分解ベース進化的多目的最適化アルゴリズムの並列化に関する検討

Cui, XinTong / サイ, シントウ

(出版者 / Publisher)
法政大学大学院情報科学研究科
(雑誌名 / Journal or Publication Title)
法政大学大学院紀要. 情報科学研究科編
(巻 / Volume)
17
(開始ページ / Start Page)
1
(終了ページ / End Page)
6
(発行年 / Year)
2022-03-24
(URL)
https://doi.org/10.15002/00025261

分解ベース進化的多目的最適化アルゴリズムの並列化に関する検討

Parallelization of decomposition-based evolutionary multi-objective optimization algorithms

サイ　シントウ
Cui XinTong
法政大学情報科学研究科
E-mail: xintong.cui.9m@stu.hosei.ac.jp

Abstract

This paper focus on parallelization of Multi-objective Evolutionary Algorithm Based on Decomposition (MOEA/D), and addresses the decrease of precision and the increase of execution time in Parallel MOEA/D. This paper proposes several grouping methods aim to parallelize MOEA/D while solve the problems such as the sparse boundary and Pareto front dislocation in Parallel MOEA/D, thereby improving the accuracy of the Parallel MOEA/D. The proposed method is also based on the idea of setting overlap zones, but using grouping method that uses inter-individual distances, etc., as grouping methods to reduce additional calculations, instead of setting additional overlap zones. We Implemented and evaluated the grouping methods based on K-means, Mean Shift, Random and Improved Random (Smoother-random). Based on the testing result, the parallel MOEA/D achieved a significant acceleration when the individual size reached a certain size. In addition, these clustering methods also obtain a better accuracy performance in massively parallel environments compared with the grouping method based on neighbor weight vector.

1. まえがき

現実世界の最適化問題は、常に複数の目的が含まれている。このような多目的問題（Multi-objective Optimization Problems , MOP）は様々な分野に存在している、例えば工程、科学実験、商業経営等。多目的問題に対して、伝統的な数学方法では効率的に解決することが難しいため、1985 年を初めに、遺伝的アルゴリズム[1]を多目的問題を解決するように使用された。遺伝的アルゴリズムを用いた多目的最適化問題の解法は、目的関数が明示的に分からない問題に対しても何らかの報酬値が得られれば適用可能、パラメータ空間から目的関数空間へ写像が不線型でも対応可能、最適パレートフロントの形状が非凸の問題にも適用可能などの利点から、急速に研究が発展している。

2007 年、Zhang 等が伝統的な多目的最適問題の分解策から、分解に基づく多目的進化アルゴリズム（Multi-objective Evolutionary Algorithm Based on Decomposition , MOEA/D）を提出した[2]。MOEA/D は、新しい多目的進化アルゴリズムフレームワークであり、それに基づき、例えば T-MOEA/D[3]、MOEA/D-AWA[4]などの、分解に基づくアルゴリズムの多くの改良版が現れた。

Multi-objective Evolutionary Algorithm Based on Decomposition (MOEA/D)は分解に基づくアルゴリズムで、多目的問題を幾つの単一目的問題に分解し、各解候補（個体）の隣接情報と重みベクトルを利用して多目的問題のパレートフロントを求める。一番近い隣接情報を利用するために、並列化の際は目的関数空間における重みベクトルの配置に着目して隣接個体を同じグループに割当るのが一般的であるが、この方法では、隣域情報が使えないため、並列実行して収束した後各グループの edge 部分のソリューション分布はスパースになる傾向があり、最後にフィッテングしたパレートフロントの精度に影響がある。先行研究[5]では Overlap 区域を設置することで各グループの隣域情報をある程度使えるようになり、グループ間の境界領域部分がスパースになる状況を改善した。一方、Overlap 領域の大きさ決めの問題があり、グループの数の増加に従って計算量も増えるため、ここでは計算量を抑えて、精度向上が出来るグルーピング方法を検討する。

2. 個体集団分割法の検討

以下、研究に使った個体集団分割法について説明する。

2.1. 従来法の問題点と対策案

隣接重みベクトルの配置に着目したグルーピング方法は、グループの境界付近の解分布がスパースになる傾向があるため、解決策として、図 1 に示すように、先行研究[5]では Overlap 領域を設置する方法を提案している。その本質は各グループの境界の一定範囲内にある解を隣接グループに使えるように設置することであるが Overlap 領域に所属している解が隣接グループに使うため、複数回追加計算されることも意味している。故に、グループの数、解の数が多い、または Overlap 領域が広い場合、計算量が増加する。それに対応して、もしグルーピングの時、グループの間が自然的にある程度 overlapping していれば、隣接情報を利用せず、グループ内部に集中する傾向があっても、元々グループの間お互いに overlapping しているため、収束の余地があり、スパースの状況が改善出来るだと想定する。この考え方に基づき、以下、クラスタリングなど、個体間の距離に着目した個体集団の分割法などにぁんする検討を行う。

Supervisor: Prof. Yuji Sato

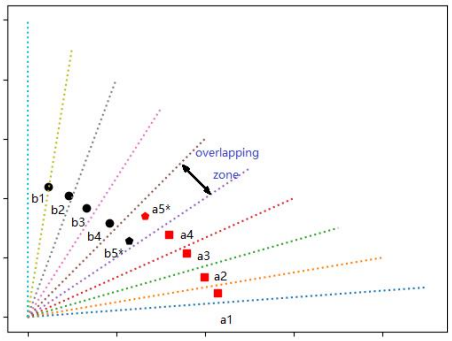

図 1 Overlap 区域を設置する方法.

2.2.クラスタリングを用いた個体集団の分割法

クラスタリングは主に類似する特徴を持つデータを分類するために使われている。MOEA/D の個体集団に適用する場合は解候補の位置情報によって分類することができる。この方法で分類したデータは収束方向に対して、各クラスターの間は常に一部 overlapping しているため、クラスタリング用のアルゴリズムをグルーピングに使って検証する。

代表的なクラスタリング手法である K-Means を用いた分割例を図 2 に示す。ここで、K-Means はグループの数を指定する必要があるため、目安のグループ数が分からない場合に対応するため、図 3 に示す、グループの数を指定せず、bandwidth を基に自適応して分割する Mean Shift に基づくグルーピング方法と比較評価する。

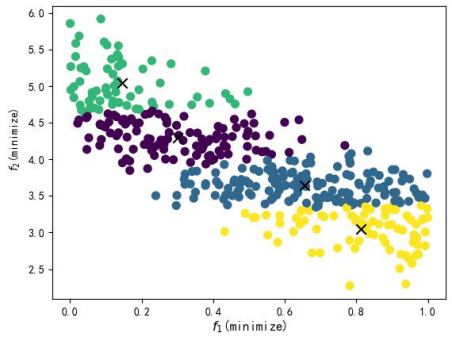

図 2 目的関数 zdt1 の場合の K-Means による分割方法.

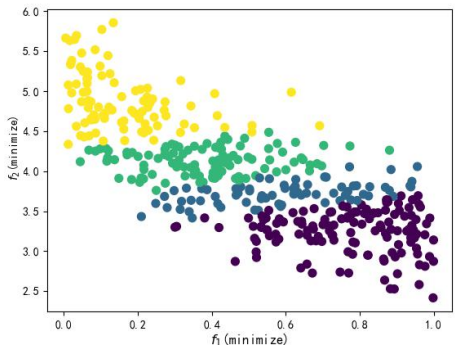

図 3 目的関数 zdt1 の場合の Mean Shift による分割方法.

2.2.1 K-Means による分割

K-means は代表的なクラスタリングアルゴリズムで、類似な特徴を持つ個体を分類するようによく使われる。グループの数を設定することが必要である反面、大規模な個体数でも対応出来て計算速度は速い特徴を持つ。手順を以下に示す。

K-means のステップ
(1) k 個グルッピングセンターを初期化する a=$a_1, a_2, \ldots a_k$。実験は 4 グループに分けるため k=4 にする。
(2) population 各個体x_iが k 個グルッピングセンターへの距離を計算して、一番近いグルッピングセンターのグループに入れる。
(3)各グループa_jに対して、再びグルーピングセンターを計算して移動する$a_j = \frac{1}{|c_i|}\sum_{x \in c_i} x$。
(4)2 と 3 を中止条件まで繰り返す（反復回数等）。

元々はクラスタリング用なアルゴリズムが、MOEA/D のグルーピングに使うとお互いに overlapping しているグルーピングが出来るので、それを MOEA/D に適すように改良し、グルーピング用に実装して比較実験を行う。

2.2.2 Mean Shift による分割

Meanshift アルゴリズムは、K-means と同じクラスタリング用なアルゴリズムであるが、お互い overlapping しているグルーピングが出来る上で、bandwidth を設置すれば自適応的にグループの数を決めることができ、目安のグループの数が分からない場合には向いていると考えられるため、MOEA/D に適すように改良し、実装して比較評価する。他の方法は 4 グループに分けるので、比較実験を行うため、Mean Shift によるグルーピグ方法も 4 グループの場合のデータを採用する。Mean-shift のステップと擬似コード（Algorithm1）[6]を以下に示す。

Mean-shift のステップ
(1)マークされてない個体の中にランダムに一つをセンターとして選ぶ。
(2)センターへの距離<bandwidth の個体を同じグループに入れる、グループ内の個体がこのグループに所属確率+1 とする。
(3)センターを中心にして、センターからグループ内の個体への vector、各 vector を加算することで vector shift を得る。
(4)center=center+shift、即ちセンターを shift に沿って移動する、移動距離は||shift||。
(5)2～4 の過程を収束（shift が非常に小さくなる）まで繰り返して、最後のセンターを記録する。反復過程であった個体を同じグループ c に所属する。
(6)収束したグループ c は他の既存グループc_2のセンターとの距離<threshold 際、c_2と c を一つのグループにする、でないとグループ c を新しいグループにする。
(7)(1)～(5)を全ての個体がマークされるまで繰り返す

(8)各グループが各個体に対する所属確率によって、個体を所属確率一番高いグループに所属する。

Algorithm1 Mean-shift Clustering

Require: h : Bandwidth ; S : Vector set from population
Ensure: modes : The modes of each cluster :

1: for x ∈ S do
2: # Initialization for each vector
3: x_g←x
4: Create a window : Bandwidth : h , Center : x_g
5: # Mean-shift iteration
6: while x_g not converge do
7: $x_g \leftarrow m_h(x_g)$
8: Update window to the new center
9: end while
10: modes append x_g
11: end for
12: Prune modes

大量な population に対応するため、GPU を用いた Faster Mean-shift[7]も実装する。比較実験はそれ程多いな population を使ってないため、普通の Mean-shift で実験を行う。Faster Mean-shift の擬似コード(Algorithm2)を以下に示す。

Algorithm 2 Faster Mean-shift Clustering

Require: h: Bandwidth; S: Vector set;
Ensure: x − modes: A vector-modes list
1: # Seed Selection
2: Evenly random select seed vector set Sseed ∈ S
3: # Parallelization with GPU
4: for x_{seed} ∈ S_{seed} do
5: while x_{seed} not converge do
6: $x_{seed} \leftarrow \mathrm{m}(x_{seed}) \cdot k_h(x_{seed}, x_i)$
7: end while
8: modes append x_{seed}
9: end for
10: Prune modes
11: for x ∈ S do
12: Cluster x by the distance to modes
13: end for

2.3.Random と Smoother Random による分割

クラスタリングを用いた手法と比較するために、図 4 に示すように、境界だけではなく、グループ全体的に overlapping するように、ランダムによるグルーピング方法も実装して検証する。実験によって、解の分布が比較的平均分布するようになっている場合が得たパレートフロントが比較的に良い結果が出たので、それを検証するため、ランダム分割に基づき、制限条件を追加することで、Smoother なランダム分割方法(S-random)を実装する、Smoother random を用いた分割例は図 5 に示す。

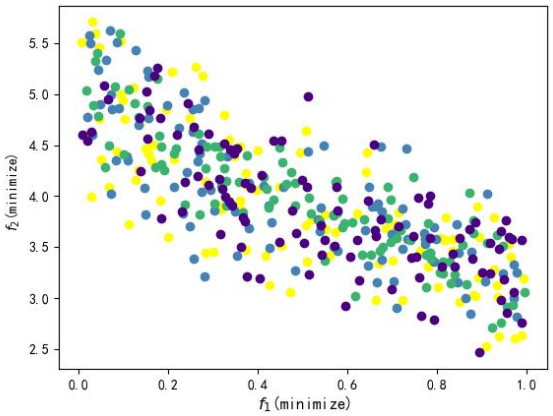

図 4 目的関数 zdt1 の場合の random による分割方法.

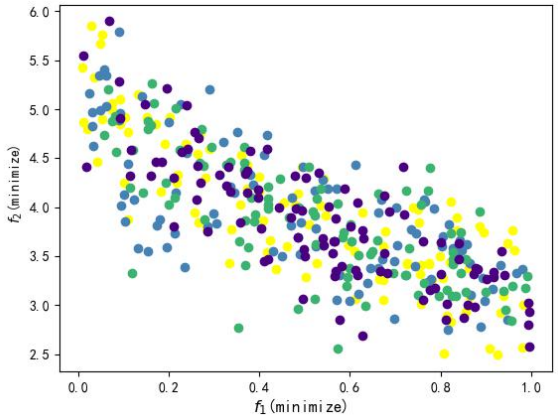

図 5 目的関数 zdt1 の場合の Smoother-random による分割方法.

ランダムによる分割では population をランダムに 4 グループにする。一番近い隣接情報を利用することを諦め、overlap 領域を population 空間全域まで拡大するやり方である。一番近い隣接情報を利用出来ないとは、精度をある程度低下して行くことを意味しているが、低下している部分をグループの間でお互いに補足する、並列実行した後、最後に Last generation というもう一回の進化を行うことで、各グループの良い部分を採用し、劣解を淘汰することで、パレートフロントを得る。Smoother random は、random 方法に基づき、制限条件を追加することで、各グループが population 空間内比較的均一的に分布するように改良した方法である。例えば 400 個体 4 グループの場合、個体を下から上に並べて、1～20, 21~40, ..., 381~400 の間に、4 グループは必ず各 5 個の個体を持つ形で、ランダム方法を均一的にする、それで比較実験を行う。

3.実装法の検討

C++で実装する並列化 MOEA/D アルゴリズムは multi-thread を使って複数のコアを使用して並列化実行するが、本研究では Python を使うので、CPython パーサーの環境で実行する場合、Global Interpreter Lock(GIL)があるため、multi-thread は multi-core の機能を発揮出来ず、特に CPU-bound 計算がある場合、python での multi-thread はシリアルより遅くなり、逆効果になる。GIL を回避して並列実行をするため、multi-thread の代わりに multiprocessing を使う。アルゴリズムは [3] と [7] を参考して、multiprocessing に適すように改良する。

Multi-processing を使う場合、各 sub-process が自分の GIL を持っているため、お互いに独立して実行する。一方、sub-process は比較的に独立ため、データの交流は難しいので、高速化するため、MOEA/D の進化過程を multi-processing に適すように改良することが必要となる。各 process の進化に必要なデータを並列化の最初に送り、なるべく process 間のデータ交流を減らす。最終の結果は pipe または Queue で回収する。改良した並列化 MOEA/D の流れは図 6 に示す。

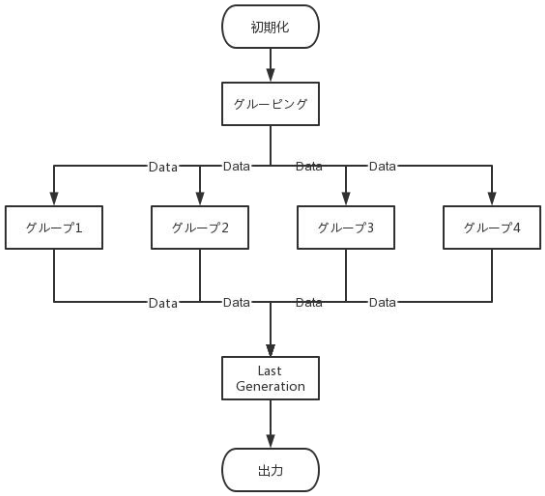

図 6 並列化 MOEA/D の流れ.

4.実験

4.1.実験環境

実行環境として、GPU：NVIDIA GeForce GTX1070、CPU：Intel(R)Core(TM)i7-8750H 2.20GHz 2.21GHz を使用する。アルゴリズムは python で実装する。並列化は multiprocessing を使い、CPython パーサーの環境で実行する。population の数は 400、目的関数は zdt1 と zdt3 で、シリアルオリジナル MOEA/D(original)、隣接重みベクトルによる multiprocessing MOEA/D（MP-basic）、K-means 方法 multiprocessing MOEA/D(MP-Kmeans)、Mean-shift 方法 multiprocessing MOEA/D(MP-Meanshift)、Random 方法 multiprocessing MOEA/D(MP-random) と Smoother-random 方法 multiprocessing MOEA/D(MP-Srandom)で、比較実験を行う。データは 11 回実行して、平均値または中央値を使う。比較実験のため、自適応でグループ数を決める Mean-shift 方法の bandwidth を調整して、4 グループの場合のデータを採用する。実験で使用する population はそれ程多いではないため、CPU での Mean-shift 方法を使う。

4.2.評価基準

実験は Hypervolume によって評価する。Hypervolume は Zitzler 等が提出して、解のセットが目的空間に形成した超立方体の体積（volume of the hypercube）を表示する。Hypervolume はパレートフロントと一致している、即ちもし解セット S は解セット S'より良いなら、解セット S の Hypervolume は S'より大きいとなる。

4.3.実験データ

実験は上記方法を異なる世帯に設置して実行することで、様々なデータを記録して比較する。

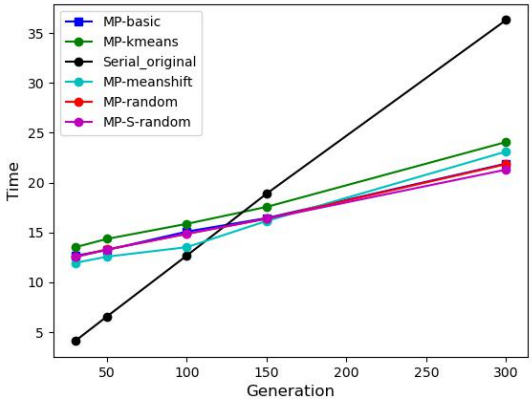

図 7 Time-Gen with zdt1.

Table1 zdt1 関数に対する各分割法の実行時間.

分割法 世帯数	30	50	100	150	300
original	4.11	6.58	12.69	18.90	36.32
MP-basic	12.66	13.25	15.09	16.45	21.91
MP-kmeans	13.52	14.36	15.87	17.57	24.08
MP-meanshift	11.95	12.58	13.53	16.16	23.12
MP-random	12.56	13.31	14.86	16.39	21.85
MP-Srandom	12.54	13.27	14.88	16.38	21.31

Table2 zdt3 関数に対する各分割法の実行時間.

分割法 世帯数	30	50	100	150	300
original	4.91	8.20	16.16	23.96	46.33
MP-basic	12.82	13.60	15.74	17.99	24.03
MP-kmeans	13.48	14.24	16.94	19.66	23.19
MP-meanshift	12.77	13.23	14.57	15.97	24.69
MP-random	12.77	13.57	15.66	17.69	24.30
MP-Srandom	12.73	13.49	15.56	17.54	23.49

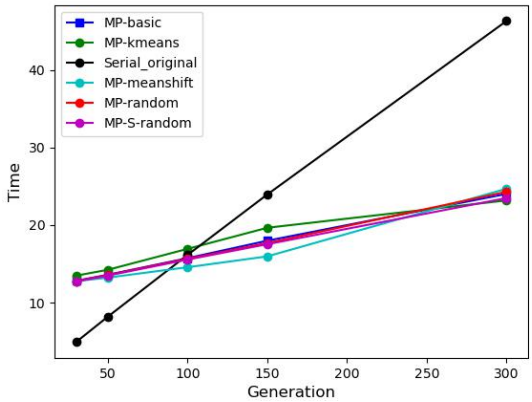

図 8 Time-Gen with zdt3.

図 7, 8 と Table1,2 の結果によって、明らかに、計算量が少ない場合、シリアルのオリジナル MOEA/D の計算が速いが、計算量の増加と共に、multiprocessing 並列化の効果が現れた。その原因は、python の multiprocessing は、各 process を順番に起動して計算を始める、実験環境で、4 つのプロセスを全部起動するには、約十秒程かかるので、計算量が少ない場合は主にプロセスを起動することで時間がかかる。計算量が増えたら、起動時間のパーセンテージが減少して、multiprocessing の加速効果が徐々に現れる。Zdt3 目的関数の計算量は zdt1 より多い為、加速の効果も比較的に良い、300 世帯の場合最大は約 50%加速となっている。世帯数を更に増やすと、加速効果も良くなることを予想出来る。

Table3 zdt1 関数に対する各分割法の HyperVolume.

分割法 世帯数	30	50	100	150	300
original	1.0917	1.0937	1.0938	1.0939	1.0991
MP-basic	1.0570	1.0802	1.0971	1.0962	1.0947
MP-kmeans	1.0918	1.0991	1.0997	1.1003	1.1015
MP-meanshift	1.0948	1.0976	1.0991	1.1012	1.1019
MP-random	1.0974	1.0999	1.1003	1.1006	1.1013
MP-Srandom	1.1000	1.1009	1.1013	1.1015	1.1019

Table4 zdt3 関数に対する各分割法の HyperVolume.

分割法 世帯数	30	50	100	150	300
original	1.6027	1.6176	1.6202	1.6220	1.6189
MP-basic	1.4078	1.4176	1.4593	1.4608	1.5555
MP-kmeans	1.6139	1.6263	1.6269	1.6308	1.6315
MP-meanshift	1.6108	1.6211	1.6289	1.6311	1.6336
MP-random	1.6262	1.6276	1.6271	1.6279	1.6290
MP-Srandom	1.6285	1.6313	1.6310	1.6315	1.6320

図 9,10 と Table3,4 によって、世帯数が少ない場合、基本的な隣接重みベクトルによる分割方法は精度に影響があること、そして検証する方法はそれに対して精度向上の効果があることが分かった。特に非連続的な目的関数 zdt3 に対して、隣接重みベクトルによる分割はパレートフロントの dislocation 又は劣解を起こす可能性が高い為、結果は比較的悪いが、k-means, meanshift, random, Srandom の方法ではそれより精度向上の効果があることが分かった。

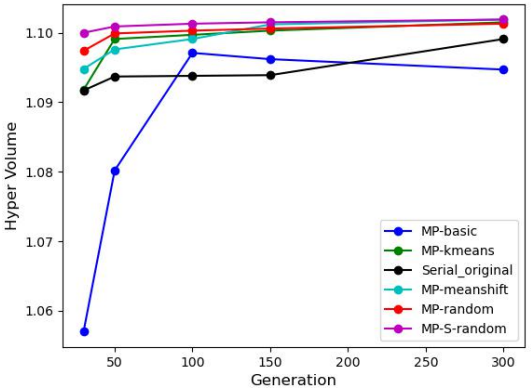

図 9 Hypervolume-Gen with zdt1.

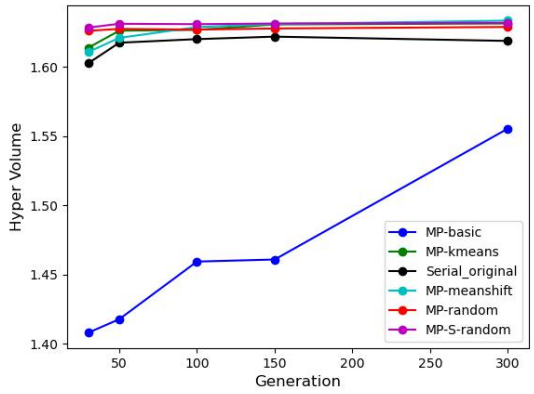

図 10 Hypervolume-Gen with zdt3.

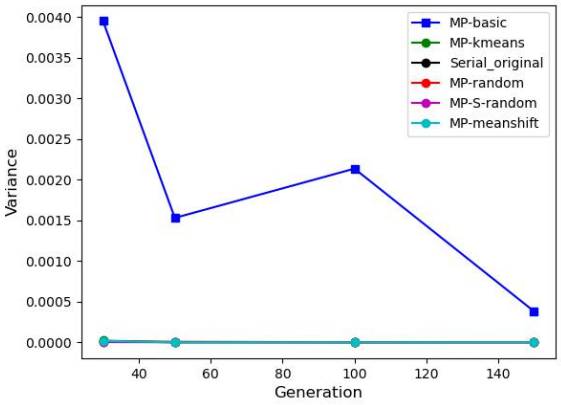

図 11 Variance-Gen with zdt1.

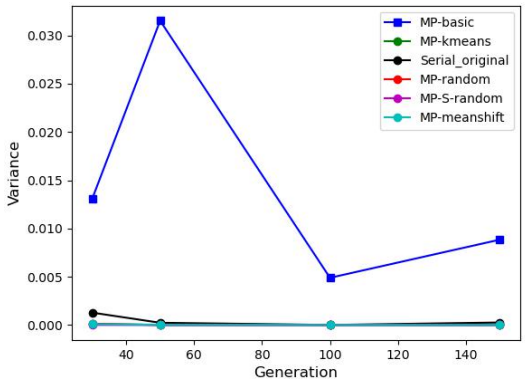

図 12 Variance-Gen with zdt3.

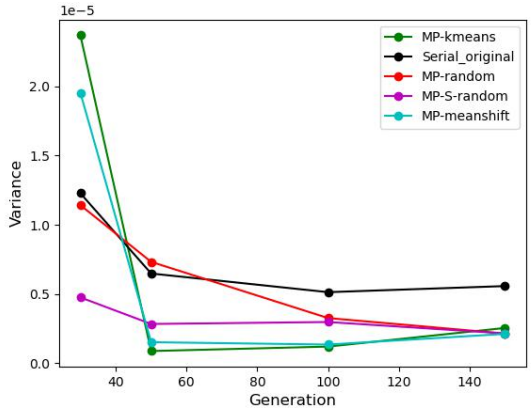

図 13 Variance-Gen with zdt1(without MP-basic).

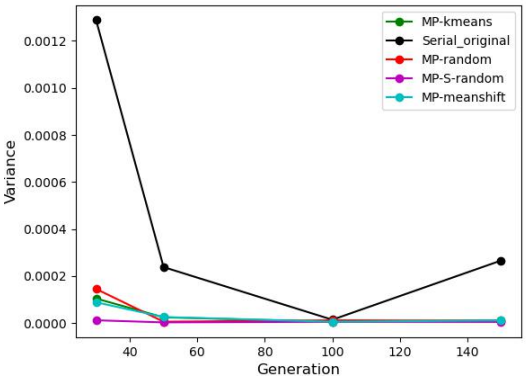

図 14 Variance-Gen with zdt3(without MP-basic)

図 11、と図 12 に示すように、Variance から見ると、提案する方法は隣接重みベクトルでの分割方法より安定し、良いパレートフロントを得やすいことが分かった。図 13、14 から、隣接重みベクトル分割方法を除いて見ると、random と Srandom 方法は比較的に安定。その原因は population の数が十分の場合、ランダム分割することは一番近い隣接情報を利用出来ずことでの精度低下は、四つのグループでお互い補足することで精度向上して、更に一つのグループが dislocation 又は悪い解を得たとしても、他のグループによって悪い解を淘汰することで、最終的に Fault tolerance が良くなる。

4.4 結果分析

実験によって、提案する方法の精度、安定性は隣接重みベクトルによる分割法より良い、特に世帯数が少ない、又は非連続目的関数 zdt3 の場合に対してはより効果的となっている。その中に、Smoother random の精度向上効果と安定性は比較的に良い、random 方法は Smoother random と似っているが、安定性は少し下げている。K-means と Mean shift の精度向上効果と安定性の傾向は基本的に同じレベルで、世帯数が低い場合の安定性はランダムより少し低いが、50 世帯以上の場合は好調となる。低い世帯の安定性向上と高速化は今後の課題にする。

5. むすび

本稿では、計算量を抑えて、解探索制度の低下なしに、MOEA/D を並列高速化するための個体集団の分割法に関する検討と評価を行なった。具体的には、k-means, meanshift, random, Srandom の有効性に関する検討を行い、python 環境でそれらの方法を用いて並列化 MOEA/D アルゴリズムを実装した。多目的最適化問題の代表的なベンチマーク問題である Zdt1,Zdt3 を用いて、並列化の加速効果、そして分割方法の効果を比較した。その結果 population の数が充分の場合、k-means、mean-shift、random そして Smoother random 方法の精度向上効果を検証して、提案する方法は有効的であることを示した。

文　献

[1] J Holland. Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence. Quarterly Review of Biology, 1975, 6(2):126 – 137

[2] Qingfu Zhang, Hui Li. MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition. IEEE Transactions on Evolutionary Computation, 2007, 11(6):712 – 731

[3] Hai-Lin Liu, Fang-qing Gu, Yiu-ming Cheung. T-MOEA/D: MOEA/D with objective transform in multi-objective problems. In: Proc of Information Science and Management Engineering (ISME), 2010 International Conference of, volume 2. IEEE, 2010, 282 – 285

[4] Yutao Qi, Xiaoliang Ma, Fang Liu, et al. MOEA/D with adaptive weight adjustment. Evolutionary Computation, 2014, 22(2):231 – 264

[5] Sato M , Sato Y , Midtlyng M , et al. Inconstant Update of Reference Point Value for Parallel and Distributed MOEA/D[C]// 2021 IEEE Congress on Evolutionary Computation (CEC). IEEE, 2021.

[6] You, L. , Jiang, H. , Hu, J. , Chang, C. , Chen, L. , & Cui, X. , et al. (2021). Gpu-accelerated faster mean shift with euclidean distance metrics.

[7] Sato Y , Sato M , Midtlyng M , et al. Parallel and Distributed MOEA/D with Exclusively Evaluated Mating and Migration[C]// 2020 IEEE Congress on Evolutionary Computation (CEC). IEEE, 2020.