 c_1 と往復する経路を構成する. 次に, 都市 c_j から一度都市 c_1 を経由して c_k に向う全ての経路について,

$$\max_{j,k} \{d(c_j,c_1) + d(c_1,c_k) - d(c_j,c_k)\}$$

を選択し, 全ての都市を巡回する経路が出来るまで繰り返す方法である. このアルゴリズムも単純にインプリメントした時, 時間計算量は $O(n^3)$ である.

FI(1), Saving(2), 本論文で述べたアルゴリズム (3) の 3 つについて, 述べたテストセット a に含まれる全てのインスタンスの解を逐次計算機で求めた. この時の実行時間は図 1, 精度は表 3 に示す. 精度の単位は百分率である.

実行速度の比較では, 計算量の評価と比例して, 本論文で提案する方法が速いことが分かる. インスタンス rl5934 では, FI が 2352 秒, Saving が 40336 秒に対して, 本論文の方法は 86 秒である.

精度の比較では, 平均値でこそ本論文の方法は他のアルゴリズムよりも良い結果を出している. しかし個々のインスタンスに対する結果を見ると, 安定した結果を出力する Saving に比べて, FI と本論文の方法はかなりばらつきがある. 2 つのアルゴリズムでは, 既に形成されている経路にある都市 (または地点) を挿入するアルゴリズムが同一であることに関係があると思われる.

表 3: 逐次計算における精度の比較

Instance	(1)	(2)	(3)	Instance	(1)	(2)	(3)	Instance	(1)	(2)	(3)
d198	19.42	10.78	4.78	pcb1173	20.67	12.99	14.89	u2152	22.02	12.80	18.53
lin318	9.95	8.79	8.02	d1291	25.17	15.85	19.09	pr2392	14.03	14.38	12.75
fl417	27.62	12.02	7.66	rl1323	26.06	10.36	23.97	pcb3038	17.09	11.63	12.97
pcb442	14.17	11.01	13.77	fl1400	18.93	12.07	6.50	fl3795	30.94	16.86	19.28
u574	12.12	11.35	7.02	u1432	16.58	8.07	10.29	fnl4461	16.79	11.36	9.84
p654	37.24	16.74	10.31	fl1577	29.05	10.49	15.76	rl5934	24.19	12.48	32.57
rat783	11.86	10.59	11.18	d1655	19.73	13.04	16.45				
pr1002	12.16	11.76	8.08	vm1748	19.19	13.12	9.91				
u1060	13.55	10.75	7.29	rl1889	22.05	12.16	28.42	Avarage	20.02	14.10	13.00

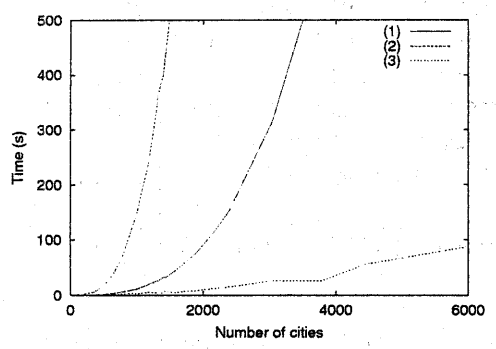

図 1: 逐次計算における実行時間の比較

3.2 並列計算時の性能評価

並列計算機への実装は次のようにした. 1つのプロセスがマスターとして, 初期の商インスタンス $\hat{I}^{100}$ を構成し, その商インスタンスの巡回路に沿って経路を分割し, スレーブに渡す. スレーブが計算している間, マスターは休眠状態となる. そのため, マスターに専用のプロセッサを割当てることを止め, あるスレーブとプロセッサを共有することとした.

テストセットaとテストセットbに含まれる全てのインスタンスの解を, スレーブ数を1から4まで変化させ求めた. この時の実行時間は図2に, それぞれのインスタンスに対する解の精度を表4に示した.

実行時間で注意しなければならない点は, スレーブ数が増えると, それぞれのスレーブが処理する都市数が減るため, 商インスタンス $\hat{I}^{k-1}$ から $\hat{I}^{k}$ を求めた後の解構成の処理が軽くなるということである. そのため, インスタンス pla85900 では, スレーブ1つの時の実行時間が14146秒であるのに対し, スレーブ4つの時の実行時間は1511秒と9.4倍も速くなった. 1つのプロセッサ上で4つのスレーブをシミュレートした場合, その実行時間は4848秒だったので, 実際には1つのプロセッサに比べて4つのプロセッサを利用した場合は3.2倍速くなったことになる. 速度向上がこの程度になっている理由として, 各プロセッサの能力の差と, 分割された仕事量との差にギャップがあることが挙げられる.

精度については, 経路を分割する影響が若干見られる. しかし, インスタンスによっては, 分割した方がかえって解の精度が良くなる場合もある. 全体の平均でも, 3つのスレーブを使った場合よりも, 4つのスレーブを使った場合の方が良い結果となっている. それでも, 逐次アルゴリズムと比べると4つのスレーブを使った場合でだいたい1%程度の解の悪化が予想される.

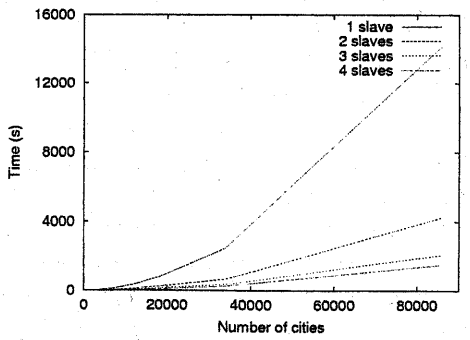

図 2: 並列計算におけるスレーブ数に対する実行時間

表 4: 並列計算における精度

	Number of Slaves					Number of Slaves			
Instance	1	2	3	4	Instance	1	2	3	4
d198	4.78	4.43	4.69	4.68	rl1889	15.70	29.65	28.56	17.14
lin318	8.02	8.09	8.09	8.04	u2152	18.53	18.14	18.25	18.05
fl417	7.66	7.26	7.88	7.18	pr2392	12.75	13.04	16.19	13.04
pcb442	13.77	13.22	14.28	13.92	pcb3038	12.97	13.59	14.47	14.02
u574	7.02	7.56	7.73	8.00	fl3795	19.28	19.38	19.23	19.42
p654	10.31	10.30	10.47	11.27	fnl4461	9.84	10.08	10.63	9.37
rat783	11.18	12.99	12.72	13.39	rl5934	27.90	31.34	35.15	31.09
pr1002	8.08	8.58	8.31	9.09	pla7397	11.68	12.17	12.90	12.72
u1060	7.29	7.17	7.64	7.49	rl11849	24.09	30.04	23.14	33.86
pcb1173	14.89	15.03	15.9	15.97	usa13509	10.95	19.78	19.79	11.78
d1291	19.09	20.29	20.68	19.69	brd14051	18.00	10.18	10.57	11.18
rl1323	23.97	17.57	25.14	18.80	d15112	10.86	11.30	11.25	11.42
fl1400	6.50	6.48	6.67	6.52	d18512	13.63	10.71	13.93	14.34
u1432	10.29	11.15	11.99	12.06	pla33810	16.65	18.79	17.24	18.98
fl1577	15.76	15.22	14.96	15.02	pla85900	15.93	14.69	16.61	17.14
d1655	16.45	17.15	16.44	16.48					
vm1748	9.91	10.94	10.45	11.51	Avarage	13.55	14.26	14.75	14.15

4 考察

実験のために実装した全てのプログラムは, 基本的なアルゴリズムの部分のみをそのまま実現したものとなっている. そのため例えば, candidate sets[1], heap, kd-tree など, プログラムの実行速度を向上させるような手法は, 何も採用していない. 記憶容量についても, 都市数 n について $O(n)$ 程度になるようにした. これらの手法を組み合わせることにより, さらに数倍から数十倍以上に実行速度を向上させることが出来ることが分っている.

アルゴリズムの並列化については, 今回採用した方法以外にも, 様々な並列化が考えられる. 本論文では商インスタンス $\hat{I}^{100}$ の時点での解の経路に沿ってクラスタを分割して各プロセッサに渡した. この方法は結局 $\hat{I}^{k}$ から $\hat{I}^{k+p}$, p はプロセッサ数, の商インスタンスを構成することである. 実行速度の点では有利であるが, 同時に複数のクラスタを生成することや, プロセッサを跨がって都市が所属するクラスタの移動をすることが出来ないことにより, 解の精度を1%程度悪化させた.

今後は解精度 5%未満で, 1000 万都市程度までが現実的な時間内に解を求める ETSP の近似解法を開発する. そのために先に挙げた candidate sets 等の技法と OPT-2/3 や LK 等の改善法を組み合わせ, 並列計算に適したアルゴリズムを検討する. 特に並列計算機上で改善法の効率的な計算が行なえるアルゴリズムの検討を行なう予定である.

参考文献

[1] G. Reinelt: "The Traveling Salesman, Computational Solutions for TSP Applications", Springer-Verlag, Lecture Notes in Computer Science, No.840 (1994).

[2] 新妻, 村田, 山田: "帰納的アルゴリズムに基づく巡回セールスマン問題の解法", 人工知能学会誌, Vol.11, No.1, pp.130-136 (1996).

[3] 山本, 久保: "巡回セールスマン問題への招待", 朝倉書店, シリーズ現代人の数理, No. 12 (1997).