Journal of the Society of Korea Industrial and Systems Engineering
Vol. 33, No. 4, pp.58－68, December 2010.

순서화 문제에서 이산적 Particle Swarm Optimization들의 성능 비교

임 동 순†

한남대학교 산업경영공학과

Performance Comparison of Discrete Particle Swarm Optimizations in Sequencing Problems

D. S. Yim†

Dept. of Industrial and Management Engineering, Hannam University

Particle Swarm Optimization (PSO) which has been well known to solve continuous problems can be applied to discrete combinatorial problems. Several DPSO (Discrete Particle Swarm Optimization) algorithms have been proposed to solve discrete problems such as traveling salesman, vehicle routing, and flow shop scheduling problems. They are different in representation of position and velocity vectors, operation mechanisms for updating vectors. In this paper, the performance of 5 DPSOs is analyzed by applying to traditional Traveling Salesman Problems. The experiment shows that DPSOs are comparable or superior to a genetic algorithm (GA). Also, hybrid PSO combined with local optimization (i.e., 2-OPT) provides much improved solutions. Since DPSO requires more computation time compared with GA, however, the performance of hybrid DPSO is not better than hybrid GA.

Keywords : Particle Swarm Optimization (PSO), Traveling Salesman Problem (TSP), Genetic Algorithm (GA)

1. 서 론

Particle Swarm Optimization(PSO)은 먹을 것을 찾아 다니는 한 무리 새 떼의 행동을 본 따 최적화 문제를 해결할 수 있도록 고안되었다[8]. Particle이라고 불리는 각 개체는 지금까지 자신이 경험한 가장 좋은 해 그리고, 모든 개체들이 경험한 것 중 가장 좋은 해의 두 가지 정보를 이용하여 현재 위치에서 자신의 이동방향을 결정하고, 이에 따라 다음 단계에서의 위치가 결정된다. 이 같은 위치는 최적화 문제에 대한 해로 일대일 매핑되어 개체의 이동은 해 공간에서의 탐색과정을 의미한다.

n-차원 공간에서 정의되는 변수 벡터 x의 목적함수 $f(x)$에 대한 최적화 문제를 위한 기본적인 PSO 알고리즘은 m개의 개체로 구성되는 집단을 사용하고, k번째 반복에서의 i번째 개체는 연속적인 n-차원 탐색 공간에서의 해인 위치 벡터 x_i^k로 표현되어 적응값(fitness value) $f(x_i^k)$를 가진다. i번째 개체가 k번의 반복을 통해 경험한 가장 좋은 해를 p_i라고 하고, 모든 개체가 경험한 가장 좋은 해를 g라고 하자. $k+1$번째 반복에서 i번째 개체의 속도 벡터와 위치 벡터는 다음과 같이 표현된다.

$$v_i^{k+1} = w \cdot v_i^k + c_1 r_1 (p_i - x_i^k) + c_2 r_2 (g - x_i^k) \quad (1)$$

$$x_i^{k+1} = x_i^k + v_i^k \quad (2)$$

논문접수일 : 2010년 09월 01일　논문수정일 : 2010년 09월 18일　게재확정일 : 2010년 09월 28일
† 교신저자 dsyim@hnu.kr
※ 이 논문은 2010년도 한남대학교 학술연구조성비 지원에 의하여 연구되었음.

위 식에서 w, c_1, c_2는 각 항의 가중치이고, r_1, r_2는 0부터 1사이의 난수이다. 위 식은 비교적 적은 수의 파라미터를 포함하여 최적화 문제에 대한 적용이 용이하다는 장점을 가진다. 더욱이 실수값의 범위를 가지는 연속적인 n-차원 탐색 공간에서는 위치벡터가 유효한 해를 직접적으로 표현하기 때문에 PSO의 적용이 용이하다.

연속적인 해 공간 뿐만 아니라, 외판원 문제[3, 12, 14, 16], 스케쥴링[1, 2, 9, 15], 차량 라우팅[10] 등 이산적인 해 공간을 갖는 문제에서도 PSO가 적용되고 있다. 그러나 이러한 이산적인 값을 갖는 n-차원 탐색 공간에서의 최적화 문제에서는 PSO의 적용이 그다지 용이하지 않다. 해가 순열(permutation)로 표현되는 순서화(sequencing) 문제에서는 PSO의 적용을 더욱 어렵게 한다. 우선, 속도벡터와 위치벡터를 구하는 위의 두 식에 요구되는 연산은 벡터 x_i^k의 성분 값이 실수라는 것을 가정한다. 만약 x_i^k를 순열로 표현할 경우 위의 두 식에 의한 연산은 의미를 잃어버리고, 유효한 해를 생성하지 못한다. 그러나 순열 형태의 해를 실수의 벡터인 x_i^k로 코드화(encoding)할 수 있다면 위의 두식에 따른 연산은 수행 가능하다. 이 경우 코드화뿐 만 아니라 x_i^k를 유효한 순열의 해로 바꾸어주는 복호화(decoding) 작업이 요구된다. 코드화와 복호화 절차가 필요한 순열 형태로 해를 표현하는 방법 외에 보다 효과적으로 PSO 연산을 적용할 수 있도록 다양한 위치 벡터에 대한 표현방법이 제안되었다. 지금까지 순서화 문제에 적용된 PSO들에서 위치 벡터를 표현하기 위해 개발된 방법은 크게 4가지로 나뉜다.

1) 순열 표현(Permutation representation)
2) 실수 표현(Real valued representation)
3) 에지 표현(Edge representation)
4) 행렬 표현(Matrix representation)

각 표현 방법들은 식 (1), 식 (2)에서 정의된 속도와 위치 벡터를 구하는 고유의 연산 방법을 포함한다. 위치벡터를 X_1, X_2라고 하고, w를 상수, 두 속도 벡터를 V_1, V_2라고 하자. PSO의 적용을 위해 다음의 4가지 기본 연산이 요구된다.

1) $V' = X_1 - X_2$: 두 위치 벡터의 차이로 속도 벡터의 의미를 가진다.
2) $V' = w \cdot V_1$: 상수와 속도벡터의 곱으로 속도 벡터의 의미를 가진다.
3) $V' = V_1 + V_2$: 두 속도 벡터의 합으로 속도 벡터의 의미를 가진다.
4) $X' = X_1 + V_1$: 위치 벡터와 속도벡터의 합으로 위치 벡터의 의미를 가진다.

본 연구는 순서화 문제에 적용되는 4가지 위치 벡터 표현 방법과 이에 따른 연산 방법의 성능에 대한 분석을 목적으로 한다. 각각의 표현 방법에 대한 성능은 제안된 논문들에서 분석되었지만, 다양한 DPSO(Discrete Particle Swarm Optimization)들에 대한 비교 분석은 수행되지 않아 이에 대한 연구가 요구된다. 본 논문에서는 비교 분석을 통해 각 표현 방법에 대한 장단점을 도출하고, 개선점을 제시한다. 분석은 전형적인 순서화 문제인 외판원 문제(TSP)를 대상으로 실험하여 수행되었다.

2. PSO 알고리즘

PSO의 전반적인 절차는 다음과 같다.

1) 초기화를 수행한다.
2) 종료 조건을 만족할 때까지 k를 증가하여 다음을 반복 수행한다.
   2-1) 각 개체(i번째 particle)에 대해 다음을 수행한다.
      2-1-1) 개체의 현재 위치벡터 x_i^k를 유효한 현재해 z_i^k로 변환한다.
      2-1-2) 개체의 적응값 $f(x_i^k)$을 계산한다.
      2-1-3) 만약 현 적응값이 개체가 경험한 가장 좋은 적응값 보다 좋다면 p_i를 현재 위치벡터로 치환한다.
   2-2) 모든 개체의 현재 해 중 가장 좋은 해가 지금까지의 가장 좋은 해보다 좋다면 g를 현재해 중 가장 좋은 해의 위치벡터로 치환한다.
   2-3) 각 개체에 대해 다음을 수행한다.
      2-3-1) 개체의 속도 벡터를 수정한다.
      2-3-2) 개체의 위치 벡터를 수정한다.

PSO 알고리즘은 집단에 기초한 방법으로 m개의 개체를 유지한다. 단계 (1)에서 m개의 개체에 대한 속도벡터와 위치벡터를 초기화 한 후 단계 (2)를 반복 실행한다. 단계 (2-1)은 각 개체의 위치 벡터로부터 해와 적응값을 구하고, 개체가 경험한 것 중 가장 좋은 해에 해당하는 위치벡터인 p_i를 수정하는 절차이다. 단계 (2-1-1)는 위치벡터를 해로 복호화하는 절차로 순열 표현에서와 같이 위치 벡터의 표현이 해의 표현과 동일하다면 복호화는 불필요하다. 단계 (2-2)는 모든 객체가 경험한

것 중 가장 좋은 해에 해당하는 위치 벡터인 g를 수정하는 절차이고, 단계 (2-3)은 식 (1), 식 (2)에 따라 속도벡터와 위치 벡터를 수정하는 절차이다. 종료 조건은 일반적으로 단계 (2)의 최대 반복 수에 의한다.

3. 순서화 문제에서의 PSO 해 표현 방법

3.1 순열 표현 방법

순열 표현 방법에서는 위치 벡터 x_i^k의 성분들이 순열 형태를 가져 순서화 문제의 해를 직접적으로 나타낸다. 이 같은 표현 방법에 의한 두 가지 방법을 소개한다.

3.1.1 방법 1(PP1)

Hu et al.[6]에 의해 개발된 순열 표현 방법에서는 다음과 같은 연산이 정의 된다.

1) $V' = X_1 - X_2$
   $v'_j = |x_{1j} - x_{2j}|,\ j = 1, \cdots, n$
2) $V' = w \cdot V_1$
   $v'_j = w \cdot v_{1j},\ j = 1, \cdots, n$
3) $V' = V_1 + V_2$
   $v'_j = v_{1j} + v_{2j},\ j = 1, \cdots, n$
4) $X' = X_1 + V_1$

   Step 1 : // V_1의 성분 중 가장 큰 값을 찾는다.

   $$v_{\max} = \max_j \{v_{1j}\}$$

   Step 2 :
   // V_1의 각 성분을 (0, 1)의 범위로 정규화 한다.

   $$v'_{1j} = v_{1j} / v_{\max}$$

   Step 3 : // 두 위치의 성분을 바꾼다.
   X_1을 X'으로 복사한다.

```
for each j {
 if rand < v'_1j {
     find i where x_i' = g_j
     x_j' = x_i', x_i' = x_j' // Swap x_j' and x_i'
     if (j==i) perform mutation (select
        l <> i  randomly then swap x_j'
        and x_l')
   }
}
```

두 위치벡터의 차를 구하는 연산 (1)과 벡터에 대한 통상적인 연산인 (2), (3)번 연산을 통하여 속도 벡터는 양의 실수 성분을 가지게 된다. (4)번 연산에서는 이전의 위치벡터와 속도벡터를 이용하여 새로운 위치벡터를 구한다. 위치 벡터는 순열 표현이므로 통상적인 벡터 연산이 의미를 갖지 못한다. 때문에 위치벡터를 구하기 위해 특별한 방법이 고안되었다. (4)번 연산의 Step 1과 Step 2는 실수 성분의 속도벡터 값들을 0부터 1 사이의 확률로 변환하는 과정이다. 마지막 Step 3에서는 생성된 확률을 이용해 랜덤으로 위치 벡터의 값들에 대한 위치를 바꾼다. 연산이 가해지는 위치 벡터는 순열 표현이므로 값들의 위치가 바뀐 새로운 위치벡터는 순열 형태의 표현을 유지하게 된다. 구체적인 설명을 위해 다음과 같은 위치벡터와 속도벡터의 예를 보자.

$$X_1 = [2\ \ 3\ \ 1\ \ 4\ \ 5]$$
$$V_1 = [1\ \ 0.5\ \ 2\ \ 1.5\ \ 1]$$
$$G = [5\ \ 1\ \ 3\ \ 4\ \ 2]$$

$X' = X_1 + V_1$을 구하는 (4)번 연산의 Step 1과 Step 2를 통해 다음의 확률 벡터 V_1'을 구한다.

$$V_1' = [0.5\ \ 0.25\ \ 1\ \ 0.75\ \ 0.5]$$

Step 3에서는 확률적으로 X_1 각 성분에 대한 위치를 바꾼다. 예를 들어 생성된 일련의 난수를 0.4, 0.6, 0.1, 0.2, 0.9라고 하자. 첫 번째 난수 0.4는 확률벡터 V_1'의 첫 번째 성분 0.5보다 작으므로 X_1의 첫 번째 성분인 2에 대해 위치를 바꾼다. 지금까지 가장 좋은 해인 G에서 2는 5번째 성분이므로 X_1에서 첫 번째 성분인 2와 5번째 성분인 5의 위치를 변경한다. 이 같은 절차를 반복한 결과는 다음과 같다.

1) $X' = [2\ \ 3\ \ 1\ \ 4\ \ 5]$
2) $X' = [5\ \ 3\ \ 1\ \ 4\ \ 2]$: 난수 0.4 < 0.5이므로 (2, 5)가 교환되었다.
3) $X' = [5\ \ 3\ \ 1\ \ 4\ \ 2]$: 난수 0.6 > 0.25이므로 3은 교환되지 않았다.
4) $X' = [5\ \ 1\ \ 3\ \ 4\ \ 2]$: 난수 0.1 < 1이므로 (1, 3)이 교환되었다.
5) $X' = [5\ \ 4\ \ 3\ \ 1\ \ 2]$: 난수 0.2 < 0.75이므로 (4, 4)를 교환하여야 하나 같은 수이므로 4와 임의의 위치에 있는 1이 교환되는 돌연변이를 수행하였다.
6) $X' = [5\ \ 4\ \ 3\ \ 1\ \ 2]$: 난수 0.9 > 0.5이므로 2는 교환되지 않았다.

최종적으로 생성된 새로운 위치벡터 X'은 유효한 순열 표현을 나타낸다.

3.1.2 방법 2(PP2)

Anghinolfi and Paolucci[1]에 의해 개발된 순열 표현 방법에서는 다음과 같은 연산이 정의 된다.

1) $V' = X_1 - X_2$
   $v'_j = s - j, j = 1, \cdots, n$. 단, $x_{1j} = x_{2s}$
2) $V' = w \cdot V_1$
   $v'_j = w \cdot v_{1j}, j = 1, \cdots, n$
   if $v'_j <> [v'_j]$ {
   generate random r
   if $(r < 0.5)$ then $v'_j = [v'_j]$
   else {
   if $v'_j > 0 v'_j = [v'_j]+1$
   else $v'_j = [v'_j]$-1
   }
   }
3) $V' = V_1 + V_2$
   $v'_j = v_{1j} + v_{2j}, j = 1, \cdots, n$
4) $X' = X_1 + V_1$
   Step 1 :
   Queue $Q_1, Q_2, \cdots Q_n$를 초기화한다.
   Step 2 :
   for each j {
   $i = j + v_{1j}$
   add x_{1j} to Q_i
   }
   Step 3:
   for each j {
   if size of Q_j = 0 {
   for each l {
   if size of $Q_l > 1$ {
   delete y from Q_l
   add y to Q_l
   break
   }
   }
   }
   }
   for each j {
   delete y from Q_j
   $x'_j = y$
   }

이 방법도 Hu 등의 방법과 같이 위치벡터에 속도벡터를 더하는 (4)번 연산을 위한 특별한 방법이 고안되었다. 4가지 연산의 자세한 설명을 위해 다음과 같은 예를 보자.

$$X_1 = [2\ 3\ 1\ 4\ 5]$$
$$X_2 = [3\ 5\ 2\ 4\ 1]$$

연산 (1)에 의해 두 위치 벡터의 차인 속도 벡터 V_1은 다음과 같다.

$$V_1 = X_1 - X_2 = [2\ -1\ 2\ 0\ -3]$$

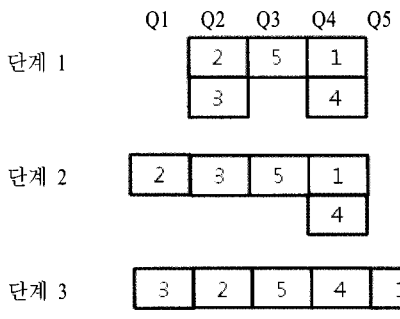
X_1의 첫 번째 성분인 2는 X_2에서 3번째에 위치하고 있어 두 위치의 차 2가 V_1의 첫 번째 성분이 된다. 같은 방법에 의해 X_1의 두 번째 성분인 3은 X_2에서 1번째에 위치하고 있어 두 위치의 차 -1가 V_1의 두 번째 성분이 된다. 이 속도 벡터에 0.5를 곱한 V_2를 구해보자. 우선 벡터와 스칼라의 곱인 V_2^0를 구한다.

$$V_2^0 = 0.5 \cdot [2\ -1\ 2\ 0\ -3] = [1\ -0.5\ 1\ 0\ -1.5]$$

(2)번 연산에서 정의된 방법에 의해 정수 값을 성분으로 하는 속도벡터 V_2를 구한다. 생성된 난수를 0.4, 0.8이라고 하자. V_2^0의 첫 번째 성분 1은 정수이므로 변환이 필요 없다. 두 번째 성분 -0.5는 실수이므로 난수 0.4(< 0.5)에 의해 소수점 이하를 없앤 0으로 변환된다. 마지막 값 -1.5에 대해 난수 0.8(> 0.5) 이므로 -2(= -1 - 1)로 계산된다.

$$V_2 = [1\ 0\ 1\ 0\ -2]$$

마지막 연산에 해당하는 $X_1 + V_2$를 하기 위하여 <그림 1>에서와 같이 5개의 큐를 만들고 $X_1 = [2\ 3\ 1\ 4\ 5]$의 각 성분을 V_2의 성분만큼 이동시킨 위치의 큐에 넣는다(단계 1). 그 다음 왼쪽부터 빈 큐를 찾는다. Q_1이 비었으므로 이동 가능한 가장 처음의 값인 Q_2의 2를 Q_1으로 이동시킨다(단계 2). Q_5가 비었으므로 이동 가능한 가장 처음의 값인 $Q_4$4의 1을 이동시킨다(단계 3). 단계 3의 (3 2 5 4 1)은 결과값이 된다.



<그림 1> 순열 표현 PSO에서의 위치벡터와 속도벡터의 합 연산

3.2 실수 표현 방법(PR)

위치 벡터 x_i^k는 실수 값의 벡터로 표현되고, 특별한 복호화 방법에 의해 해 $Z=\{z_1, z_2, \cdots, z_n\}$으로 변환된다. 복호화 방법으로 널리 이용되는 SPV(Smallest Position-ing Value) 방법[15]을 소개한다. SPV 방법에서는 위치 벡터와 속도벡터가 실수 값을 가져 일반적인 벡터 연산이 적용된다.

1) $V' = X_1 - X_2$
   $v'_j = x_{1j} - x_{2j}, j=1, \cdots, n$
2) $V' = w \cdot V_1$
   $v'_j = w \cdot v_{1j}, j=1, \cdots, n$
3) $V' = V_1 + V_2$
   $v'_j = v_{1j} + v_{2j}, j=1, \cdots, n$
4) $X' = X_1 + V_1$
   $x'_j = x_{1j} + v_{1j}, j=1, \cdots, n$

연산을 통해 구해진 위치 벡터 $X=\{x_1, x_2, \cdots, x_n\}$를 해 으로 변환하는 복호화 방법은 다음과 같다.

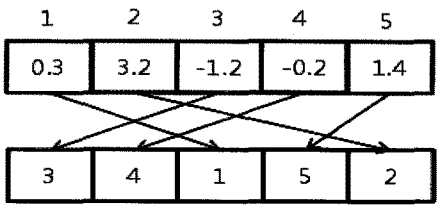
복호화 방법
```
for each j {
    find s where x_s = min_l{x_l}, l = 1, ..., n

    z_j = s
    x_s = ∞
}
```

예를 들어, 다음과 같이 실수 성분을 갖는 위치 벡터를 해로 변환한다고 하자.

$$X=[0.3 \;\; 3.2 \;\; -1.2 \;\; -0.2 \;\; 1.4]$$

가장 작은 값은 -1.2로 3번째 값이다. 따라서, 해의 첫 번째 위치에 3을 놓는다. 두 번째로 작은 값은 -0.2로 4번째 수이다. 4를 두 번째에 위치한다. 이런 방법에 의해 다음 <그림 2>와 같은 해 (3, 4, 1, 5, 2)를 생성한다.



<그림 2> 실수 표현 SPV에서의 복호화

3.3 행렬 표현 방법(PM)

해의 위치와 속도를 정방행렬로 표현하는 방법으로 Pang 등[12]에 의해 개발된 방법을 소개한다. 행렬의 각 성분이 실수값을 가지므로 (1), (2), (3)번 연산은 일반적인 행렬 연산에 따른다. (4)번 연산은 행렬 계산과 동일하나 부가적으로 정규화를 시킨다. 위치 행렬은 특별한 복호화 방법에 의해 해 $Z=\{z_1, z_2, \cdots, z_n\}$으로 변환된다.

1) $V' = X_1 - X_2$
   $v'_{ij} = x_{1ij} - x_{2ij}, i=1, \cdots, n, j=1, \cdots, n$
2) $V' = w \cdot V_1$
   $v'_{ij} = w \cdot v_{ij}, i=1, \cdots, n, j=1, \cdots, n$
3) $V' = V_1 + V_2$
   $v'_{ij} = v_{1ij} + v_{2ij}, i=1, \cdots, j=1, \cdots, n$
4) $X' = X_1 + V_1$
   $x'_{ij} = x_{1ij} + v_{1ij}, i=1, \cdots, n, j=1, \cdots, n$
   // 정규화
   for each j {
      sum = 0
      for each i {
         if $x'_{ij} < 0$ then $x'_{ij} = -1 \times x'_{ij}$
         sum + $= x'_{ij}$
      }
      for each i
         $x'_{ij} = x'_{ij}$ / sum
   }

정규화된 위치 행렬 X를 유효한 해 $Z=\{z_1, z_2, \cdots, z_n\}$로 복호화하는 절차는 다음과 같다.

복호화 방법
```
initialize matrix Y_{n×n}
copy X to Y
for each l = 1, ..., n {
    find max y_ij, for every i and j
    z_i = j
    for each s {
        y_sj = -1
        y_js = -1
    }
}
```

위 복호화 방법을 설명하기 위해 예를 들어 설명한다. 4)번 연산을 통해 정규화된 위치행렬을 다음과 같

다고 하자.

$$X=\begin{bmatrix}0.4 & 0.3 & 0.1 & 0.1\\ 0.1 & 0.2 & 0.4 & 0.1\\ 0.3 & 0.2 & 0.3 & 0.1\\ 0.2 & 0.3 & 0.2 & 0.7\end{bmatrix}$$

가장 큰 성분은(4행, 4열)에 있는 0.7이다. 4를 Z 벡터의 4번째에 위치시킨다. X를 복사한 Y에서 4번째 행과 4번째 열의 성분 값을 -1로 놓는다.

$$Y=\begin{bmatrix}0.4 & 0.3 & 0.1 & -1\\ 0.1 & 0.2 & 0.4 & -1\\ 0.3 & 0.2 & 0.3 & -1\\ -1 & -1 & -1 & -1\end{bmatrix}$$

Y에서 가장 큰 성분은 (1행, 1열)과 (2행, 3열)의 0.4이다. 이중 랜덤으로 (1행, 1열)을 우선적으로 선택하여 1을 1번째에 위치 시킨다. 1번째 행과 1번째 열의 성분 값을 -1로 놓는다.

$$Y=\begin{bmatrix}-1 & -1 & -1 & -1\\ -1 & 0.2 & 0.4 & -1\\ -1 & 0.2 & 0.3 & -1\\ -1 & -1 & -1 & -1\end{bmatrix}$$

Y에서 가장 큰 수는(2행, 3열)의 0.4가 선택되어 2를 3번째에 위치 시킨다.

$$Y=\begin{bmatrix}-1 & -1 & -1 & -1\\ -1 & -1 & -1 & -1\\ -1 & 0.2 & -1 & -1\\ -1 & -1 & -1 & -1\end{bmatrix}$$

Y에서 가장 큰 수는 0.2로 (3행, 2열)에 있다. 3을 2번째에 위치시킨다. 최종 해는 다음과 같다.

$$z=[1\ 3\ 2\ 4]$$

3.4 에지 표현방법(PE)

에지 표현방법인 Zhong 등[16]의 방법을 소개한다. 위치벡터는 해에 속하는 에지들의 집합으로 표현된다. 예를 들어, 노드 순서로 표현된 외판원 문제에서의 해 (2, 3, 1, 4)는 에지 집합 $X_1=\{(2,3),(3,1),(1,4),(4,1)\}$인 위치벡터로 표현된다. 속도 벡터는 에지들의 집합과 선택 확률의 혼합으로 표현된다. 예를 들어, $V_1=\{0.2(2,3)\ 0.3(3,4),0.9(4,1)\}$로 표현된 속도벡터는 에지 $(2,3)$이 0.2의 확률, $(3,4)$가 0.3의 확률, $(4,1)$이 0.9의 확률로 선택됨을 의미한다.

에지 표현방법에서의 4가지 연산은 다음과 같이 정의된다.

1) $V'=X_1-X_2$
   X_1에는 존재하나, X_2에는 존재하지 않는 에지들의 집합으로 각 에지들의 선택확률은 1이다.
2) $V'=w\cdot V_1$
   V_1의 각 에지에 대한 선택확률에 w를 곱한다.
3) $V'=V_1+V_2$
   두 속도 벡터의 모든 성분들을 포함하는 bag이다. 단, 동일한 노드를 포함하는 에지들은 최대 4이다. 집합이라기 보다 bag이므로 동일한 에지, 특히, 동일한 선택확률의 동일한 에지들도 다수 포함될 수 있다.
4) $X'=V_1+X_1$
   Step 1 : V_1에 포함된 에지들을 대상으로 유효한 에지일 경우 확률적으로 선택하여 X'으로 놓는다.
   Step 2 : X_1에 포함된 에지들을 대상으로 유효한 에지일 경우 $C_3\times r_3$의 확률에 의해 선택하여 X'에 더한다(r_3는 (0, 1)사이의 난수이다).
   Step 3 : 유효한 해를 생성하기 위해 가장 적은 거리의 유효한 에지를 X'에 더한다.

위 연산들의 이해를 돕기 위해 다음과 같은 예를 보자. 대칭 TSP 문제의 유효한 해를 나타내는 두 위치벡터가 다음과 같다고 하자.

$$X_1=\{(1,2),(2,3),(3,4),(4,5),(5,1)\}$$
$$X_2=\{(1,5),(5,3),(3,4),(4,2),(2,1)\}$$

두 위치벡터의 차 $V'=X_1-X_2$는 다음과 같이 계산된다. 에지 (2, 3)과 (4, 5)는 X_1에 포함되지만, X_2에는 포함되지 않아 V'에 포함시킨다. 에지 앞의 수는 그 에지의 선택 확률로 1이다.

$$V'=X_1-X_2=\{1(2,3),1(4,5)\}$$

이 속도 벡터에 0.3을 곱하는 두 번째 연산을 해보자. 단지 선택 확률에 0.3을 곱한다.

$$V_1=0.3V'=\{0.3(2,3),0.3(4,5)\}$$

또 다른 속도 벡터를 다음과 같다고 하자.

$$V_2=\{0.3(4,5),0.2(3,5),0.3(3,4),1\,(2,1)\}$$

두 속도 벡터의 합은 두 집합의 합인 bag로 표현된다. 에지 (4, 5)는 동일한 선택 확률로 두 개가 포함되어 있음에 유의하라.

$$V' = V_1 + V_2 = \begin{Bmatrix} 0.3(2,3), 0.3(4,5), 0.3(4,5), 0.2(3,5), \\ 0.3(3,4), 1(2,1) \end{Bmatrix}$$

이제 위치벡터 X_1과 속도벡터 V'의 합을 구하는 4)번 연산을 해보자. 일련의 난수가 0.4, 0.3, 0.7, 0.1, 0.5라고 하고 Step 1을 수행한다. V'의 각 에지에 대해 난수를 발생하여 선택확률 보다 작으면 그 에지를 X'에 포함시킨다. 단 에지는 X'의 에지 집합에 더해졌을 때 유효하여야 한다. 만약, 모든 노드를 포함하는 유효한 해가 되지 못할 경우에는 포함시키지 않는다.

1) $X' = \{\ \}$ 0.4 > 선택확률 = 0.3이므로 (2, 3)을 포함시키지 않는다.
2) $X' = \{(4,5)\}$, 0.3 <= 선택확률 = 0.5이므로 (4, 5)를 포함시킨다.
3) $X' = \{(4,5)\}$, (4, 5)는 이미 포함되어 있으므로 무시한다.
4) $X' = \{(4,5), (3,5)\}$, 0.1 <= 선택확률 = 0.2이므로 (3, 5)를 포함시킨다.
5) $X' = \{(4,5), (3,5)\}$, (3, 4)는 유효한 에지가 아니다. 부분적 서킷 (4, 5, 3)을 생성하기 때문이다.
6) $X' = \{(4,5), (3,5), (2,1)\}$, 0.5 <= 선택확률 = 1이므로 (2, 1)을 포함시킨다.

이제 Step 2를 수행하자. $c_3 = 1$, 난수 $r_3 = 0.4$라고 하자.

$$c_3 r_3 \times X_1 = \begin{Bmatrix} 0.4(1,2), 0.4(2,3), 0.4(3,4), \\ 0.4(4,5), 0.4(5,1) \end{Bmatrix}$$

또한, 일련의 난수들을 0.6, 0.5, 0.2 이라고 하자.

1) $X' = \{(4,5), (3,5), (2,1)\}$, (1, 2)는 중복되어 무시한다.
2) $X' = \{(4,5), (3,5), (2,1)\}$, 0.5 > 선택확률 = 0.4이므로 (2, 3)을 포함하지 않는다.
3) $X' = \{(4,5), (3,5), (2,1)\}$, (3, 4)는 유효한 에지가 아니므로 포함하지 않는다.
4) $X' = \{(4,5), (3,4), (2,1)\}$, (4, 5)는 중복되어 무시한다.
5) $X' = \{(4,5), (3,4), (2,1), (5,1)\}$, 0.2 <= 선택확률 = 0.4이므로 (5, 1)을 포함시킨다.

Step 3에서는 완전한 해를 만들기 위해 X'에 에지를 추가한다. 이 때 가급적 최소거리의 에지를 추가한다. X'에는 에지 (2, 3)만이 포함될 수 있는 유효한 에지이다. 따라서, 최종적인 결과는 다음과 같다.

$$X' = \{(4,5), (3,4), (2,1), (5,1), (2,3)\}$$

위의 연산들은 연산의 순서가 중요한 의미를 가지기 때문에 수식 (1), 수식 (2) 대신에 다음 식을 적용한다.

$$v_i^{k+1} = c_2 r_2 (g - x_i^k) + c_1 r_1 (p_i - x_i^k) + w \cdot v_i^k \quad (3)$$

$$x_i^{k+1} = v_i^k + c_3 r_3 x_i^k \quad (4)$$

4. TSP에서의 실험

4.1 유전자 알고리즘

본 연구에서는 대표적인 순서화 문제인 TSP를 대상으로 설명된 DPSO들의 성능을 분석한다. TSP에 대해 PSO의 성능 분석을 위한 Cunkas and Ozsaglam[3]의 연구에서는 그들이 제안한 DPSO와 유전자 알고리즘을 비교하였다. 본 연구에서도 대표적인 여러 DPSO들의 성능을 분석하기 위하여 유전자 알고리즘과 비교하였다. 유전자 알고리즘은 각 세대에서의 초기 염색체들에 대하여 적응값에 기초한 선택 확률을 구하고 룰렛 휠(roulette wheel) 방식에 의해 염색체들을 선택한다. 선택된 염색체들은 교배와 돌연변이 연산을 통하여 다음 세대의 염색체들을 생성한다.

TSP에서의 유전자 알고리즘 역시 DPSO와 마찬가지로 다양한 해 표현 방법이 존재한다. 본 연구에서는 DPSO에서의 순열 표현 방법과 동일한 Path representation을 이용하여 TSP의 해를 표현하였다. 패스(path) 표현방법에서는 교배 연산자인 PMX(partially-mapped)[5], OX(order)[4], CX(cycle)[11]등을 사용할 수 있다. 본 연구에서는 교배 연산자로 OX를 사용하였다. 또한, 돌연변이 연산자로는 염색체에서 임의의 두 위치를 구해 그 위치 사이의 값들을 꺼꾸로 바꾸는 반전(inversion) 연산자를 사용하였다.

4.2 파라미터 및 초기화

DPSO와 유전자 알고리즘에서 결정되어야 하는 파라미터가 존재한다. PSO에서의 파라미터들은 제안된 논문에서 인용된 값들을 가급적 사용하였다. 그외 필요한 파라미터 값들과 유전자 알고리즘에서의 파라미터들은 예비 실험을 통해 바람직하다고 판단되는 값들로 결정하였다. <표 1>은 본 연구에서 사용한 DPSO에서의 파라미터 값들을 나타낸다.

w_0와 w_k는 가중치 w의 초기값과 k번째 반복에서의 w를 의미한다. PP1, PR, PM에서는 반복에 따라 w값이 점차 감소하도록 설정되어 있는 반면, PP2와 PE에서는 반복에 상관없이 상수를 사용한다. DPSO에서의

위치 벡터와 속도 벡터의 초기화와 범위는 제안된 논문에서 사용한 방법들을 그대로 이용하였다.

유전자 알고리즘에서는 교배 확률을 0.8, 돌연변이 확률을 0.01로 정하였다. DPSO와 유전자 알고리즘 모두 객체 수는 2×노드 수(도시 수)로 하였고, 반복 수 또는 세대 수는 최대 1,000으로 한정하였다.

<표 1> DPSO 파라미터

PSO	해 표현	c	w_0	w_k
PP1	순열표현 1	$c_1 = 1.49445$ $c_2 = 1.49445$	-	$w_k = 0.5 + rand() \times 0.5$
PP2	순열표현 2	$c_1 = 1.5$ $c_2 = 2.0$	1.0	1.0
PR	실수표현	$c_1 = 2.0$ $c_2 = 2.0$	0.9	$w_k = 0.975 \times w_{k-1}$ $w_k < 0.4$ *then* $w_k = 0.4$
PM	행렬표현	-	0.9	$w_k = 0.975 \times w_{k-1}$ $w_k < 0.4$ *then* $w_k = 0.4$
PE	에지표현	$c_1 = 1.5$ $c_2 = 2.0$ $c_3 = 2.0$	0.6	0.6

5. 실험 및 분석

TSPLIB[17]에 있는 5개 벤치 마크 문제를 선택하여 DPSO들의 성능 분석을 위한 실험을 수행하였다. 결과의 신뢰성을 위해 각 문제마다 초기 난수를 바꾸어 25번을 반복하여 총 750번(5문제 × 25번 반복 × 6 방법)을 실행하였다. 실험 대상 방법들은 JAVA 프로그래밍 언어로 구현되어 Intel 2.33 GHz CPU와 2 GB의 메모리를 갖춘 PC에서 실행되었다.

<표 2>는 반복수 또는 세대수를 1,000으로 했을 때 각 방법에 의해 구한 총 거리의 평균, 표준편차, 오차율, 그리고, 실행시간을 나타낸다. 오차율은 최적해를 기준으로 구한 평균 오차를 의미하여 다음 식에 의해 구하였다.

$$\text{오차율} = \frac{\text{각 방법에 의한 총거리} - \text{최적해의 총거리}}{\text{최적해의 총거리}} \times 100$$

PM과 PE의 경우 적지 않은 실행시간이 걸렸다. PE는 대상 문제 중 가장 큰 문제인 a280 문제에서 한번의 실행에 약 17시간의 많은 시간이 걸려 한번의 실행만을 수행했고, PM의 경우 KroA200 문제에서 약 3시간 이 걸려 한번 만을 실행하였다. PM은 a280 문제에서 34시간 이상이 걸려 실행을 중지하였다.

<표 2>의 결과에 따르면 에지 표현에 기초한 DPSO인 PE의 해가 가장 우수하여 5문제에서의 평균 오차율이 최소 2%에서 최대 13%이었다. 다른 DPSO와 유전자 알고리즘이 최소 54%에서 최대 1046%의 오차율을 보인 것에 비하면 매우 큰 차이를 나타냈다고 볼 수 있다.

<표 2> 반복수/세대수가 1000일 때의 결과

문제		PP1	PP2	PR	PM	PE	GA
Bays29	평균 (오차율)	3121 (54%)	4041 (100%)	2963 (47%)	3956 (96%)	2062 (2%)	4107 (103%)
	표준편차	183	147	231	178	39	120
	시간(초)	0.340	0.825	0.226	6.806	2.408	0.104
Berlin52	평균 (오차율)	18444 (145%)	22223 (195%)	16091 (113%)	21825 (189%)	8488 (13%)	22520 (199%)
	표준편차	712	597	1484	1379	365	520
	시간(초)	1.192	3.070	1.046	55.840	20.571	0.328
kroA100	평균 (오차율)	119411 (461%)	133458 (527%)	95715 (350%)	126449 (494%)	22766 (7%)	133932 (529%)
	표준편차	2780	1565	6761	6638	600	1963
	시간(초)	4.840	14.906	6.442	647.264	258.784	1.532
kroA200	평균 (오차율)	266445 (807%)	284447 (869%)	229299 (681%)	278977 (850%)	31844 (8%)	285350 (872%)
	표준편차	4567	3568	16060	-	600	3767
	시간(초)	24.234	87.158	45.828	9014.577	3729.663	8.839
a280	평균 (오차율)	27967 (984%)	29448 (1042%)	24440 (848%)	-	2728 (5.8%)	29561 (1046%)
	표준편차	295	233	1672	-	-	268
	시간(초)	55.139	217.353	120.692	-	61565.245	22.004

<표 3> 실행시간이 동일할 때의 결과

문제		PP1	PP2	PR	PM	PE	GA
Bays29	평균 (오차율)	3246 (61%)	4203 (108%)	3045 (51%)	4399 (118%)	2126 (5.2%)	4107 (103%)
	표준편차	160	162	2117	184	70	120
Berlin52	평균 (오차율)	19090 (153%)	22544 (199%)	16712 (122%)	24258 (222%)	8763 (16%)	22520 (199%)
	표준편차	672	447	1242	551	294	520
kroA100	평균 (오차율)	121448 (471%)	134905 (534%)	103764 (388%)	142561 (570%)	25314 (19%)	133932 (529%)
	표준편차	2382	2499	9429	2617	735	1963
kroA200	평균 (오차율)	266926 (809%)	287672 (880%)	231943 (690%)	297894 (914%)	35478 (21%)	285350 (872%)
	표준편차	3401	2365	17461	4834	537	3767
a280	평균 (오차율)	28211 (994%)	29684 (1051%)	24706 (860%)	30789 (1094%)	2911 (13%)	29561 (1046%)
	표준편차	247	237	1197	351	38	268

유전자 알고리즘의 해는 가장 나빠 실험 대상이 된 모든 DPSO들이 GA에 필적하거나 우월하여 우수한 해를 생성할 수 있음을 의미한다. 그러나 실행 시간 측면에서는 반대의 결과를 가져왔다. 에지 표현에 기초한 PE가 가장 많은 계산시간을 요하였고, 유전자 알고리즘은 가장 적은 시간이 걸려 가장 큰 문제인 a280에서 PE가 17시간 걸린 반면 GA는 22초가 걸렸다. DPSO 중에서는 PP1과 PR이 대체로 적은 시간이 걸렸다. PR은 25번의 반복에 의한 해의 표준편차가 가장 커 다른 방법에 비해 불안정한 방법임을 의미한다. 반면 PE는 가장 적은 표준편차를 나타내어 가장 안정적으로 해를 생성하는 방법임을 알 수 있다.

공정한 비교를 위해 동일한 실행시간을 설정하여 해의 질을 분석하였다. <표 2>에서 GA가 걸린 시간만큼 DPSO들의 실행시간을 한정하였고, <표 3>은 이러한 결과를 나타난다. GA 열은 <표 2>의 GA 결과와 동일하다. 동일한 시간 동안을 수행한 결과도 이전의 결과와 크게 벗어나지 않는다. PE가 평균 오차율(5.2%, 21%) 범위의 우수한 해를 생성한 반면, 다른 방법들은 (51%, 1094%) 범위의 큰 오차율을 나타낸다. 결론적으로 해의 질에서 PE가 가장 좋은 해를 생성하였고, PR, PP1, GA, PP2, PM의 순에 따른다. PM은 행렬 표현에 기초하므로 많은 메모리를 필요로 할 뿐 아니라 많은 계산시간을 요하여 해의 질이 좋지 않은 결과를 초래하였다. PM을 제외한 모든 PSO들이 GA에 필적하거나 우월하여 우수한 해를 생성할 수 있음을 나타낸다. 이전의 결과와 같이 PR은 표준편차가 가장 커 다른 방법에 비해 불안정된 방법임을 재확인 시켜 준다.

유전자 알고리즘 등을 포함한 메타 휴리스틱의 응용에 대한 많은 연구에서는 보다 우수한 해를 생성하기 위하여 부분 최적화를 결합하고 있다[7, 13]. 즉, 부분 최적화를 메타 휴리스틱의 해에 적용하여 보다 우수한 해를 생성하고 있다. 서로 다른 두 알고리즘인 메타 휴리스틱과 부분 최적화를 결합하는 방법에는 다양한 방법이 있을 수 있다. 메타 휴리스틱의 어떤 해에 부분 최적화를 적용하는지, 그리고, 부분 최적화를 통한 해를 메타 휴리스틱의 해로 피드백 시키는지의 여부 등에 대한 다양한 방법이 있을 수 있다. 본 연구에서도 이러한 부분 최적화의 결합을 모색하고, 이러한 결합이 어떠한 성능을 가져오는지를 분석하였다. 본 연구에서 고려한 부분 최적화의 결합 방법은 <표 4>와 같다.

<표 4> 부분 최적화 결합 방법

구 분	결합 방법
부분 최적화 방법	2-OPT
부분 최적화 적용	각 반복 또는 각 세대에서의 해 중 가장 좋은 해
부분 최적화 해의 피드백	부분 최적화 해는 메타 휴리스틱해로 피드백 되지 않음

DPSO에서는 각 반복 마다 생성된 개체 중 가장 좋은 개체를 초기해로 하여 2-OPT를 수행하였고, 비슷하게 유전자 알고리즘에서는 각 세대 마다 생성된 개체 중 가장 좋은 해를 초기해로 하여 2-OPT를 수행하였다. 따라서 2-OPT의 실행 횟수는 DPSO의 반복수 또는 GA의 세대 수와 일치한다. 부분 최적화를 통한 해는 DPSO 또는 GA의 해로 반영되지 않도록 하였다. 이는

<표 5> 실행시간이 동일할 때 부분 최적화 결합의 결과

문제(실행시간)		PP1	PP2	PR	PM	PE	GA
Bays29 (0.206초)	평균 (오차율)	2020 (0%)	2020 (0%)	2021 (0%)	2032 (0.6%)	2029 (0.5%)	2020 (0%)
	표준편차	0	0	2	9	12	0
Berlin52 (0.952초)	평균 (오차율)	7542 (0%)	7557 (0.2%)	7576 (0.4%)	7781 (3.2%)	7938 (5.3%)	7542 (0%)
	표준편차	0	64	58	183	160	0
kroA100 (6.768초)	평균 (오차율)	21368 (0.4%)	21369 (0.4%)	21396 (0.5%)	22056 (3.6%)	21721 (2.1%)	21350 (0.3%)
	표준편차	49	84	87	377	330	43
kroA200 (54.556초)	평균 (오차율)	30186 (2.8%)	30260 (3.0%)	30166 (2.7%)	31273 (6.5%)	30519 (3.9%)	30144 (2.6%)
	표준편차	162	162	142	454	247	133
A280 (149.387초)	평균 (오차율)	2713 (5.2%)	2715 (5.3%)	2723 (5.6%)	2850 (10.5%)	2711 (5.1%)	2703 (4.8%)
	표준편차	19	22	13	36	36	18

에지 표현 방법 등과 같은 일부 DPSO에서 부분 최적화의 해를 DPSO의 해로 변환하기가 쉽지 않을 뿐 만 아니라 새로운 해에 대한 속도 벡터를 재 정의하기가 용이하지 않기 때문이다.

<표 5>는 이러한 부분 최적화를 결합하여 유전자 알고리즘에서 1000세대 동안 걸린 시간과 동일한 시간 동안을 실행한 결과를 나타낸다. 결과는 부분 최적화의 적용으로 지대한 해의 향상과 표준편차의 감소로 반복에 따른 해의 안정성을 이루었음을 나타낸다. 비교적 작은 문제인 Bays29와 Berlin52에서 PP1과 GA는 25번의 반복 모두 완벽하게 최적해를 생성하였다. 다른 방법들도 최대 오차율 5.3%의 좋은 성능을 나타내었다. 큰 문제에서도 부분 최적화의 적용이 해의 향상에 큰 기여를 하여 부분 최적화를 적용하지 않은 경우에 비해 많은 향상을 이루었음을 알 수 있다. 해의 질은 PM이 다른 방법에 비해 좋지 않음을 나타낸다. 이는 한번의 반복에 걸리는 시간이 길어 실행시간 동안 반복횟수가 적었고, 결과적으로 부분 최적화를 수행한 횟수가 적었기 때문이다. 반면, GA의 경우에는 한 세대의 실행시간이 짧아 세대수가 상대적으로 많았고, 이는 많은 횟수의 부분 최적화를 수행토록 하여 비교적 좋은 해를 생성하였다. 그러나 PM과 PE를 제외한 나머지 방법들 간에 큰 차이를 나타내지 않는다.

6. 결 론

PSO를 이산적인 순서화 문제에 적용하기 위한 다양한 방법들이 개발되었다. 본 연구에서는 5가지 DPSO 방법을 선정하여 TSP 문제들을 대상으로 이들의 성능을 비교 분석하였다. 동일한 실행시간 동안의 성능을 분석한 결과는 DPSO들이 전형적인 유전자 알고리즘에 필적하거나, 우월하였음을 나타내었다. 특히, 에지에 기초한 표현 방법은 고려된 방법 중 가장 우수한 결과를 나타내어 최적해에 어느 정도 근접하였다. 그러나, 유전자 알고리즘을 포함한 다른 DPSO 들은 최적해와는 동 떨어진 좋지 않은 해를 생성하였다.

DPSO 단독 적용보다는 부분 최적화 방법인 2-OPT를 결합하였을 때 DPSO들의 성능이 더욱 우수하여 최적해에 근사한 해들을 생성하였다. 이는 DPSO 뿐만 아니라 유전자 알고리즘에서도 동일한 결과를 가져왔다. 2-OPT와 결합된 유전자 알고리즘의 경우 큰 문제인 a280에서 149초의 적은 시간 동안을 실행한 결과는 최적해와 4.8%의 차이를 나타내었다. DPSO들은 행렬에 기초한 방법을 제외하고, 최적해와의 차이가(5.1%, 5.6%)의 범위를 나타내어 유전자 알고리즘과 약간의 차이를 나타내었다. 이는 유전자 알고리즘의 실행시간이 적어 많은 횟수의 부분 최적화를 수행한 결과이다.

이 같은 분석으로 TSP 문제에서 DPSO의 단독 적용보다는 부분 최적화를 결합한 방법이 우수한 해를 생성할 수 있다는 기존의 연구 결과를 확인할 수 있었다. 이 경우 DPSO를 통한 다양한 해를 초기해로 하여 많은 횟수의 부분 최적화를 수행하는 것이 중요하다. DPSO들은 유전자 알고리즘에 비해 다소 많은 실행시간이 소요되므로 적은 횟수의 부분 최적화로 인해 유전자 알고리즘 보다 좋은 해를 생성하기 어렵다. 따라서 DPSO의 성능을 개선하기 위해 실행시간을 단축시킬 수 있는 방안을 모색하는 것이 중요하다.

참고문헌

[1] Anghinolfi, D. and Paolucci, M.; "A new discrete particle swarm optimization approach for the single-machine total weighted tardiness scheduling problem with sequence-dependent setup times," *European Journal of Operations Research*, 193 : 73-85. 2009.

[2] Chandrasekaran, S., Ponnambalam, S. G., Suresh, R. K., and Vijayakumar, N.; "An Application of particle swarm optimization algorithm to permutation flow shop scheduling problems to minimize makespan, total flowtime and completion time variance," 2006 *IEEE International Conference on Automation Science and Engineering*, 513-518, 2006.

[3] Cunkas, M and Ozsaglam, M. A.; "A Comparative Study on Particle Swarm Optimization and Genetic Algorithms for Traveling Salesman Problems," Cybernetics and Systems : *An International Journal* 40 : 490-507, 2009.

[4] Davis, L.; "Applying Adaptive Algorithms to Epistatic Domains," Proceedings of the International Joint Conference on Artificial Intelligence, 162-164, 1985.

[5] Goldberg, D. E. and Lingle, R.; "Alleles, Loci, and the TSP," Proceedings of the First International Conference on Genetic Algorithms, Lawrence Erlbaum Associates, Hillsdale, NJ, 154-159, 1985.

[6] Hu, X., Eberhart, R. C., and Shi, Y.; "Swarm Intelligence for Permutation Optimization : A Case Study of n-Queens Problem," 2003 *IEEE Swarm Intelligence Symposium*, 243-246, 2003.

[7] Johnson, D. S. and McGeoch, L. A.; "The Traveling Salesman Problem : A Case Study in Local Optimization," Local Search in Combinatorial Optimization, Aarts, EHL and Lenstra, JK(Ed.) John Whiley and Sons, London, 215-310, 1997.

[8] Kennedy, J. and Eberhart, R. C.; "Particle Swarm Optimization," *Proceeding of the 1995 IEEE International Conference on Neural Networks*, 1942-1948, 1995.

[9] Liao, Ching-Jong, Tseng, Chao-Tang, and Luarn, Pin; "A discrete version of particle swarm optimization for flowshop scheduling problems," Computers and OR 34 : 3099-3111, 2007.

[10] Marinakis, Y. and Marinaki, M.; "A Hybrid genetic-Particle Swarm Optimization Algorithms for the Vehicle Routing Problem," *Expert Systems with Applications*, 37 : 1446-1455, 2010.

[11] Oliver, I. M., Smith, D. J., and Holland, J. R. C.; "A Study of Permutation Crossover Operators on the traveling Salesman Problem," Proceedings of the Second International Conference on Genetic Algorithms, Lawrence Erlbaum Associates, Hillsdale, NJ, 224-230, 1987.

[12] Pang W., Wang K., Zhou C., Dong L.; "Fuzzy discrete particle swarm optimization for traveling salesman problem," Proceedings of the fourth international conference on computing and information technology(CIT'04), 796-800, 2004.

[13] Potvin, J. Y.; "Genetic Algorithms for the Traveling Salesman Problem," *Annals of Operations Research*, 63 : 339-370, 1996.

[14] Shi, X. H., Liang, Y. C., Lee, H. P., Lu, C., and Wang, Q. X.; "Particle Swarm Optimization-based algorithms for TSP and generalized TSP," *Information Processing Letters*, 103 : 169-176, 2007.

[15] Tasgetiren, F., Sevkli, M., Lian, Y. C., and Gencyilmaz, G.; "Particle Swarm Optimization algorithm for single machine weighted tardiness problem," *Proceedings IEEE congress on evolutionary computation*, 1412-1419, 2004.

[16] Zhong, Wei-Liang, Zhang, Jung, and Chen, Wei-Neng; "A novel discrete particle swarm optimization to solve traveling salesman problem," *Proceedings IEEE Congress on Evolutionary Computation*, 3283-3287, 2007.

[17] TSPLIB : http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/.