マルチコア計算機における進化計算の並列化

Parallel Implementation of the Evolutionary Computation on a Multi-core Processor

筒井　茂義
Shigeyoshi Tsutsui

阪南大学
Hannan University

This paper describes the parallel evolutionary algorithms in a multi-core computation environment. WE propose two types of parallelization methods; a synchronous parallel thread method and asynchronous parallel thread method. These approaches are evaluated using the edge histogram based sampling algorithm (EHBSA) on TSP benchmarks. Future directions of this research are also discussed.

1. はじめに

近年，マルチコア型のマイクロプロセッサが一般の PC にも使われ，それらを安価に利用できるようになってきた．このタイプの計算機では，並列プログラミングが容易に実現できるため，進化計算の並列化の実現に有効である．本発表では，筆者らが先に発表した順序表現向きの分布推定アルゴリズム（estimation of distributionalgorithm; EDA）の一種であるエッジヒストグラムベースドサンプリングアルゴリズム（Edge Histogram Based Sampling Algorithm; EHBSA）の改良型モデルを用いて，そのマルチコア PC による並列化方式として同期型マルチスレッド方式と非同期型マルチスレッド方式を提案し，TSP を用いて行った評価結果について述べる．

2. 改良型 EHBSA

近年新しい進化計算のモデルとして分布推定アルゴリズム（estimation of distributionalgorithm,）が注目されている [Pelikan 02]．しかし多くの研究は，バイナリ表現や実数値表現問題への適用が多く，スケジューリング問題などの順序表現で表される領域への適用は多くない．筆者は先に順序表現向きの EDA の一手法として，集団のノード間の連結関係（エッジ; edge）の分布を基に新しい個体を生成する手法として EHBSA を提案し，良好な性能を有することを明らかにした [Tsutsui 02, 筒井 03, Tsutsui 04]．ここでは，EHBSA の改良モデルについて簡単に述べる．

2.1 エッジの分布

EHBSA では，集団全体に含まれる個体のエッジの分布を図 1 に示すようにマトリックス形式で表される (Edge Histogram Matrix; EHM) で表現する．ここで，集団サイズ $N = 5$，問題サイズ $L = 5$ としている．EHM の各エントリー e_{ij} の整数部分は，集団におけるエッジ $i \to j$ の総数であり，小数点以下の値（$\epsilon = 0.1$）は，個体性生成時における外乱を与えるバイアスであり，アルゴリズムのパラメータの一つである．ここでは，対称 TSP の場合のようにエッジ $i \to j$ とエッジ $j \to i$ とが同じである場合を示し，EHM は対称マトリックスとなっているが，スケジューリング問題のようにエッジ $i \to j$ とエッジ $j \to i$ とが異なる場合にも同様に EHM は非対称マトリックスとして同様に表現できる（詳しくは文献 [筒井 03] を参照）．

$$
\begin{array}{l}
s^t_1 = (0,\ 1,\ 2,\ 3,\ 4) \\
s^t_2 = (1,\ 3,\ 4,\ 2,\ 0) \\
s^t_3 = (3,\ 4,\ 2,\ 1,\ 0) \\
s^t_4 = (4,\ 0,\ 3,\ 1,\ 2) \\
s^t_5 = (2,\ 1,\ 3,\ 4,\ 0)
\end{array}
\qquad
\begin{pmatrix}
0 & 3.1 & 2.1 & 2.1 & 3.1 \\
3.1 & 0 & 4.1 & 3.1 & 0.1 \\
2.1 & 4.1 & 0 & 1.1 & 3.1 \\
2.1 & 3.1 & 1.1 & 0 & 4.1 \\
3.1 & 0.1 & 3.1 & 4.1 & 0
\end{pmatrix}
$$

(a) $P(t)$　　(b) EHM^t

図 1: EHM の例

2.2 個体生成法

図 2 に EHM から新個体を生成する方法を示す．個体は以下のようにして生成される．

(i) 現在の集団より，テンプレート個体が一つ選ばれる．

(ii) EHM を用いて生成するノード数 l_s を以下に示す式 (1) で決定する．まず，ノード $l_c = L - l_s$ をテンプレート個体からコピーする．コピーの開始位置 c_{top} はランダムに決定する．

(iii) EHM をもとに l_s のノードが式 (2) に基づいてサンプリングされる．

先の EHBSA の研究では，l_s の決定に n カットポイント法を用いていたが [筒井 03]，この方法では，l_s の平均値（$E(l_s)$）が L/n（n=2, 3, ...）に制限されるという問題があり，改良型 EHBSA では，$E(l_s)$ が $L \times \gamma$ ($\gamma \in (0, 1]$) となるパラメータ γ を導入し，上記 (ii) の l_s は次式による．

$$
f(l_s) = \begin{cases}
\frac{1-\gamma}{L\gamma}\left(1 - \frac{l_s}{L}\right)^{\frac{1-2\gamma}{\gamma}} & \text{for } \gamma \in (0,\ 0.5] \\
\frac{\gamma}{L(1-\gamma)}\left(\frac{l_s}{L}\right)^{\frac{2\gamma-1}{1-\gamma}} & \text{for } \gamma \in (0,\ 1]
\end{cases} \tag{1}
$$

(iii) のサンプリングでは，現在のノードが i であるとき，i につづくノード j は，次式の確率 p_{ij}

$$
p_{i,j} = \begin{cases}
\frac{e_{i,j}}{\sum_{s \in F(i)} e_{i,s}} & \text{if } j \in F(i) \\
0 & \text{otherwise}
\end{cases} \tag{2}
$$

連絡先: 筒井茂義，阪南大学，郵便 580-8502 大阪府松原市天美東５－４－３３，Tel: 072-332-1224，Fax: 072-336-2633，E-mail: tsutsui@hannan-u.ac.jp

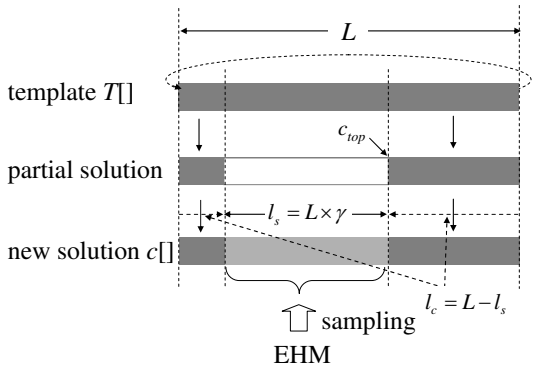

図 2: 個体の生成法

で決定される．ただし，$F(i)$ は，現在のノード i からつぎに選べるノードの集合である．この式に基づくサンプリングの計算量は一般に $O(L^2)$ になり，問題サイズが大きくなるとサンプリング時間が問題となる．このような場合，Candidate リストを使うと計算量を $O(L)$ に近づけることができる．

2.3 世代交代モデル

先の EHBSA では，1 世代に一個体を生成するモデルを用いたが，本研究では，図 3 に示すように，1 世代で N 個体を生成する方式とし，効率的な世代交代モデルに改良している．同図において，新しい個体 I'_i は，集団 $P(t)$ の個体 I_i をテンプレートとして用いる $(i = 1, 2, ..., N)$．各 i のペア（I_i，I'_i）$(i = 1, 2, ..., N)$ を比較し，良い個体を次の世代 $P(t+1)$ のメンバーとする．I_i と I'_i とを比較するこの方式は，Mahfoud の deterministic crowding 法 [Mahfoud 95] のように多様性維持に効果がある．また，この方式では，図 2 で示した γ も重要な設計パラメータとなる．

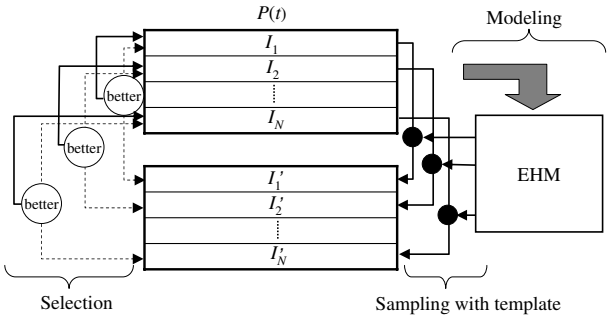

図 3: 世代交代モデル

2.4 性能

ここでは，改良 EHBSA の性能を示すために，巡回セールスマン問題（TSP）を用いて評価する．ここでは，(1) 小規模問題：berlin52，pr76，(2) 中規模問題：att532，rat783，(3) 大規模問題：fl3795，rl5934 の 3 つの規模の問題を取り上げる．実験条件は以下の通りである．(1) 小規模問題に対しては，集団サイズを $2L$，最大解生成数を $2L \times 10000$ とし，ローカルサーチは適用しない．(2) 中規模問題に対しては，集団サイズを $L/15$，最大解生成数を $L \times 1000$ とし，ローカルサーチには 3OPT を適用する．(3) 大規模問題に対しては，集団サイズを 4 とし，最大解生成数を $L/10$ とし，ローカルサーチには TSP における最強のローカルサーチとされている Lin-Kernighan (LK) ヒューリスティックを適用する．

プログラミング言語には Java を用い，マシンには Intel ® Core™ i7 965 プロセッサを用いる．LK コードには Concorde [Applegate 06] を用い，JNI で EHBSA コードと結合した．それぞれの問題に対して γ0.1 から 1.0 まで 0.1 刻みで実験を行った．各実験における試行回数は 20 回とした．各規模の問題に対して，#OPT（20 回の実験中で最適解を発見した回数）および T_{avg}（最適解を発見した実験において，最適解を発見するのに要した時間の平均）による性能評価をそれぞれ，図 4，図 5，および図 6 に示す．

これらの結果から分かるように EHBSA では，γ の適切な設定が重要であることが分かる．また，いずれの規模の問題においても，$0.2 \geq \gamma \geq 0.4$ において#OPT=20 であり，また，T_{avg} も小さい値となっていることが分かる．

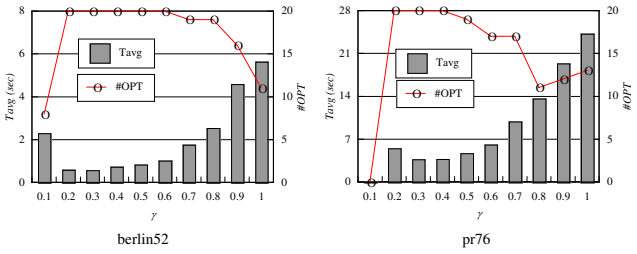

図 4: 小規模問題（berlin52，pr76）の結果（ローカルサーチなし）

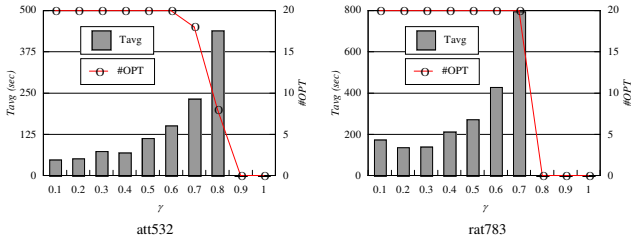

図 5: 中規模問題（att532，rat783）の結果（3OPT ローカルサーチ）

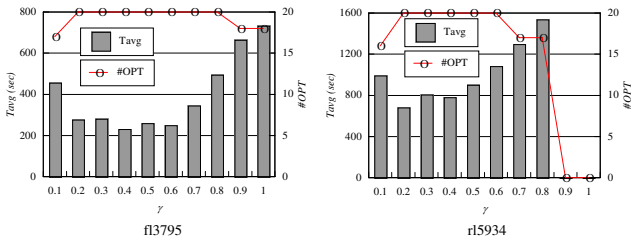

図 6: 大規模問題（fl3795，rl5934）の結果（LK ローカルサーチ）

3. EHBSA の並列化

3.1 マルチスレッドプログラミング

ここでは，2 種類の EHBSA の同期モードと非同期モードの 2 つの並列化スレッドプログラミングを考える．ここで用いるマルチコアプロセッサーは，2.4 で述べた Intel ® Core™ i7 965 プロセッサである．このプロセッサーは 4 つの CPU を内蔵している．BIOS においてハイパースレッディング機能を有効に設定すると OS から見て見かけ上 8 個の CPU が存在するが，ここではその機能は用いないとした．したがって，本研究では，4 つのスレッドを生成して EHBSA の並列実行を評価する．

2 章で述べた EHBSA のプロセスは以下のように以下のステップとなる．

(1) $t \leftarrow 0$. I_i $(i = 1, 2, ..., N)$ からなる初期集団 $P(t)$ をランダムに生成.

(2) $P(t)$ の個体を評価 (ローカルサーチを適用する場合は適用後評価し，I_i も変更) .

(3) $P(t)$ から EHM を作成.

(4) 新個体 I'_i $(i = 1, 2, ..., N)$ を生成.

(5) 各 i $(i = 1, 2, ..., N)$ に対して I_i と I'_i とを比較し，I'_i の方がよければそれを I_i と置き換え，集団 $P(t)$ を更新する.

(6) $t \leftarrow t + 1$.

(7) 更新された $P(t)$ を基に EHM を生成

(8) 終了条件を満たせば終了，満たさなければ (4) へ.

.

同期スレッドモードでは，上記のステップを同期をとって処理をする．したがって 4 つのスレッドは，各ステップの処理において集団サイズ N の処理を全て終えて次の処理に進む．

非同期スレッドモードでは，上記 (3) まで同期モードで動作し，それ以降は独立に未処理の個体を並列実行する．ステップ (7) では，全ての個体が更新されてからとはせず，I_i と I'_i の入れ替えとなったときには，現在の EHM から I_i の属するエッジを取り出し（それらのエッジの数を-1 する），I'_i に属するエッジを加える（それらのエッジの数を+1 する）．この仕組みにより，非同期スレッドモードでは処理の待ち合わせに伴う遅延は発生しない．ただし，処理が非同期に行われるので，各個体に対して各ステップの処理は完全な世代単位には行われなくなる．

3.2 結果

TSP のテスト問題として，oliver30，gr48，berlin52，pr76（以上，ローカルサーチなし），lin318，pr439，att532，rat783（以上，3OPT ローカルサーチを適用），fl3795，rl5934（以上，LK ヒューリスティックを適用）を用いて，2 種類のスレッドモードによる並列化の結果を表 1 に示す．ここで，EHBSA の重要なパラメータである γ には 0.3 を用いた（2.4 参照）．

同図において速度比は，並列処理を行わない場合の T_{avg} をマルチスレッド方式による場合の T_{avg} で割ったものである．なお，マルチスレッドでの実行条件は，2.4 と同じとしている．また，この場合マルチスレッド方式の#OPT はいずれも 20 となった．

結果を先ず同期モードで見ると，ローカルサーチを用いない小規模問題とローカルサーチを用いる中，大規模問題で大きな違いがあることが分かる．中，大規模問題では，速度比が 3.1〜3.7 であるのに対して，ローカルサーチを用いない小規模問題では速度比が 1.5〜2.5 と高速化の度合いが小さくなっている．

一方，非同期モードの結果を見ると，中，大規模問題では同期モードの結果と大きな違いがなく，同程度の高速化の結果が得られている．大きな違いは，ローカルサーチを適用しない場合であるが，この場合にも，同期モードと異なり，3.4〜から 4.4 と高速化の結果が得られている（速度比が 4 を超えるのは，T_avg がある決められた繰り返し時間ではなく，最適解が得られるまでの時間で測定している）.

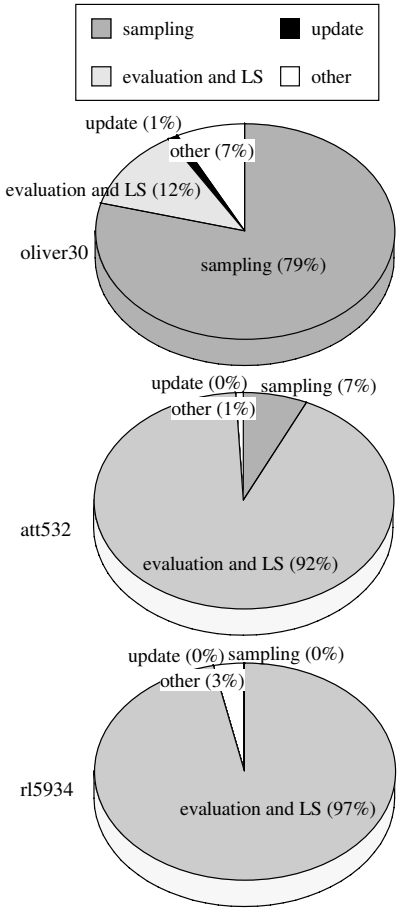

図 7: EHBSA の処理時間の分布（oliver30, att532, rl5934）

以上の結果を解析するために，EHBSA の処理時間の分布を，それぞれ oliver30，att532，rl5934 を使って図 7 に示す．この図から明らかなように，小さな問題である oliver30 では，サンプリングに要する時間が全体の約 70% を占めている．評価の時間はわずか 12% 程度である．サンプリングの時間はサンプリングノード数を決定する式 (1) で確率的に決められるので，同期モードでは長い時間に合せて待ち時間が長くなる．非同期モードではこのような待ち時間はない．これが，小さな問題において同期モードが不利になる最大の要因である．これに対して，中，大規模問題である att532，rl5934 では，同図から分かるようにローカルサーチに要する時間が処理時間の大きな割合を占めている．これらの時間は，各個体ごとに大きな差はないので，同期モードと非同期モードの実行時間に大きな差が現れないものと考えられる．

表 1: マルチスレッドプログラミングによる並列化の結果

分類	インスタンス	ローカルサーチ	シングルスレッド		マルチスレッドモード					
					同期			非同期		
			T_{avg} (sec)	*std*	T_{avg} (sec)	*std*	速度比	T_{avg} (sec)	*std*	速度比
小	oliver30	non	0.08	0.08	0.05	0.08	**1.5**	0.02	0.04	**3.4**
	gr48		0.60	0.70	0.29	0.25	**2.1**	0.16	0.12	**3.8**
	berlin52		0.57	0.62	0.29	0.38	**2.0**	0.13	0.09	**4.3**
	pr76		3.71	2.85	1.46	1.41	**2.5**	0.86	0.67	**4.3**
中	lin318	3OPT	2.57	12.46	0.84	1.26	**3.1**	0.76	0.73	**3.4**
	pr439		3.33	4.05	1.25	1.34	**2.7**	1.17	1.79	**2.8**
	att532		73.84	239.62	21.84	58.40	**3.4**	19.41	52.19	**3.8**
	rat783		139.48	147.63	38.77	53.10	**3.6**	38.37	53.57	**3.6**
大	fl3795	LK	280.88	676.51	76.77	196.99	**3.7**	92.53	306.60	**3.0**
	rl5934		807.27	1482.29	252.44	516.74	**3.2**	209.62	329.65	**3.9**

4. むすび

以上，本稿では EHBSA のマルチコア環境における並列処理方式を提案し，非同期マルチスレッド方式が同期マルチスレッド方式よりも優れていることを示した．大規模な並列処理環境を用いなくても，現在の PC の多くはマルチコアを用いられるようになると思われ，今後さらに多くのコアを有する安価な PC が利用できるようになると考えられる．そして，今回提案したようなマルチスレッド化により，高速な進化計算を手軽に行えるようになると考えられる．

今後の新しい方向として，安価なグラフィックプロセッシングユニット (Graphic Processing Unit; GPU) を用いた並列計算が今後の進化計算の並列処理方式として有望な一手法になると期待できる [NVIDIA 09]．GPU では，1000 個程度以上のスレッドを効率よく実行できる機能を有している．ただし，現時点では進化計算に用いるには高速共有メモリのサイズが小さいなどの問題点があるが，今後有望な並列計算手法になると期待できる．2009 年の GECCO では，GPU への進化計算のワークショップが開かれ，筆者らも発表を行う予定である [Tsutsui 09]．

参考文献

[Applegate 06] Applegate, D., Bixby, R., Chvatal, V., and Cook, W.: ANSI C Code as gzipped tar file, Concorde TSP Solver (2006), http://www.tsp.gatech.edu/concorde.html

[Mahfoud 95] Mahfoud, S.: A Comparison of Parallel and Sequential Niching Methods, in *Procedings of the Six International Conference on Genetic Algorithms*, Morgan Kaufmann (1995)

[NVIDIA 09] NVIDIA, (2009), http://www.nvidia.com/page/home.html

[Pelikan 02] Pelikan, M., Goldberg, D., and Lobo, F.: A survey of optimization by building and using probabilistic models, *Computational Optimization and Applications*, Vol. 21, No. 1, pp. 5–20 (2002)

[Tsutsui 02] Tsutsui, S.: Probabilistic Model-Building Genetic Algorithms in Permutation Representation Domain Using Edge Histogram, *Proceedings of the Seventh International Conference on Parallel Problem Solving from Nature (PPSN VII)*, pp. 224–233 (2002)

[Tsutsui 04] Tsutsui, S. and Miki, M.: Using Edge Histogram Models to Solve Flow Shop Scheduling Problems with Probabilistic Model-Building Genetic Algorithms, *Recent Advances in Simulated Evolution and Learning, Kay Chen Tan, Meng Hiot Lim, Xin Yao, and Lipo Wang Eds, Advances in Natural Computation Series, Chapter 13*, pp. 230–249 (2004)

[Tsutsui 09] Tsutsui, S. and Fujimoto, N.: Solving Quadratic Assignment Problems by Genetic Algorithms with GPU Computation: A Case Study, in *Proceedings of the GECCO 2009 Workshop on Computational Intelligence on Consumer Games and Graphics Hardware CIGPU-2009 (to appear)*, ACM (2009)

[筒井 03] 筒井 茂義：エッジヒストグラムを用いる順序表現向き確率モデル GA の提案, 人工知能学会論文誌, pp. 173–182 (2003)