Journal of Institute of Control, Robotics and Systems (2014) 20(8):854-867
http://dx.doi.org/10.5302/J.ICROS.2014.13.0019 ISSN:1976-5622 eISSN:2233-4335

베타 확률분포를 이용한 입자 떼 최적화 알고리즘의 성능 비교

On the Comparison of Particle Swarm Optimization Algorithm Performance using Beta Probability Distribution

이 병 석*, 이 준 화, 허 문 범
(ByungSeok Lee[1,*], Joon Hwa Lee[2], and Moon-Beom Heo[3])
[1,2]School of Electrical and Computer Engineering, University of Seoul
[1,3]Satellite Navigation Team, Korea Aerospace Research Institute

Abstract: This paper deals with the performance comparison of a PSO algorithm inspired in the process of simulating the behavior pattern of the organisms. The PSO algorithm finds the optimal solution (fitness value) of the objective function based on a stochastic process. Generally, the stochastic process, a random function, is used with the expression related to the velocity included in the PSO algorithm. In this case, the random function of the normal distribution (Gaussian) or uniform distribution are mainly used as the random function in a PSO algorithm. However, in this paper, because the probability distribution which is various with 2 shape parameters can be expressed, the performance comparison of a PSO algorithm using the beta probability distribution function, that is a random function which has a high degree of freedom, is introduced. For performance comparison, 3 functions (Rastrigin, Rosenbrock, Schwefel) were selected among the benchmark Set. And the convergence property was compared and analyzed using PSO-FIW to find the optimal solution.

Keywords: PSO (Particle Swarm Optimization), PSO-FIW (PSO with Fixed Inertia Weight), beta probability distribution, benchmark set

I. 서론

1995년 말 Kennedy와 Eberhart에 의해 입자 떼 최적화(PSO: Particle Swarm Optimization)가 발표된 이래 과학 및 공학 분야 등 수많은 분야에서 PSO 알고리즘이 활용되고 있다[13-17]. PSO는 인공생명(AL: Artificial Life)과 떼 이론(Swarm theory)을 바탕으로 진화계산(EC: Evolutionary Computation)과 관련된 알고리즘이다. 특히, 유전자 알고리즘(GA: Genetic Algorithms)과 진화 프로그래밍(EP: Evolutionary Programming) 둘 다와 밀접한 연관이 있다[1,2].

AL이나 떼 이론이 근간이 된 PSO는 인간의 사회적 행동(social behavior)을 모델링하기 위해 유기체 즉, 동물들의 무리(herd), 새무리(bird flock), 물고기떼(fish school) 등이 포식자를 피하거나 먹이를 찾거나 짝짓기를 하는 등의 행동 패턴을 연구하고 컴퓨터 시뮬레이션을 만드는 과정에서 그들만의 특징들을 발견하면서 영감을 얻어 만들어진 최적화 알고리즘이다[2].

GA와 같은 EC 기술들과 많은 유사한 부분들을 공유하는 PSO는 떼 지능(SI: Swarm Intelligence)에 대한 5가지 기본 원칙과 인공지능이나 GA에서 질량이나 볼륨이 없는 agent 혹은 individual로 표현되는 부분을 알고리즘을 이루는 위치와 속도 식으로 인해 point보다 입자(particle)라는 용어를 사용하게 되었다[2,3]. 이 때, particle은 해결하고자 하는 문제의 잠재적인 solution으로 대표된다. 또한 GA와 같은 population 기반의 알고리즘과 같이 PSO에서는 Swarm이라는 용어가 그 의미를 대신하게 된다.

본 논문에서는 PSO 알고리즘을 이루는 위치와 속도에 관한 항목 중 유일한 연산식인 속도에 대한 연산에 포함되어 있는 무작위성을 달리하여 실험한 결과를 소개하고 있다. 실험에 사용된 알고리즘은 초기 PSO 알고리즘의 수렴속도(convergence speed)나 지역적 탐색(local search) 혹은 전역적 탐색(global search)간의 균형, 속도의 제한성 문제 등을 해결하기 위해 개선된 PSO 알고리즘을 대상으로 하고 있다. 이 때, PSO 알고리즘 구현에 있어 일반적으로 정규분포(normal distribution)나 균등분포(uniform distribution)를 많이 사용하는데 이러한 분포의 무작위성에 대한 차이를 인식하여 준난수(quasi-random or pseudo random)인 저불일치 수열(low-discrepancy sequences)인 Sobol, Faure, Halton, Van der Corput 등을 이용한 연구가 진행되었으며[18,19], [20]에서는 Beta와 Gamma 분포에 대한 연구도 진행하여 Beta 분포가 비교적 좋은 성능을 가짐을 보였다.

하지만 [20]의 경우는 초기값을 설정하는데 있어 여러 가지 분포를 사용하였고, Beta 분포의 경우 형상모수(shape parameter)에 대한 언급이 직접적으로 없으며, 그 가지 수 또한 한 가지로 제한적임에 반해 본 논문은 Beta 분포의

* Corresponding Author
Manuscript received December 25, 2013 / revised April 13, 2014 / accepted June 2, 2014
이병석: 서울시립대학교 전자전기컴퓨터공학부 및 한국항공우주연구원 위성항법팀(byungseok@uos.ac.kr)
이준화: 서울시립대학교 전자전기컴퓨터공학부(joonhwa@uos.ac.kr)
허문범: 한국항공우주연구원 위성항법팀(hmb@kari.re.kr)
※ 본 논문은 기초기술연구회 "재난예방 및 국민안전제고를 위한 위성기반 위치추적기술 연구"과제로 수행되었음.

형상모수를 총 8가지로 다양하게 임의 선정하였고, 초기값들을 선정할 때 분포를 달리하는 것뿐만 아니라 알고리즘을 매번 수행 시 업데이트 되는 PSO 알고리즘의 속도 연산식에 사용되는 랜덤함수 또한 Beta나 Normal, Uniform 분포를 사용하였다. 또한, 이들에 대한 총 알고리즘을 100회(1회 당 300번의 반복) 실행하여 평균과 표준편차를 제시하여 비교하였다.

본 논문의 II 장은 PSO 알고리즘과 그 구현에 대해 소개하고 있다. III 장은 초기 PSO 알고리즘에서 수렴 속도를 가속화하기 위해 관성하중(Inertia Weight) 계수를 갖는 PSO 알고리즘(이하, PSO-IW)과 PSO-IW 보다 더 나은 수렴율(convergence rate)를 갖는 제한 계수(Constriction Coefficient) χ를 갖는 PSO 알고리즘(이하, PSO-CC)에 대해 다룬다. IV 장은 본 논문의 핵심이 되는 베타 분포에 대한 소개와 특징들을 알아보며, 시뮬레이션에 사용하게 될 벤치마크 함수 중 선정된 3가지 함수를 소개하고 있다. V 장에서 시뮬레이션 결과를 보여주고 있으며, VI 장은 본 논문의 결론을 맺고 있다. 부록에서는 무작위성과 차원, 함수에 따른 PSO 알고리즘 수행 결과를 표와 그림으로 나타내고 있다.

II. PSO 알고리즘

1. PSO 알고리즘 연산식

유기체들의 사회적인 행동 근저에 있는 규칙에 영감을 얻어 발전된 PSO는 간결한 구현과 동시 처리를 위한 병렬화가 쉽고, 전역해를 구하는데 효율적인 것으로 알려져 있다[2,4]. PSO는 임의의 문제를 해결할 수 있는 잠재적인 해(solution)인 입자(particle)의 위치(position)와 속도(velocity)에 대한 식으로 표현된다.

식 (1)과 (2)는 PSO의 위치와 속도에 관한 식을 각각 나타내고 있다.

$$X_i(t+1) = X_i(t) + V_i(t+1) \tag{1}$$

$$v_{ij}(t+1) = v_{ij}(t) + c_1 r_{1j}(t)[y_{ij}(t) - x_{ij}(t)] + c_2 r_{2j}(t)[\hat{y_j}(t) - x_{ij}(t)] \tag{2}$$

식 (1)은 단순한 더하기 연산으로 이루어진 위치 업데이트를 나타내며, PSO에서 더하기 이외의 연산이 포함된 연산식은 식 (2)의 속도 업데이트에 관한 식이다. 식 (2)에서 아래 첨자 ij는 j 차원에 있는 입자 i를 의미한다. 즉, $v_{ij}(t)$는 t시각에 j 차원에 있는 입자 i의 속도를 의미한다. c_1와 c_2는 각각 가속 상수(acceleration constant)로 국부적인 해와 전역적인 해를 찾는데 있어 가중값(weighting value)을 나타낸다.

식 (2)에서 볼 수 있듯이 속도에 관한 업데이트 식은 우변의 3가지 항으로 이루어져 있다. 첫 번째 항은 이전 시각에서의 속도를 의미하며, 두 번째 항은 일반적으로 Cognitive Component 혹은 Nostalgia로 불리 우는 정해진 크기의 Swarm 안에서 입자 자신의 개인적인 최적위치해(y_{ij})와 해당 시각에서 같은 차원의 위치해(x_{ij}) 간의 차이에 비례하는 값으로 표현되며, 세 번째 항은 Social Component로 불리우는 것으로 전역적인 최적위치해($\hat{y_j}$)와 해당 시각에서 같은 차원의 위치해(x_{ij}) 간의 차이에 비례하는 값을 나타낸다. 이 때, 두 번째, 세 번째 항에는 랜덤 값인 r_{1j}와 r_{2j}가 곱해져서 탐색공간 안에서 입자들이 무작위성을 가지며 퍼져나가게 된다[1-4].

```
1: data_Initialization();
2: pos_vel_Initialization();
3: FitnessValue_Lbest_gbest_Initialization();
4:
5: While(FV_gbest<Stop_Condition)
6:   pos_vel_update();
7:   FitnessValue_calculate();
8:   if FV_pos < FV_lbest
9:     FV_lbest <- FV_pos;  Store_lbest;
10:  end
11:  if FV_lbest < FV_gbest
12:    FV_gbest <- FV_lbest;  Store_gbest;
13:  end
```

그림 1. PSO 알고리즘 의사코드.

Fig. 1. Pseudo code of PSO algorithm.

2. PSO 알고리즘 의사코드

PSO 알고리즘을 구현할 때 필요한 연산은 식 (1)과 (2) 이외에 알고리즘을 실행하는 목적 즉 해결하고자 하는 문제의 해를 찾을 때 그 기준이 되는 값이 요구된다. 일반적으로 적합도 값(fitness value)이나 목적함수(objective function) 등으로 표현하게 된다. 만일 적합도 값이나 목적함수가 최소값을 찾는 문제라면, 앞서 1절에서 언급한 PSO의 속도 연산식에서 두 번째와 세 번째 항에 포함되어 있는 best position 값 중 y_{ij}는 각각 일정한 Swarm 크기 안의 입자 중에서 가장 작은 값을 나타내는 최적위치벡터(personal best position, y_{ij})를 의미한다. 한편, $\hat{y_j}$은 알고리즘을 수행하면서 입자들이 탐색공간 안에서 퍼져 나갈 때, personal best position의 해와 비교해서 더 작은 값들이 나오면 이는 전역적인 최적위치벡터(global best position, $\hat{y_j}$)로 대치되는 과정을 거치면서 적합도 값이나 목적함수를 만족하면 알고리즘 수행을 중단하는 과정을 거치게 된다. 이 때, personal best position은 pbest 혹은 국부적인 최적위치벡터를 의미해서 lbest로 표현하기도 하며, global best position은 gbest로 나타낸다. 이렇게 최적해를 찾아가는 과정을 의사코드(pseudo code)로 나타내면 그림 1과 같다.

III. 계수 및 제한성이 개선된 PSO 알고리즘

1. 관성하중계수를 갖는 PSO 알고리즘

고전적인 PSO는 2장 1절에서 언급하였듯이 식 (2)의 우변에서 두 번째 항과 세 번째 항을 통해 입자들이 주어진 문제의 최적해를 찾아 각각 exploitation과 exploration을 하게 된다. 이러한 절차는 일찍이 Reynolds가 새떼들(bird flocks)의 시뮬레이션을 구현하면서 고려한 3 가지 기본 규칙(Collision Avoidance, Velocity Matching, Flock Centering) [7]과 같이 자신(입자) 주위(neighbor)의 해와 일정 크기의 그룹 혹은 swarm 이상의 차원에서 얻은 최적해를 적절히 이용하게 된다.

이렇게 탐색 공간상에서 최적해를 찾기 위해 다른 영역을 탐색하는 exploration과 어느 영역에서 집중적으로 해를 찾는 exploitation을 조절하는 방법으로 식 (2)의 속도식에서 우변의 첫 번째 항인 이전 시각의 속도 $v_{ij}(t)$에 inertia weight라는 계수를 이용하는 방법의 PSO 알고리즘인 PSO-IW가 Shi와 Eberhart에 의해 고안되었다[5].

$$v_{ij}(t+1) = wv_{ij}(t) + c_1r_{1j}(t)[y_{ij}(t) - x_{ij}(t)] + c_2r_{2j}(t)[\hat{y_j}(t) - x_{ij}(t)] \quad (3)$$

식 (3)에서 w의 값이 크면 보다 넓은 영역에서 해를 찾게 되고, 다양성(diversity)이 증가하는 반면, w 값이 작다면 입자가 국부적인 부분에서 해를 찾게 된다. 주어진 문제의 해를 찾는 과정에서 w값은 다양하게 변할 수 있는데 아래와 같은 종류를 예로 들 수 있다[4].

- 랜덤 조절 방법 · 선형 감소 방법 · 비선형 감소 방법
- 퍼지 적응 방법 · 관성 체증(遞增) 방법

1.1 랜덤 조절 방법

랜덤 조절 방법(random adjustments)은 w를 알고리즘에서 속도 업데이트 시 매번 랜덤하게 다른 관성값을 선택하는 것으로 구현된다. 식 (4)는 w가 0.72의 평균값을 갖고 분산이 작은 Normal(Gaussian) distribution의 경우의 예이다.

$$w \sim N(0.72, \sigma^2) \quad (4)$$

또한, w는 식 (5)와 같이 식 (2)의 cognitive와 social component 부분의 계수들인 c_1, r_1, c_2, r_2를 이용하여 만들 수 있다[4,21]. 이 때, c_1, c_2는 랜덤값이 아닌 상수이다.

$$w = (c_1r_1 + c_2r_2) \quad (5)$$

1.2 선형 감소 방법

선형 감소 방법(linear decreasing)은 w의 값을 초기의 큰 값에서 작은 값으로 감소시키는 방법이다. 보통 초기값은 0.9로 정하며, 0.4의 값까지 선형적으로 감소시킨다. 식 (6)은 선형감소 방법에 대한 w에 관한 식이다[11,12].

$$w(t) = (w(0) - w(n_t))\frac{n_t - t}{n_t} + w(n_t) \quad (6)$$

1.3 비선형 감소 방법

비선형 감소 방법(nonlinear decreasing)은 앞서 소개한 선형 감소 방법과 비슷하게 w값을 큰 값에서 작은 값으로 비선형적으로 감소시키는 방법으로 식 (7)-(9)와 같은 방법으로 구현될 수 있다[4,22].

$$w(t+1) = \frac{(w(t) - 0.4)(n_t - t)}{n_t + 0.4} \quad (7)$$

$$w(t+1) = w(1) + (w(n_t) - w(0))\frac{e^{m_i(t)} - 1}{e^{m_i(t)} + 1} \quad (8)$$

$$w(t+1) = \alpha w(t') \quad (9)$$

식 (7)에서 w값은 초기에 1.4에서 마지막 시각에는 0.35까지 감소하는 방법을 사용하며, 식 (8)에서 $m_i(t)$는 상대 개선값(relative improvement)으로 일정 시각 t에서의 $gbest$ 값 혹은 $lbest$값과 같은 시각에서 위치에 대한 목적함수 값의 차와 합의 비율로 정해지게 된다.

1.4 퍼지 적응 방법

퍼지 적응 방법(fuzzy adaptive)은 관성하중계수가 퍼지 집합과 퍼지 규칙에 의해 동적으로 조절되는 방법이다. 이 방법은 2개의 입력과 1개의 출력을 갖는다. 입력으로는 임의의 시각에서 전역최적해의 값과 관성하중계수의 값을 가지고 있으며, 출력으로는 변경된 관성하중계수를 갖는다. 그리고 LOW, MEDIUM, HIGH라는 3가지 퍼지 집합을 갖는다[23].

1.5 관성 체증 방법

관성 체증 방법(increasing inertia)은 선형 감소 방법과는 반대로 w값이 0.4에서 0.9까지 선형적으로 증가하게 되는 방법이다[24].

2. 제한 계수를 갖는 PSO 알고리즘

χ라는 제한 계수를 식 (2)의 속도 관련 식에 포함하여 식 (10)과 같이 스케일링 역할을 하게하는 방법이다.

$$v_{ij}(t+1) = \chi[v_{ij}(t) + \phi_1r_{1j}(t)(y_{ij}(t) - x_{ij}(t)) + \phi_2r_{2j}(t)(t)(\hat{y_j}(t) - x_{ij}(t))] \quad (10)$$

식 (11)에서 제한 계수 값 χ는 아래와 같은 식 (11)와 같이 계산되어진다[6].

$$\chi = \frac{2\kappa}{|2 - \phi - \sqrt{\phi(\phi-4)}|} \quad (11)$$

식 (11)에서 κ는 $[0, 1]$의 범위에 있으며, ϕ는 4 이상의 값을 갖는다. 그리고 ϕ는 식 (11)의 ϕ_1과 ϕ_2의 합으로 표현된다.

IV. 베타 확률 분포와 벤치마크 함수

1. 베타 확률 분포

일반적으로 필터 및 제어기 설계 시 외란의 확률분포를 가정하거나 랜덤한 값이 요구될 때 또는 어떤 제품의 파손율 조건 등을 모델링 할 때 정규(가우시안)분포 혹은 균등분포, 지수분포와 같은 확률분포가 많이 사용된다. 정규(가우시안)분포나 지수분포 등은 확률변수(RV: Random Variable)가 균등분포와 달리 특정한 범위를 가지지 않는다. 또한, 확률밀도함수를 임의의 비대칭 모양으로 자유롭게 변경하는데 제한이 있는 확률밀도함수가 존재한다.

하지만, 베타분포의 경우는 식 (12)와 같은 확률밀도함수인 PDF에서 확인할 수 있듯이 a와 b라는 양수의 형상 모수(shape parameter)를 가지고 있어 이 둘의 값을 변화시키면서 확률밀도함수를 다양한 모양으로 변형시킬 수 있다. 이 때, 베타함수(beta function) $B(\bullet)$는 감마함수로도 나타낼 수 있다.

$$f(x|a,b) = \frac{1}{B(a,b)}\frac{(x-l)^{a-1}(u-x)^{b-1}}{(u-l)^{a+b-1}}, \; l \le x \le u$$
$$= 0, \; otherwise \quad (12)$$

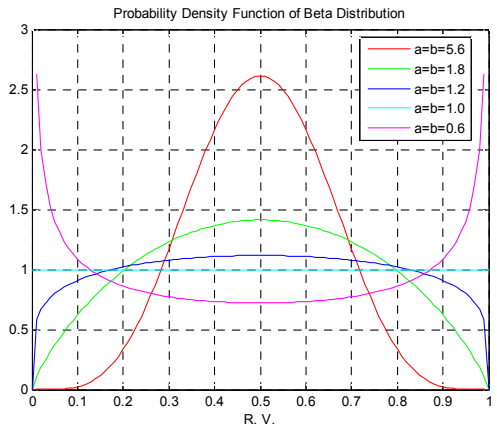


그림 2. 베타분포의 확률밀도함수 #1.

Fig. 2. Probability density function of beta distribution #1.

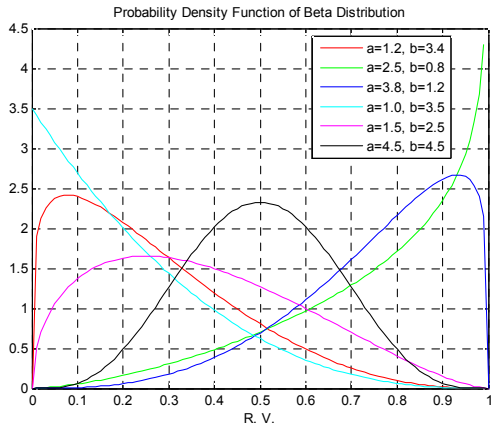


그림 3. 베타분포의 확률밀도함수 #2.

Fig. 3. Probability density function of beta distribution #2.

식 (12)에서 l과 u는 확률변수 x의 범위가 된다. 이 범위의 하한과 상한을 0과 1사이로 한정할 때, 식 (13)과 같은 간결한 형태가 된다.

$$f(x|a,b) = \frac{1}{B(a,b)} x^{a-1}(1-x)^{b-1}, \; l \le x \le u \\ = 0 \qquad\qquad , otherwise \qquad (13)$$

표 1. 선정된 벤치마크 함수.

Table 1. Selected benchmark functions.

Func.	Name & Formula	Min. Value
f_1	*Rastrigin function* $\sum_{i=1}^{N}\left(x^2 - 10\cos(2\pi x_i) - 10\right) + 40$	0
f_2	*Rosenbrock function* $\sum_{i=1}^{N-1}\left(100\left(x_{i+1} - x_i^2\right)^2 + (x_i - 1)^2\right)$	0
f_3	*Schwefel function* $418.9829N + \sum_{i=1}^{N}\left(x_i \sin\sqrt{\lvert x_i \rvert}\right)$	0

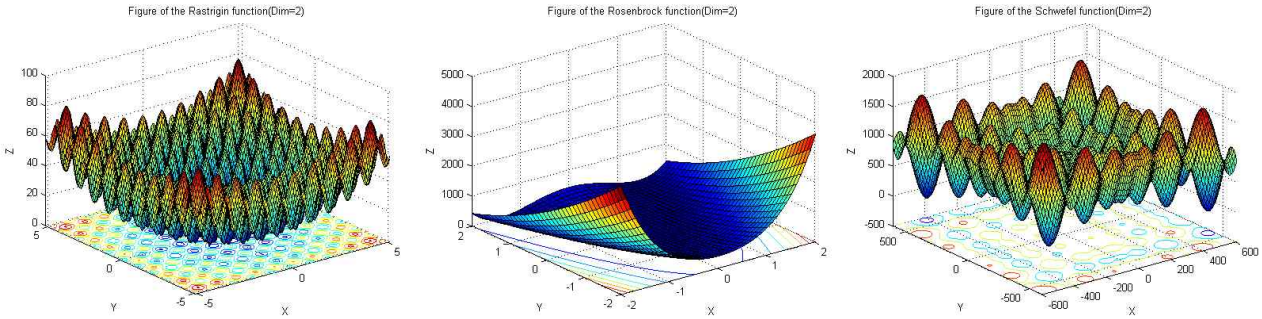
베타분포가 형상모수에 따라 그 확률밀도함수가 어떻게 변하는지 살펴보면 그림 2-3과 같다. 그림 2는 2개의 형상모수가 서로 같은 값을 가질 때 나타낸 경우이며, 그림 3은 형상모수가 서로 다른 값을 가질 때의 결과이다.

2. 벤치마크 함수

알고리즘의 성능을 평가하기 위해 자주 사용하는 방법 중 벤치마크 함수를 이용하는 방법이 있다. 기준이 되는 몇 가지 함수를 통해 최적해를 찾는 결과를 비교하는 방법으로 본 논문에서는 3가지 함수를 선정했다. 선정 기준은 아래와 같은 요인을 고려하여 벤치마크 함수를 정하였다.

- 차원(dimensionality)
- 비선형성(non-linearity)
- 다양한 형태(multi-modal)

위와 같은 요인에서 다수의 국부적인 최적해가 여러 가지 존재하고, 비선형적이며, 여러 차원을 가질 수 있는 함수를 선택하였다[8-10]. 표 1은 선정된 벤치마크 함수에 대한 정보를 나타내고 있으며, 그림 4는 이 함수들의 대략적인 모양을 볼 수 있도록 domain과 간격을 조절하여 MATLAB의 내장 함수인 surf 등을 이용하여 표현한 그림이다. 표 1에서 Min. Value 열에 해당하는 숫자는 각 함수의 최소값을 의미한다.



(a) Rastrigin function $-5.12 \le x_i \le 5.12,\ \triangle_{x_i} = 0.08$.

(b) Rosenbrock function $-2 \le x_i \le 2,\ \triangle_{x_i} = 0.1$.

(c) Schwefel function $-600 \le x_i \le 600,\ \triangle_{x_i} = 12$.

그림 4. 벤치마크 함수들의 개형.

Fig. 4. Shape of the benchmark functions.

V. 시뮬레이션 결과

시뮬레이션은 그림 1의 의사코드와 같이 기본 알고리즘을 바탕으로 MATLAB의 m-file로 구현하여 실행하였다. 시뮬레이션에서 알고리즘 수행 결과의 대조군이 되는 3가지 벤치마크 함수는 수식 및 그 형태가 서로 다르므로 particle의 탐색 범위를 제한하였고, PSO 알고리즘을 구현함에 있어 차원과 particle들의 묶음인 Swarm의 크기를 일정하게 정의하였다. 시뮬레이션에서 사용한 PSO 알고리즘은 관성하중값을 적용한 PSO-IW를 사용하였다. 이 때, 관성하중계수 w는 0.7로 고정(fixed)하였으며(이하, PSO-FIW), 가속상수 c_1과 c_2는 2.25값으로 동일하게 설정하여 랜덤값에 대한 성능 비교를 확인할 수 있도록 하였다.

알고리즘은 1회 수행 시 300회의 반복을 통해 최적해를 찾도록 하였으며 이러한 알고리즘을 총 100회씩 수행하였다[부록 참조]. PSO-FIW 알고리즘을 수행할 때 초기값은 식 (14)~(15)와 같이 설정하였다. 이러한 이유는 벤치마크 함수 $f_1 \sim f_3$의 최소값이 존재하는 곳이 "0" 이므로 초기값을 이 값 부근에 존재하지 않게 하기 위해 식 (14)~(15)와 같이 임의로 상당 부분 벗어난 영역에 편향되게 설정하였다. 또한, 탐색 범위와 관련하여 알고리즘 수행 시 particle의 위치와 속도가 탐색 범위를 초과하면 위치와 속도값을 최대 탐색 범위 값의 중간값으로 재설정하여 관심영역 밖에서 알고리즘이 수행되지 않도록 하였다. 표 2는 벤치마크 함수에 적용한 PSO 알고리즘의 주요 설정값을 보여주고 있다.

$$X_0 = (ss_u - ss_l)R + ss_l \tag{14}$$

$$v_0 = 2ss_uR - ss_u \tag{15}$$

식 (14)~(15)에서 X_0, v_o는 초기 위치와 초기 속도, R은 테스트할 확률분포를 나타내는 랜덤함수로 임의의 particle을 생성하게 하며, ss_u와 ss_l은 각각 탐색할 영역인 domain 즉, search space의 상한(upper bound)과 하한(lower bound)를 의미한다.

표 2. PSO 알고리즘 적용 시 주요 설정값.

Table 2. Principle setting value for PSO algorithm test using benchmark functions.

Func.	Search Space	Dim.	Swarm Size	Iteration # per Algorithm
f_1	[-20, 20]	10	15	300
		30		
f_2	[-20, 20]	10	15	300
		30		
f_3	[-300, 300]	10	15	300
		30		

[부록]의 표 4, 5, 6에서 보는 바와 같이 PSO 알고리즘에서 사용하는 랜덤값을 생성하기 위해 Gussian, Uniform, Beta 확률분포를 사용하였는데, 각각 MATLAB의 randn, rand, random 함수를 이용하였다. 베타 확률분포는 형상모수인 a와 b의 값을 (12, 5), (5, 12), (3.8, 1.2), (1.2, 3.8), (0.6, 0.6), (1.2, 1.2), (1.8, 1.8), (5.6, 5.6)으로 임의 설정하여 다른 확률분포와 비교하였다.

[부록]의 표 4는 Gaussian과 Uniform 분포를 사용한 경우이다. 함수 f_1, f_2, f_3의 최적해를 10차원과 30차원의 경우에 PSO-FIW 알고리즘을 100회(각 회당 300번의 반복) 실행하여 구한 평균과 표준편차를 보여주고 있다. 한편, 표 5, 6은 형상모수의 설정에 따른 결과를 보다 폭넓게 살펴보기 위해 8가지 다른 형상모수를 갖는 Beta 분포를 사용하여 표 4에서 다룬 동일한 방법으로 동일한 알고리즘의 결과를 나타내고 있다. 이 때, 표 5는 형상모수 a와 b를 다르게 설정하여 그림 3에서 대체로 확인할 수 있듯이 편향된 pdf를 갖도록 하였으며, 표 6은 형상모수 a와 b를 동일하게 설정하여 그림 2와 같은 대칭적인 pdf를 이용할 수 있도록 하였다.

결과적으로 표 4, 5, 6에서 확인할 수 있듯이 Gaussian 분포를 갖는 PSO-FIW 알고리즘의 경우는 Uniform과 Beta 분포를 갖는 알고리즘보다 최적해(함수의 최소값)에 대한 값이 모두 크게 나타나 최적해를 찾는 성능이 가장 낮음을 보였다. 반면, Uniform과 Beta 분포를 PSO-FIW 알고리즘에 적용하였을 때, 함수 f_1, f_2가 10차원, 30차원의 경우 모두 형상모수 (3.8, 1.2), (0.6, 0.6)을 제외한 6가지 Beta분포를 갖는 알고리즘 결과값이 Unifrom 분포를 사용한 알고리즘의 결과보다 작아 최적해를 찾는 성능이 더 우수했다.

또한, 함수 f_3은 10차원일 때, 형상모수 (12, 5), (3.8, 1.2), (0.6, 0.6), (1.2, 1.2), (5.6, 5.6)을 제외한 나머지 3가지의 Beta 분포를 갖는 경우만 Uniform 분포의 결과보다 우수했고, 함수 f_3이 30차원일 때, 형상모수 (3.8, 1.2), (0.6, 0.6)을 제외한 나머지 6가지 Beta 분포가 Uniform 분포의 경우보다 최적해를 찾는 성능이 우수한 결과를 나타내고 있다.

VI. 결론

본 논문은 오늘날 진화계산(EC) 알고리즘 중 다양한 분야에서 활용되고 있는 PSO 알고리즘의 랜덤값에 대한 특성을 살펴보았다. 고전적인 PSO 알고리즘이나 몬테카를로 시뮬레이션(MSC: Monte Carlo Simulation), 시뮬레이티드 어닐링(SA: Simulated Annealing) 등 에서 자주 사용하는 Gaussian과 Uniform의 확률 분포와 달리 2개의 형상모수로 다양한 확률분포를 형성할 수 있는 베타 확률 분포를 적용하여 그 성능이 달라지며 어느 경우에는 Gaussian과 Uniform 분포를 이용하여 사용하는 PSO 알고리즘보다 우수한 성능을 보일 수 있음을 비교하였다. 특히 본 논문에서 테스트한 8가지 형상모수 중 (5, 12), (1.2, 3.8), (1.8, 1.8)의 Beta 확률분포는 Gaussian과 Uniform 분포의 경우보다 함수 f_1, f_2, f_3에 대하여 10차원, 30차원 모두 더 작은 최적해를 찾았으며, 이 세 가지 형상모수 중에서 각 함수들에 대한

표 3. 베타확률분포의 성능비교와 정규 및 균등분포와의 감소율 비교.

Table 3. Performance Comparison according to the Beta probability distribution and the Decrement compared with the Gaussian and Uniform probability distribution against the Optimum Beta probability distribution.

Func.	Dim.	Beta Distribution	Compared with Gaussian	Compared with Uniform
f_1	10	② > ③ > ①	96.15229%	70.71235%
	30	① > ③ > ②	87.60040%	95.94979%
f_2	10	① > ② > ③	99.99982%	99.76433%
	30	② > ① > ③	99.99440%	99.93953%
f_3	10	② > ① > ③	25.40563%	18.37894%
	30	① > ② > ③	31.01510%	28.22743%

① : (5, 12), ② : (1.2, 3.8), ③ : (1.8, 1.8)

최적값을 찾는 경우는 함수와 차원성에 따라 표 3과 같은 순서로 성능이 우수하게 나타났다. 표 3은 3 가지 형상모수를 갖는 Beta 분포끼리의 성능비교를 포함하며, 이들 중 제일 우수한 성능과 Uniform 분포, Gaussian 분포의 경우 최소값들과 각각 비교했을 때, 최소값이 몇 % 감소 됐는지 나타내고 있다.

한편, 형상모수 (5.6, 5.6)을 갖는 Beta 분포의 경우 10차원의 f_3의 경우에 Uniform 분포일 때보다 최적해가 커서 성능 좋은 3가지 형상모수에 포함되지 않았지만 10차원 f_1의 경우 3가지 형상모수의 경우보다 더 작은 최적해를 찾았다.

표 3의 3가지 베타확률분포의 성능비교에서 ①과 ②의 경우 즉, 2 가지 형상모수 (5, 12), (1.2, 3.8)의 Beta 확률분포가 3가지 함수 모든 경우에서 최적해를 갖는 확률분포에 속한다. 이러한 이유는 particle들이 무작위성을 갖고 분포될 때 서론에서 언급한 선행되어진 연구[18-20]에서 저불일치성(low-dicrepancy)과 더불어 추가적으로 탐색공간의 조밀성과 관련 있는 것으로 판단된다. 위 2 가지 형상모수는 그림 3에서 보이는 확률밀도함수 그래프를 통해 알 수 있듯이 "0"부근에서 높은 분포를 갖게 된다.

즉, 이러한 확률분포를 갖는 의미는 다시 말해, particle이 분포될 때, 다른 형상모수의 Beta 확률분포나 Gaussian, Uniform 분포보다 이전 particle이 분포된 것 중 최적해를 갖는 particle에서 다시 새로운 particle을 분포할 때, 그 최적해 주위를 우선적으로 더 밀도 있게 탐색하게 되어 나타나는 결과로 판단된다.

향후, 베타 확률분포를 이용함에 있어 보다 다양한 벤치마크 함수에 대한 분석을 통해 그 유용성을 검증할 수 있을 것으로 판단되며, 해결하고자 하는 문제의 특성에 따라 확률분포를 변경할 수 있는 가능성을 제시할 수 있을 것으로 예상된다. 이는 Beta 확률분포의 형상모수를 변경해가며 주어진 문제(함수 등)에서 최적의 해를 찾는 방법을 통해 최적의 형상모수 a, b 값을 찾을 수도 있게 된다.

더불어, 수학적인 엄밀함을 통해 확률분포에 따른 무작위성을 이용하는 알고리즘으로 최적해를 찾을 때 확률분포에 따라 최적해의 차이가 생김을 증명할 필요가 있다. 또한, 개선 사항으로 본 논문에서 사용한 MATLAB의 Beta 확률분포 랜덤함수의 경우 그 실행시간이 Gaussian이나 Uniform 분포의 확률분포 랜덤함수보다 오래 걸리는 단점이 있으므로, Beta 확률분포 랜덤함수를 이용하여 랜덤값을 생성하는 과정의 연산 시간 단축이 필요하다.

부록

표 4, 5, 6에서 Min. Value의 Mean과 Standard Deviation 값은 100회 알고리즘(1회마다 300번 반복수행)을 수행한 결과의 평균값과 표준편차값을 의미한다.

표 4. 확률분포에 따른 PSO-FIW 알고리즘 성능 비교 #1.

Table 4. PSO-FIW algorithms performance comparison according to the probability distribution #1.

Probability Distribution	Func.	Dim.	Min. Value Mean	Min. Value Standard Deviation
Gaussian	f_1	10	5.0333×10^{2}	50.644824
		30	2.3654×10^{3}	91.870942
	f_2	10	2.9439×10^{6}	5.4940×10^{5}
		30	2.1801×10^{7}	1.8611×10^{6}
	f_3	10	3.0631×10^{3}	89.118356
		30	1.0659×10^{4}	1.8318×10^{2}
Uniform	f_1	10	66.125830	17.707349
		30	7.2416×10^{2}	1.1509×10^{2}
	f_2	10	2.2033×10^{3}	1.8100×10^{3}
		30	2.0195×10^{6}	7.9274×10^{5}
	f_3	10	2.7994×10^{3}	1.5132×10^{2}
		30	1.0245×10^{4}	2.5382×10^{2}

표 5. 확률분포에 따른 PSO-FIW 알고리즘 성능 비교 #2.

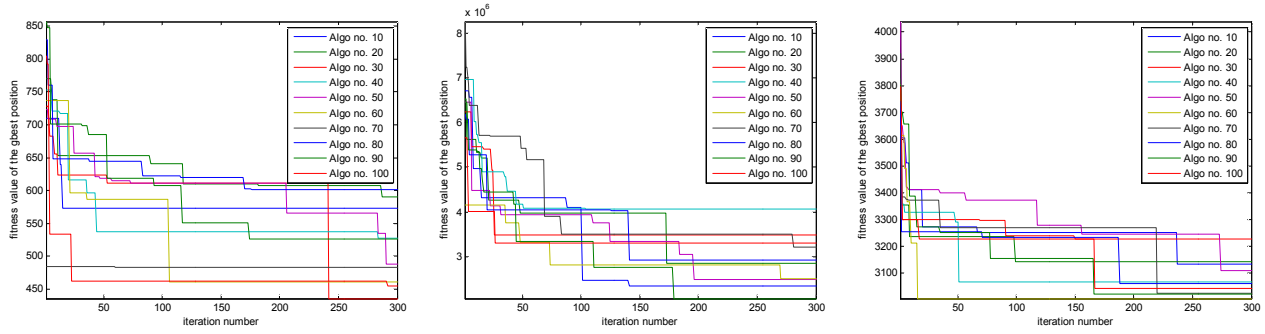
Table 5. PSO-FIW algorithms performance comparison according to the probability distribution #2.

Probability Distribution	Func.	Dim.	Min. Value Mean	Min. Value Standard Deviation
Beta (a=12, b=5)	f_1	10	29.715401	10.869493
		30	4.5545×10^{2}	64.6702
	f_2	10	1.0128×10^{2}	89.549280
		30	4.6301×10^{5}	2.3632×10^{5}
	f_3	10	3.0460×10^{3}	1.3212×10^{2}
		30	1.0132×10^{4}	3.1959×10^{2}
Beta (a=5, b=12)	f_1	10	32.893204	16.586002
		30	2.9330×10^{2}	73.0891
	f_2	10	5.192593	10.589161
		30	3.1674×10^{3}	1.1387×10^{4}
	f_3	10	2.3862×10^{3}	1.8576×10^{2}
		30	7.3531×10^{3}	4.8087×10^{2}
Beta (a=3.8, b=1.2)	f_1	10	2.4578×10^{2}	44.326746
		30	1.5886×10^{3}	1.5059×10^{2}
	f_2	10	3.6960×10^{5}	2.4459×10^{5}
		30	1.1993×10^{7}	1.8800×10^{6}
	f_3	10	3.0343×10^{3}	1.1574×10^{2}
		30	1.0328×10^{4}	1.9162×10^{2}
Beta (a=1.2, b=3.8)	f_1	10	19.3667	10.8431
		30	3.1262×10^{2}	1.0683×10^{2}
	f_2	10	6.1506	12.9232
		30	1.2212×10^{3}	5.2590×10^{3}
	f_3	10	2.2849×10^{3}	1.2025×10^{2}
		30	7.6821×10^{3}	3.3030×10^{2}

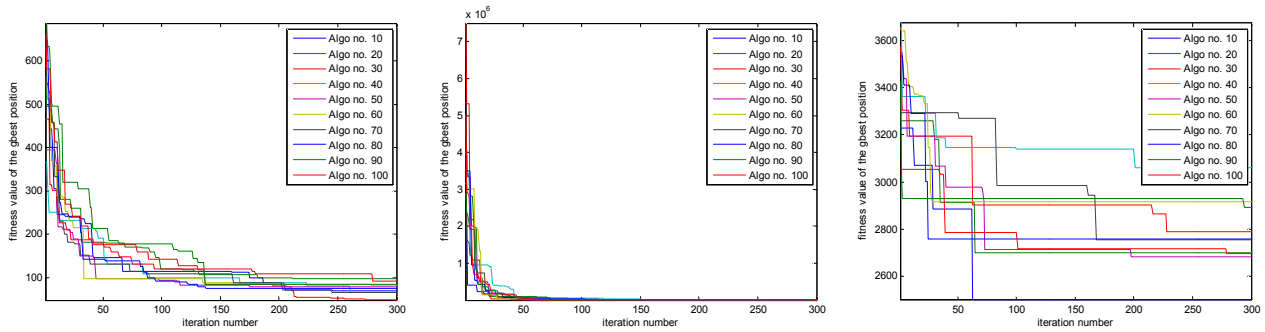
표 6. 확률분포에 따른 PSO-FIW 알고리즘 성능 비교 #3.

Table 6. PSO-FIW algorithms performance comparison according to the probability distribution #3.

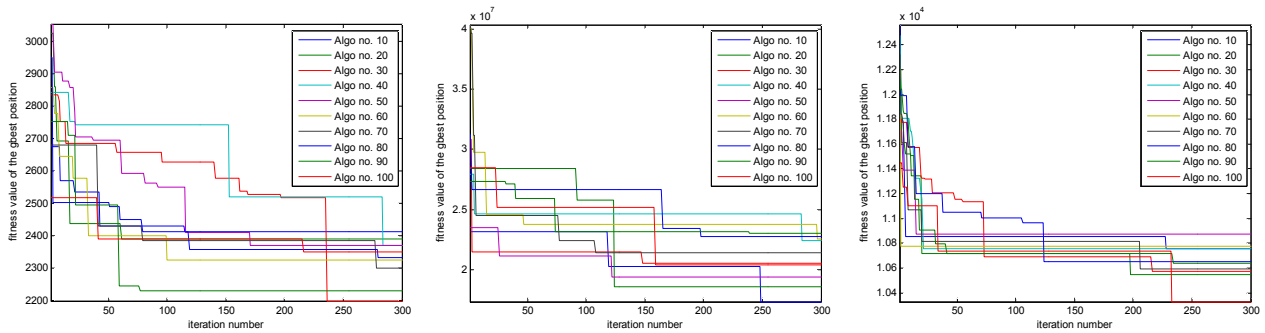
Probability Distribution	Func.	Dim.	Min. Value Mean	Min. Value Standard Deviation
Beta (a=0.6, b=0.6)	f_1	10	1.1402×10^{2}	23.540691
		30	1.0651×10^{3}	1.5718×10^{2}
	f_2	10	2.5815×10^{4}	2.2380×10^{4}
		30	5.3142×10^{6}	1.7010×10^{6}
	f_3	10	2.8785×10^{3}	1.3568×10^{2}
		30	1.0328×10^{4}	2.3293×10^{2}
Beta (a=1.2, b=1.2)	f_1	10	51.859377	14.196011
		30	5.6665×10^{2}	83.036785
	f_2	10	5.3636×10^{2}	4.6817×10^{2}
		30	1.0604×10^{6}	4.9549×10^{5}
	f_3	10	2.8201×10^{3}	1.4014×10^{2}
		30	1.0121×10^{4}	2.6994×10^{2}
Beta (a=1.8, b=1.8)	f_1	10	19.456155	8.028454
		30	2.9542×10^{2}	51.132755
	f_2	10	40.828905	52.402846
		30	6.4843×10^{4}	4.0339×10^{4}
	f_3	10	2.6350×10^{3}	1.7318×10^{2}
		30	9.4903×10^{3}	4.4772×10^{2}
Beta (a=5.6, b=5.6)	f_1	10	16.894390	7.840627
		30	1.3728×10^{2}	35.993654
	f_2	10	8.279345	16.755669
		30	3.9771×10^{2}	4.3023×10^{2}
	f_3	10	2.8792×10^{3}	1.9027×10^{2}
		30	9.4907×10^{3}	4.5021×10^{2}



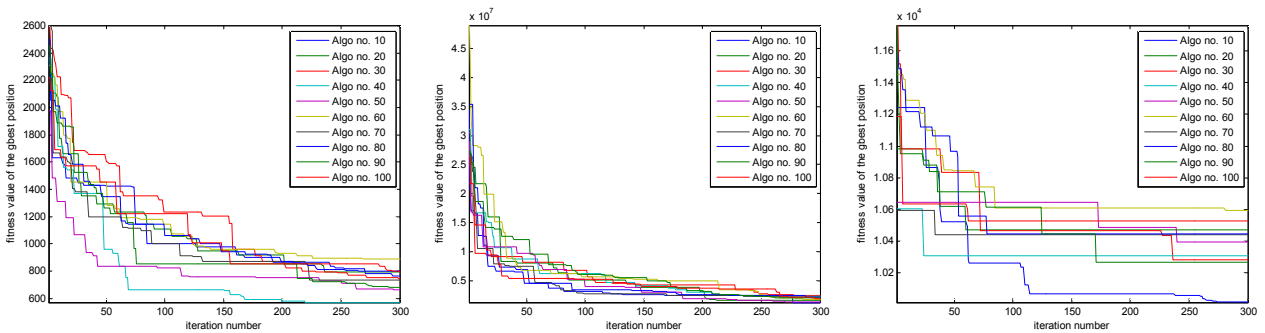
f_1, Gaussian, Dimension = 10 *f_2, Gaussian, Dimension = 10* *f_3, Gaussian, Dimension = 10*



f_1, Uniform, Dimension = 10 *f_2, Uniform, Dimension = 10* *f_3, Uniform, Dimension = 10*



f_1, Gaussian, Dimension = 30 *f_2, Gaussian, Dimension = 30* *f_3, Gaussian, Dimension = 30*

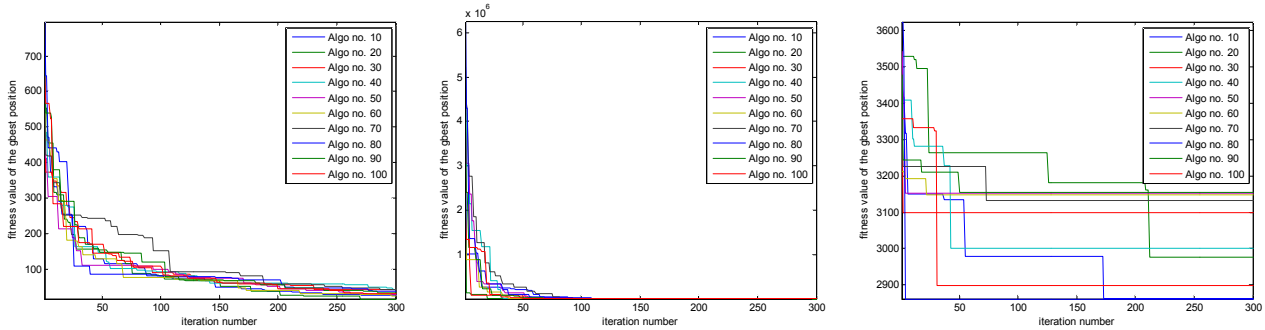


f_1, Uniform, Dimension = 30 *f_2, Uniform, Dimension = 30* *f_3, Uniform, Dimension = 30*

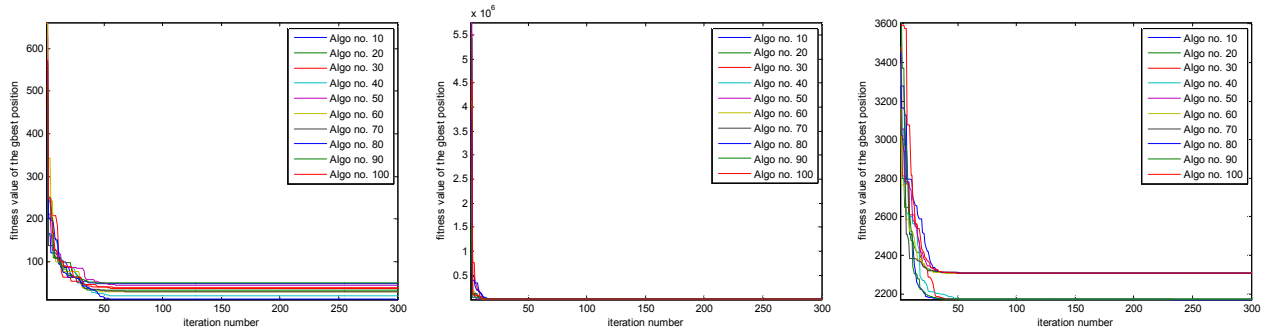
그림 5. 벤치마크 함수에 대한 확률분포에 따른 PSO-FIW 알고리즘 성능 비교 #1(차원 = 10, 30).

Fig. 5. PSO-FIW algorithm performance comparison according to the probability distributions on the benchmark functions #1(Dim. = 10, 30).

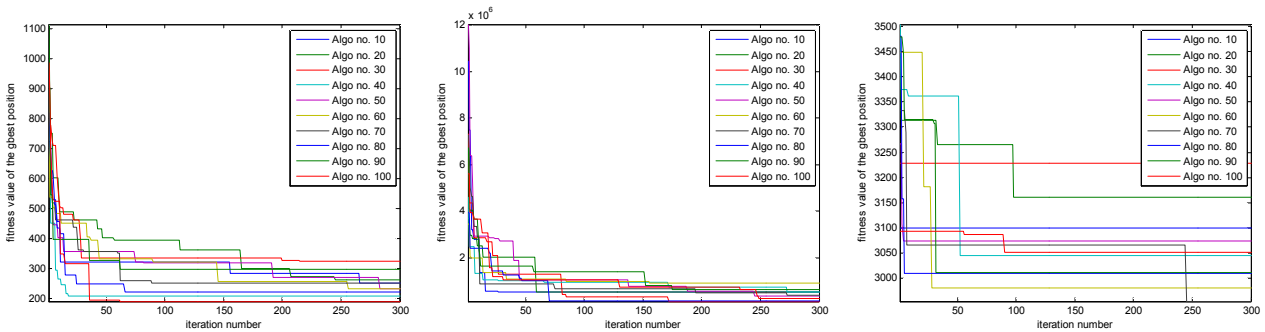
그림 5는 Gaussian 분포와 Uniform 분포에 대하여 함수 f_1, f_2, f_3의 최소값을 10차원과 30차원에 대하여 PSO-FIW 알고리즘을 적용하여 찾은 결과이다. 각 알고리즘은 300번의 반복과정을 통해서 최적해를 찾으며 이러한 과정을 총 100회 수행하였다. 그림 5의 표 안에 범례들은 100회 중 10번의 알고리즘 수행 시 결과를 보이기 위해 매 10번째 알고리즘 수행만을 그래프로 표현하였다. 함수에 따라 수렴형태는 다르지만 Uniform 분포가 Gaussian 분포보다 수렴 성능이 우수한 것을 확인할 수 있다.



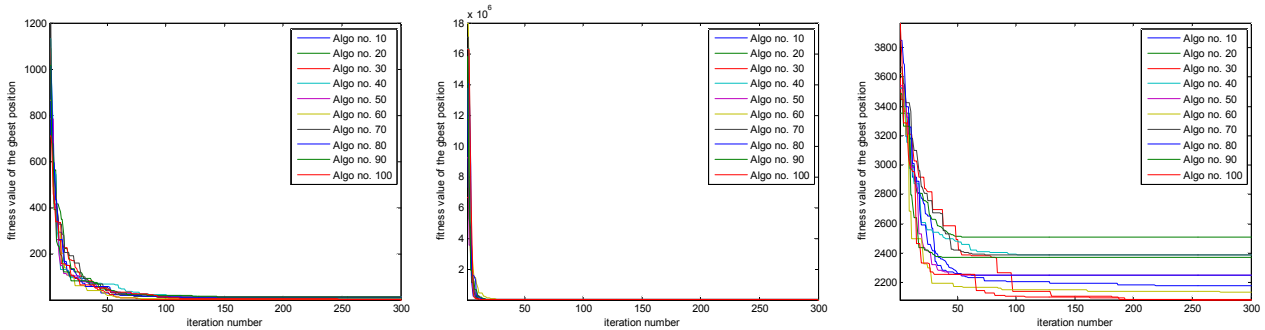
f_1, Beta(a=12, b=5), Dimension = 10 *f_2, Beta(a=12, b=5), Dimension = 10* *f_3, Beta(a=12, b=5), Dimension = 10*



f_1, Beta(a=5, b=12), Dimension = 10 *f_2, Beta(a=5, b=12), Dimension = 10* *f_3, Beta(a=5, b=12), Dimension = 10*



f_1, Beta(a=3.8, b=1.2), Dimension = 10 *f_2, Beta(a=3.8, b=1.2), Dimension = 10* *f_3, Beta(a=3.8, b=1.2), Dimension = 10*

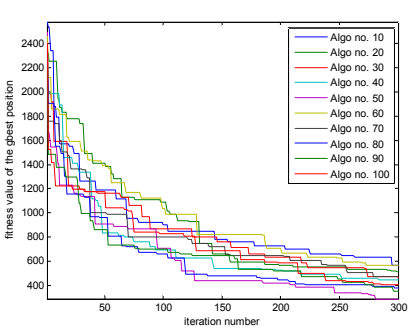


f_1, Beta(a=1.2, b=3.8), Dimension = 10 *f_2, Beta(a=1.2, b=3.8), Dimension = 10* *f_3, Beta(a=1.2, b=3.8), Dimension = 10*

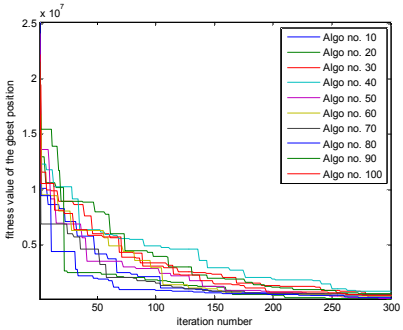
그림 6. 벤치마크 함수에 대한 확률분포에 따른 PSO-FIW 알고리즘 성능 비교 #2(차원 = 10).

Fig. 6. PSO-FIW algorithm performance comparison according to the probability distributions on the benchmark functions #2(Dim. = 10).

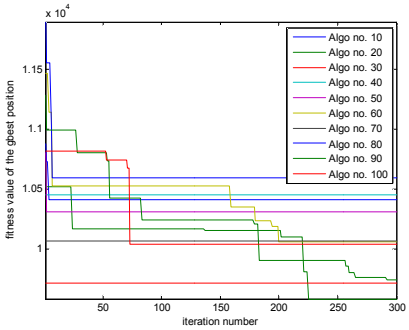
그림 6은 Beta 확률분포에서 2개의 형상모수 a, b를 한쪽 모수가 크도록 각각 임의의 차이를 두어 서로 다른 Beta 확률분포를 갖도록 한 후, 함수 f_1, f_2, f_3에 대하여 10차원과 30차원의 경우에 PSO-FIW 알고리즘을 적용하여 최적해를 찾은 결과이다. 그림 6의 결과 중 형상모수 (5, 12)와 (1.2, 3.8)의 경우 다른 형상모수의 Beta 확률분포를 적용한 알고리즘보다 수렴 성능이 우수한 것을 알 수 있다.



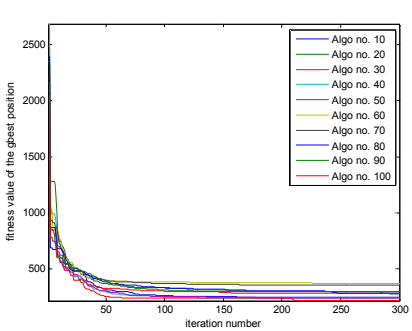
f_1, Beta(a=12, b=5), Dimension = 30



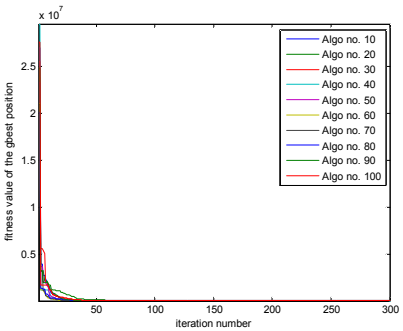
f_2, Beta(a=12, b=5), Dimension = 30



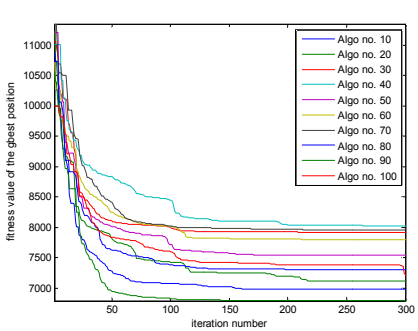
f_3, Beta(a=12, b=5), Dimension = 30



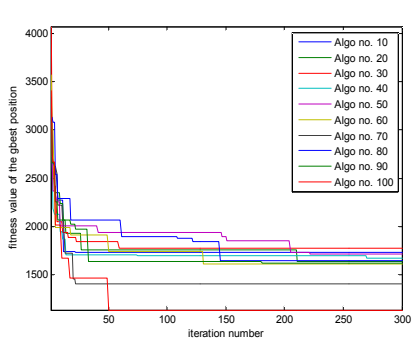
f_1, Beta(a=5, b=12), Dimension = 30



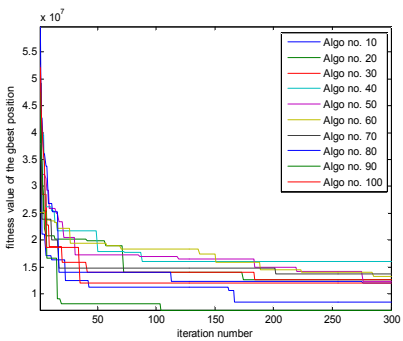
f_2, Beta(a=5, b=12), Dimension = 30



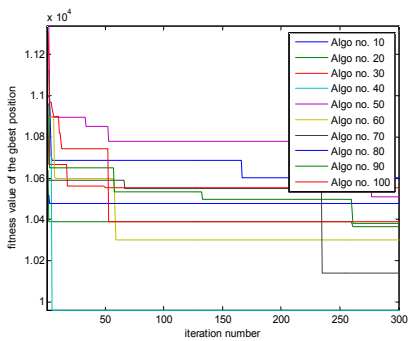
f_3, Beta(a=5, b=12), Dimension = 30



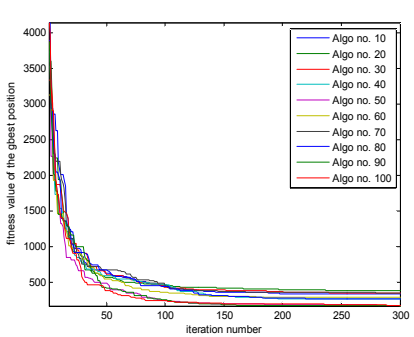
f_1, Beta(a=3.8, b=1.2), Dimension = 30



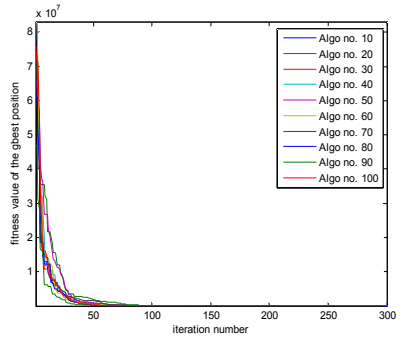
f_2, Beta(a=3.8, b=1.2), Dimension = 30



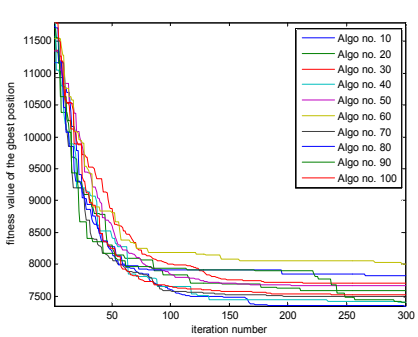
f_3, Beta(a=3.8, b=1.2), Dimension = 30



f_1, Beta(a=1.2, b=3.8), Dimension = 30



f_2, Beta(a=1.2, b=3.8), Dimension = 30

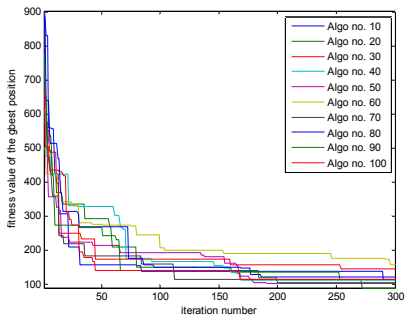


f_3, Beta(a=1.2, b=3.8), Dimension = 30

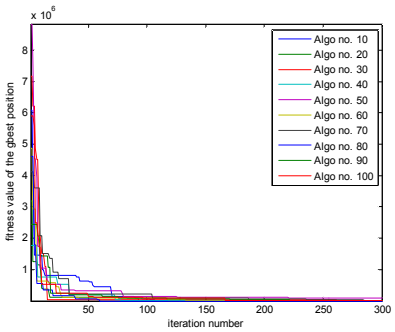
그림 7. 벤치마크 함수에 대한 확률분포에 따른 PSO-FIW 알고리즘 성능 비교 #3(차원 = 30).

Fig. 7. PSO-FIW algorithm performance comparison according to the probability distributions on the benchmark functions #3(Dim. = 30).

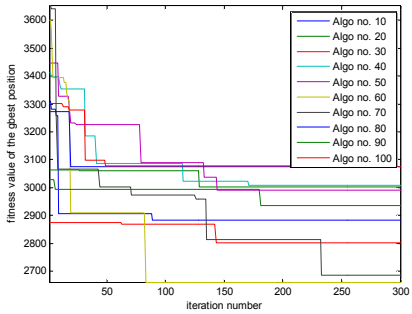
그림 7의 경우는 그림 6에서 함수의 차원만 10차원에서 30차원으로 변경한 후, 함수 f_1, f_2, f_3에 대하여 PSO-FIW 알고리즘을 적용하여 최적해를 찾은 결과이다. 그림 6의 결과와 같이 형상모수 (5, 12)와 (1.2, 3.8)의 경우 다른 형상모수의 Beta 확률분포를 적용한 알고리즘보다 수렴 성능이 우수한 것을 알 수 있다.



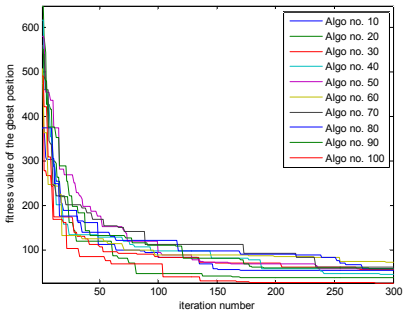
f_1, Beta(a=0.6, b=0.6), Dimension = 10



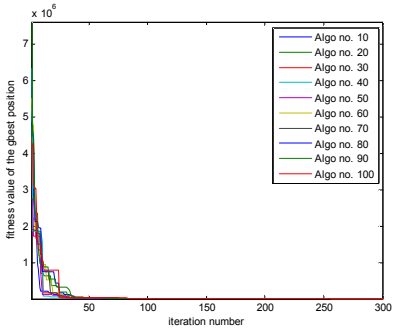
f_2, Beta(a=0.6, b=0.6), Dimension = 10



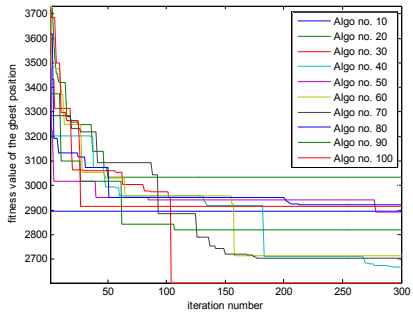
f_3, Beta(a=0.6, b=0.6), Dimension = 10



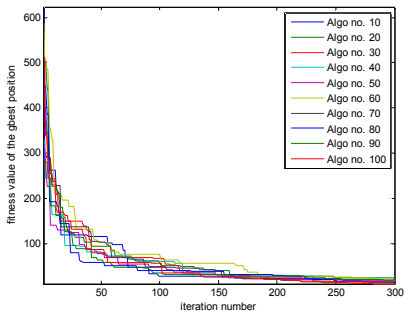
f_1, Beta(a=1.2, b=1.2), Dimension = 10



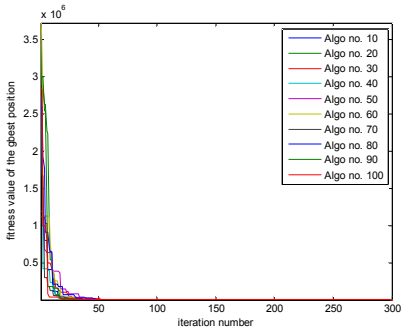
f_2, Beta(a=1.2, b=1.2), Dimension = 10



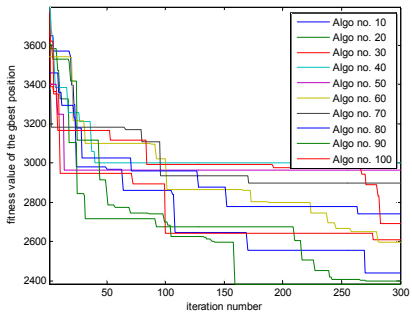
f_3, Beta(a=1.2, b=1.2), Dimension = 10



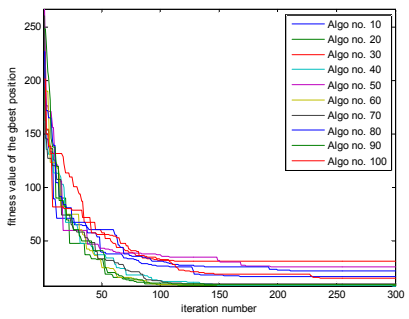
f_1, Beta(a=1.8, b=1.8), Dimension = 10



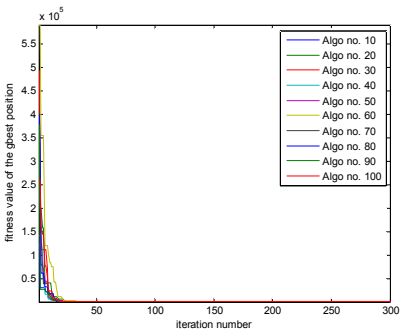
f_2, Beta(a=1.8, b=1.8), Dimension = 10



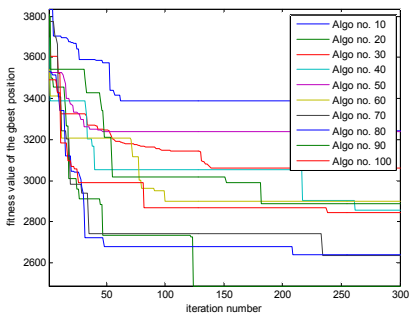
f_3, Beta(a=1.8, b=1.8), Dimension = 10



f_1, Beta(a=5.6, b=5.6), Dimension = 10



f_2, Beta(a=5.6, b=5.6), Dimension = 10

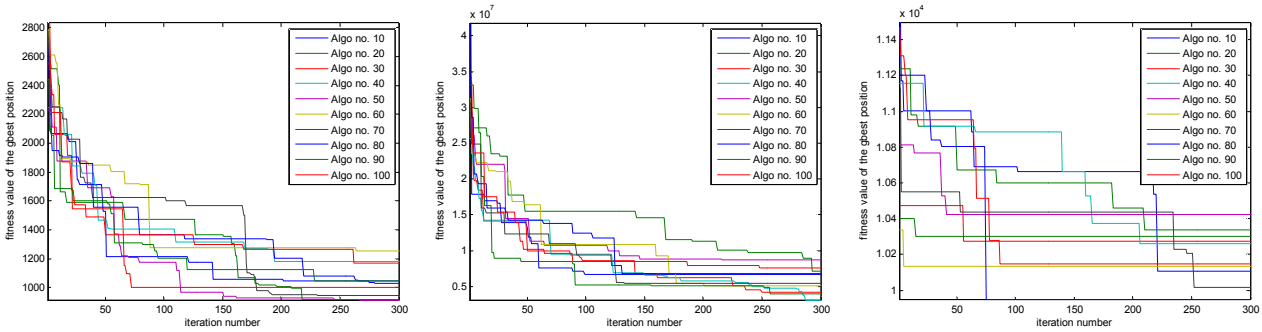


f_3, Beta(a=5.6, b=5.6), Dimension = 10

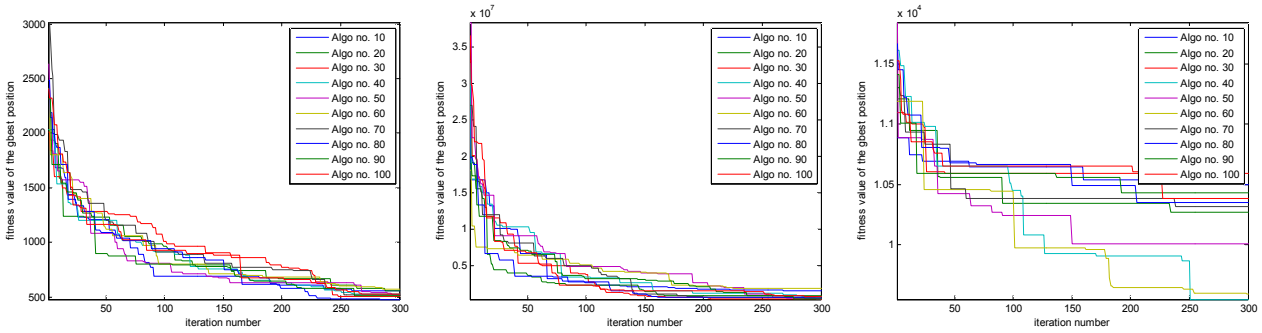
그림 8. 벤치마크 함수에 대한 확률분포에 따른 PSO-FIW 알고리즘 성능 비교 #4(차원 = 10).

Fig. 8. PSO-FIW algorithm performance comparison according to the probability distributions on the benchmark functions #4(Dim. = 10).

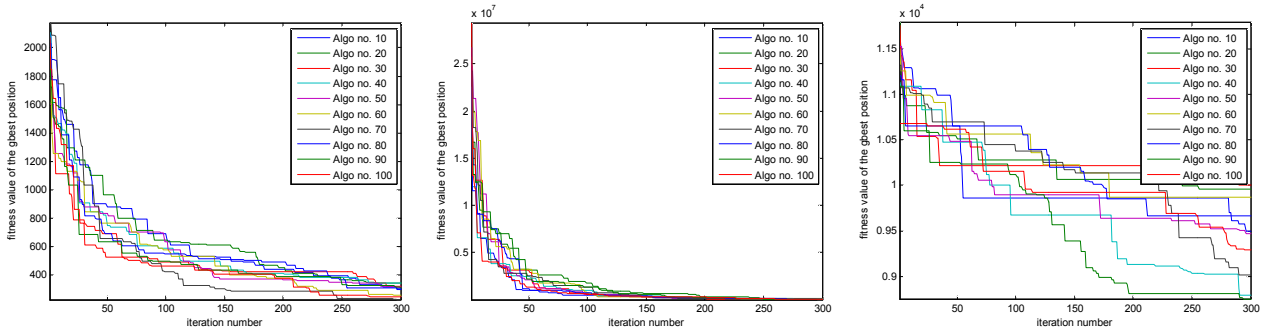
그림 8은 Beta 확률분포에서 2개의 형상모수 a, b의 값을 같게 하여 4가지 임의의 값으로 설정하였다. 이 때, 함수 f_1, f_2, f_3에 대하여 10차원과 30차원의 경우에 PSO-FIW 알고리즘을 적용하여 최적해를 찾은 결과이다. 그림 8의 결과 중 형상모수 (5.6, 5.6)이 10차원의 f_1과 f_2에서 가장 작은 최적해를 찾았으며, 수렴곡선이 급격하게 감소함을 알 수 있다. 하지만, 표 6에서도 확인 할 수 있듯이 10차원의 f_3에 대해서는 형상모수 (0.6, 0.6), (1.2, 1.2), (1.8, 1.8) 보다 더 큰 최소값을 찾고 있다.



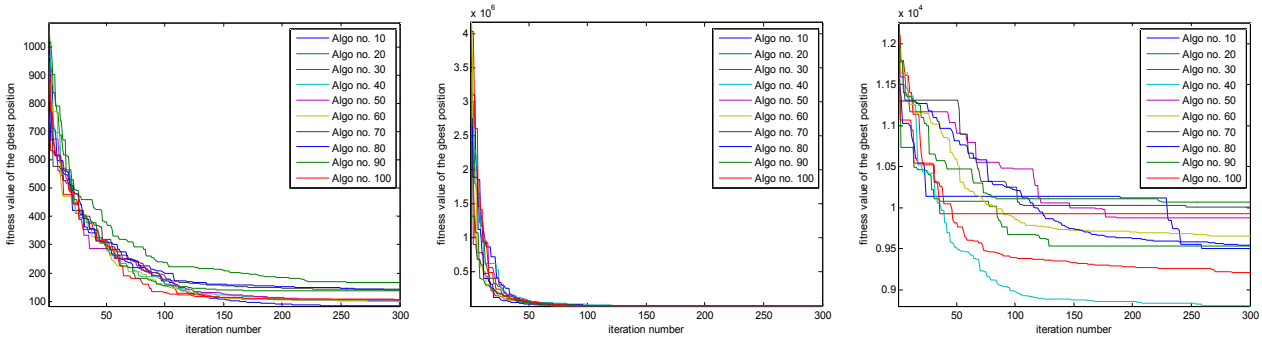
f_1, Beta(a=0.6, b=0.6), Dimension = 30 *f_2, Beta(a=0.6, b=0.6), Dimension = 30* *f_3, Beta(a=0.6, b=0.6), Dimension = 30*



f_1, Beta(a=1.2, b=1.2), Dimension = 30 *f_2, Beta(a=1.2, b=1.2), Dimension = 30* *f_3, Beta(a=1.2, b=1.2), Dimension = 30*



f_1, Beta(a=1.8, b=1.8), Dimension = 30 *f_2, Beta(a=1.8, b=1.8), Dimension = 30* *f_3, Beta(a=1.8, b=1.8), Dimension = 30*



f_1, Beta(a=5.6, b=5.6), Dimension = 30 *f_2, Beta(a=5.6, b=5.6), Dimension = 30* *f_3, Beta(a=5.6, b=5.6), Dimension = 30*

그림 9. 벤치마크 함수에 대한 확률분포에 따른 PSO-FIW 알고리즘 성능 비교 #5(차원 = 30).

Fig. 9. PSO-FIW algorithm performance comparison according to the probability distributions on the benchmark functions #5(Dim. = 30).

그림 9는 그림 8에서 함수의 차원만 10차원에서 30 차원으로 변경한 후, 함수 f_1, f_2, f_3에 대하여 PSO-FIW 알고리즘을 적용하여 최적해를 찾은 결과이다. 그림 9 의 결과에서 볼 수 있듯이 형상모수 (5.6, 5.6)의 경우 f_3을 제외하고 모든 경우에 있어 나머지 형상모수를 갖는 Beta 확률분포의 경우보다 더 작은 최적해를 찾고 수렴속도 또한 빠르다. 작은 차이(0.4)지만 표 6에서 자세히 확인 할 수 있듯이 30차원의 f_3의 경우 가장 작은 최적값은 형상모수 (1.8, 1.8)을 갖는 Beta 분포일 경우이다.

REFERENCES

[1] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," *Proc. of 6th International Symposium on Micro Machine and Human Science*, pp. 39-43, Oct. 1995.

[2] J. Kennedy and R. Eberhart, "Particle swarm optimization," *Proc. of IEEE International Conf. on Neural Networks*, vol. 4, pp. 1942-1948, Nov./Dec. 1995.

[3] J. Sun, C.-H. Lai, and X.-J. Wu, *Particle Swarm Optimisation : Classical and Quantum Perspectives*, CRC Press, Taylor & Francis Group, London, 2012.

[4] A. P. Engelbrecht, *Fundamentals of Computational Swarm Intelligence*, John Wiley & Sons Ltd., Chippenhan, Wiltshire, England, 2005.

[5] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," *Proc. of IEEE International Conf. on Evolutionary Computation*, pp. 69-73, May 1998.

[6] M. Clerc and J. Kennedy, "The particle swarm-explosion, stability, and convergence in a multidimensional complex space," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 1, pp. 58-73, Feb. 2002.

[7] C. Reynolds, "Flocks, herds, and schools : a distributed behavioral model," *Proc. of SIGGRAPH '87 Conf. on Computer Graphics*, pp. 25-34, Jul. 1987.

[8] H. H. Rosenbrock, "An automatic method for finding the greast or least of a function," *Computer Journal*, pp. 175-184, 1960.

[9] L. A. Rastrigin, *External Control System*, Theoretical Foundations of Engineering Cybernetics Series, Nauka, Moscow, Russia, 1974.

[10] H. P. Schwefel, *Numerical Optimization of Computer Models*, John Wiley & Sons, Chichester, U.K., 1981.

[11] Y. Shi and R. C. Eberhart, "Empirical study of particle swarm optimization," *Proc. of the Congress on Evolutionary Computation*, vol. 3, pp. 1945-1950, 1999.

[12] Y. Zheng, L.-H. Ma, L. Zhang, and J. Qian, "On the convergence analysis and parameter selection in particle swarm optimization," *Proc. of IEEE International Conf. on Machine Learning and Cybernetics*, vol. 3, pp. 1802-1807, Nov. 2003.

[13] S. H. Park, H. T. Kim, and K. T. Kim, "Improved auto-focusing of stepped-fequency ISAR images using new form of particle swarm optimisation," *IET Journals & Magazines on Electronics Letters*, vol. 45, no. 20, pp. 1053-1055, Sep. 2009.

[14] P. Zhang, P. Wei, and H.-Y. Yu, "Biogeography-based optimisation search algorithm for block matching motion estimiation," *IET Journals & Magazines on Image Processing*, vol. 6, no. 7, pp. 1014-1023, Oct. 2012.

[15] A. Kusiak and Z. Zhang, "Adaptive control of a wind turbine with data mining and swarm intelligence," *IEEE Transactions on Sustainable Energy*, vol. 2, no. 1, pp. 28-36, Jan. 2011.

[16] B. Yang, Y. Chen, and Z. Zhao, "Survey on applications of particle swarm optimization in electric power systems," *Proc. of IEEE International Conf. on Control and Automation*, pp. 481-486, May 2007.

[17] Wang Xin, Li Ran, Wang Yanghua, Peng Yong, and Qin Bin, "Self-tuning PID controller with variable parameters based on particle swarm optimization," *Proc. of IEEE International Conf. on Intelligent System Design and Engineering Applications*, pp. 1264-1267, Jan. 2013.

[18] Nguyen Quang Uy, N. X. Hoai, RI. Mckay, and P. M. Tuan, "Initialising PSO with randomised low-discrepancy sequences: the comparative results," *Proc. of IEEE Congress on Evolutionary Computation*, pp. 1985-1992, Sep. 2007.

[19] M. Pant, R. Thangaraj, C. Grosan, and A. Abraham, "Improved particle swarm optimization with low-discrepancy sequences," *Proc. of IEEE Congress on Evolutionary Computation*, pp. 3011-3018, Jun. 2008.

[20] R. Thangaraj, M. Pant, and K. Deep, "Initializing PSO with probability distributions and low-discrepancy sequences : the comparative results," *Proc. of World Congress on Nature & Biologically Inspired Computing*, pp. 1121-1126, Dec. 2009.

[21] J. Peng, Y. Chen, and R. C. Eberhart, "Battery pack state of charge estimator design using computational intelligence approaches," *Proc. of the Annual Battery Conference on Applications and Advances*, pp. 173-177, 2000.

[22] T. Peram, K. Veeramachaneni, and C. K. Mohan, "Fitness-distance-ratio based particle swarm optimization," *Proc. of the IEEE Swarm Intelligence Symposium*, pp. 174-181, Apr. 2003.

[23] Y. Shi and R. C. Eberhart, "Fuzzy adaptive particle swarm optimization," *Proc. of the IEEE Congress on Evolutionary Computation*, vol. 1, pp. 101-106, May 2003.

[24] Y. Zheng, L. Ma, L. Zhang, and J. Qian, "Empirical study of particle swarm optimizer with increasing inertia weight," *Proc. of the IEEE Congress on Evolutionary Computation*, vol. 1, pp. 221-226, Dec. 2003.

이 병 석

2002년 2월 서울시립대학교 전자전기공학부 졸업. 2002년 7월~2005년 9월 통신장교 복무. 2009년 2월 동 대학원 전자전기컴퓨터공학부 석사. 2011년 2월 동 대학 박사 수료. 2011년 4월~현재 한국항공우주연구원 위성항법팀 연구원 및 서울시립대학교 전자전기컴퓨터공학부 박사과정. 관심분야는 최적 · 강인 제어, 떼 지능, 군집 제어, 위성항법시스템(GNSS) 등.

이 준 화

1987년 2월 서울대학교 제어계측공학과 졸업. 1989년 2월 동 대학원 제어계측공학과 석사. 1994년 8월 동 대학원 제어계측공학과 박사. 1991년 7월~1994년 9월 서울대학교 제어계측신기술 연구센터 연구원. 1994년 10월~1995년 2월 California Institute of Technology(Caltech) 연구원. 1995년 3월~현재 서울시립대학교 전자전기컴퓨터공학부 교수. 관심분야는 최적제어 및 이미지 시스템 등.

허 문 범

1992년 2월 경희대학교 기계공학과 졸업. 1997년 12월 Illinois Institute of Technology 항공기계공학과 석사. 2004년 12월 Illinois Institute of Technology 항공기계공학과 박사. 2005년 10월~현재 한국항공우주연구원 위성항법팀 책임연구원. 관심분야는 GNSS, 위성항법, 항행시스템 등.