유전자 재배열을 이용한 유전자 알고리즘의 성능향상

Improving the Performance of Genetic Algorithms using Gene Reordering

황 인 재*

In-Jae Hwang*

요 약

유전자 알고리즘은 공학 분야에서 필요한 여러 가지 최적화 문제에 대하여 최적에 가까운 해를 제공해주는 반복적 알고리즘으로 알려져 있다. 본 논문에서는 특정 교배방법에서 유전자의 배열순서가 적합도가 높은 스키마의 길이에 미치는 영향을 고찰하였다. 또한 이에 따른 유전자 알고리즘의 성능 변화를 두 개의 예제를 이용한 실험을 통하여 관찰하였다. 예제로 사용된 그래프 분할과 knapsack 문제를 위해 몇 가지 유전자 재배열 방법을 제시하였다. 실험결과에 따르면 유전자 재배열 방법마다 서로 다른 유전자 알고리즘 성능을 보여주었으며, 적합도가 높은 스키마의 길이를 고려한 재배열 방법이 재배열을 하지 않았을 때 보다 유전자 알고리즘의 성능을 향상시켜 주는 것을 관찰하였다. 따라서 주어진 문제에 적합한 유전자 재배열 방법을 찾는 것이 대단히 중요함을 확인하였다.

Abstract

Genetic Algorithms have been known to provide near optimal solutions for various optimization problems in engineering. In this paper, we study the effect of gene order in genetic algorithms on the defining length of the schema with high fitness values. Its effect on the performance of genetic algorithms was also analyzed through two well known problems. A few gene reordering methods were proposed for graph partitioning and knapsack problems. Experimental results showed that genetic algorithms with gene reordering could find solutions of better qualities compared to the ones without gene reordering. It is very important to find proper reordering method for a given problem to improve the performance of genetic algorithms.

Keywords : Genetic algorithm, Gene reordering, Graph partitioning, Knapsack problem, Clustering.

I. 서 론

유전자 알고리즘(Genetic Algorithm)은 다윈(Darwin)의 적자생존 이론을 기반으로 한 탐색기법이다. 적자생존이란 특정형질을 가진 개체가 자연환경에서 살아남아 자신의 형질을 자손들에게 물려줄 확률이 크다는 자연적 선택(natural selection)위에 세워진 진화이론이다. 유전자 알고리즘에서는 이러한 생물의 진화과정을 특정 종류의 문제들을 풀기위한 컴퓨터 알고리즘에 적용시키고 있다. 이를 위해 유전자 알고리즘을 적용하고자 하는 문제를 코드화하고 해의 적합도(fitness) 즉 목적 함수값에 (objective function) 따라 부모의 선택, 교배, 돌연변이 등을 적용하여 여러 세대를 거쳐 개체의 적합도가 향상되는 과정을 자연현상과 유사하게 흉내내고 있다. 이와 같은 유전자 알고리즘은 VLSI 설계, 스케줄링, 네트워크 설계 및 라우팅 등 여러 종류의 최적화 문제와[1,2,3] 기계학습에[4] 적용되어 다른 알고리즘에 비해 높은 탐색 효율을 보여주었다. 유전자 알고리즘의 장점으로는 다른 경험적 알고리즘과는 달리 탐색 중 함정에 빠지지 않고 광범위한 탐색을 수행하며 알고리즘 자체가 가진 대규모 병렬성의 활용가능성을 꼽을 수 있다.

본 논문에서는 유전자의 배열순서가 유전자 알고리즘의 성능에 어떠한 영향을 미치는가를 두 개의 예제를 통하여 실험적으로 고찰하였다. 많은 문제에서 해는 일차원 스트링으로 코드화된다. 이러한 일차원 코드에 적용되는 교배 방법 중 널리 사용되는 것은 원 포인트 혹은 투 포인트 교배이다. 위의 교배 방법에서는 부모를 한 지점 혹

*충북대학교 컴퓨터교육과
접수 일자 : 2006. 6. 2 수정 완료 : 2006. 8. 16
논문 번호 : 2006-3-10
※이 논문은 2005년도 충북대학교 학술연구연구비지원에 의하여 연구되었음.

은 두 지점에서 잘라 서로 교차되게 접합시켜 자식을 만들게 되는데 유전자의 배열순서에 따라 자식에게 물려주면 바람직한 우수한 형질의 스키마가 보존이 될 수도 있고 그렇지 않을 수도 있다. 본 논문에서는 유전자 알고리즘을 적용하기 전에 유전자 배열을 바꾸어 줌으로써 유전자 알고리즘의 성능향상을 모색한다. 유전자 재배열이 유전자 알고리즘의 성능에 미치는 영향을 측정하기 위하여 전형적인 최적화 문제인 그래프 분할과 knapsack 문제를 사용하였다.

본 논문의 구성은 다음과 같다. 다음 장에서는 스키마 이론을 이용하여 유전자 배열과 교배의 관계를 설명한다. 3장에서는 유전자 재배열이 알고리즘의 성능에 미치는 영향을 평가하기 위하여 그래프 분할 문제와 knapsack 문제에 적용시켜 실험하고 결과를 제시한다. 4장은 논문의 요약과 결론을 포함하고 있다.

2. II. 유전자 배열과 교배

본 장에서는 널리 알려진 스키마 이론을 이용하여 유전자의 배열에 따라 특정 스키마가 보존될 확률이 달라질 수 있고 그로 인하여 전체 유전자 알고리즘의 성능에 영향을 줄 수 있음을 설명한다.[5] 스키마란 주어진 문제의 부분해를 구성하는 유전자들의 집합이다. 스키마에 대한 상세한 설명에 앞서 먼저 구체적 예로서 사용된 그래프 분할 문제를 다음과 같이 정의한다.

주어진 그래프 $G=(V,E)$에서, V는 점 v_i의 집합이고, E는 간선 $e_{ij}=(v_i, v_j)$의 집합이다. 점 v_i의 weight를 a_i라 하고, 간선 e_{ij}의 weight를 c_{ij}라고 하자. 그래프가 k개만큼 분할되어야 할 때 분할된 각 집합의 용량 $A_1, A_2, \ldots, A_k$이 주어진다. 그래프 분할 문제는 아래의 조건을 만족하면서 서로 겹치지 않는 부분집합 $V_1, V_2, \ldots, V_k$를 찾는 것이다.

1. $\bigcup_{n=1}^{k} V_n = V$
2. $\forall V_n, \quad \Sigma_{\forall v_i \in V_n} a_i < A_n$
3. 동일한 부분집합 V_n 에 속하지 않는 모든 쌍의 v_i, v_j에 대하여 $C=\Sigma c_{ij}$는 최소화 되어야 한다.

위에서 C는 분할비용이 되고 분할에 의해 잘린 간선의 집합 e_{ij}를 cut set이라 한다. 본 논문에서는 k 값을 2로 제한하고 모든 점 v_i의 weight a_i를 1로 가정하였다. 또한 집합 V 안의 점의 개수를 N이라 할 때 분할된 두 집합의 용량은 $A_1 = A_2 = \frac{N}{2}$ 이다.

위와 같이 그래프 분할문제를 2-분할 문제로 제한하면 유전자 알고리즘을 적용하기위한 코드화가 대단히 간단해진다. 점의 개수가 N인 분할문제의 특정해는 $b_1, b_2, \cdots, b_N$으로 표현될 수 있고 여기서 b_i의 값은 $v_i \in A_1$ 이면 0, $v_i \in A_2$ 이면 1이다. A_1과 A_2의 용량을 $\frac{N}{2}$로 가정하였으므로 $b_1, b_2, \cdots, b_N$에서 절반은 0이고 나머지는 1이 되어야 한다. 교배연산자는 이러한 코딩에서 흔히 사용되는 원 포인트 교배를 사용할 수 있다. 원 포인트 교배에서는 부모로 선택된 두 개의 개체 $a_1, a_2, \cdots, a_N$와 $b_1, b_2, \cdots, b_N$를 특정부위에서 잘라 교차하여 연결함으로써 두 개의 자식

$a_1, a_2, \cdots, a_i, b_{i+1}, \cdots, b_N$ 과

$b_1, b_2, \cdots, b_i, a_{i+1}, \cdots, a_N$을 얻게 된다.

분할문제에서 하나의 스키마는 다음과 같은 문자열로 표현될 수 있다: $H=[1{**}0{*}1{****}]$.

여기서 "*"는 don't care를 나타내는 메타심볼이다. 스키마 H의 defining length $\delta(H)$는 "*"이 아닌 첫 번째 문자에서 "*"이 아닌 마지막 문자 사이의 거리로써 정의된다. 따라서 $\delta(H)$는 5이다. 위의 스키마 H가 나타내는 해들은 모두 $v_4 \in A_1$, $v_1, v_6 \in A_2$인 공통적 성질을 가지고 있다. 스키마의 평균 적합도 $f(H)$는 H가 나타내는 모든 해들의 평균 적합도 값으로 정의된다.

스키마는 교배에 의하여 파괴될 수 있다. H의 경우 무작위로 선택된 cut point가 첫 번째와 여섯 번째 유전자 사이의 어느 한 점이 되면 스키마가 파괴되어 자식들은 그 스키마가 나타내는 성질을 갖지 않게 될 수 있다. 스키마가 교배에 의하여 파괴될 확률은 defining length δ에 비례하여 커진다. H가 교배에 의해 파괴될 확률 $p_d(H)$는 $\frac{5}{9}$이다. 본 논문에서 제시한 유전자 재배열은 이와 같이 δ에 따른 p_d에 근거하고 있다. 예를 들어 점의 수 n이 10인 분할문제에서 v_1과 v_9 사이를 연결하는 간선의 weight가 다른 간선의 weight보다 현저히 크다고 할 때, 이 간선은 cut set에 포함되지 않는 것이 분할비용을 줄이는데 절대적으로 유리하다.

즉, $v_1, v_9 \in A_1$이거나 $v_1, v_9 \in A_2$인 것이 바람직하다. 별도의 유전자 재배열을 하지 않았을 경우 위의 성질을 갖는 스키마 H_1은 $[0{*******}0{*}]$(혹은 $[1{*******}1{*}]$)으로 표현된다. 그러나 유전자 알고리즘을 적용하기 전에 어떤 방법으로 유전자들을 재배열하여 v_9이 두 번째 유전자에 의해 표현되면 위와 같이 바람직한 성질을 갖는 스키마 H_2는 $[00{********}]$(혹은 $[11{********}]$)이 된다. H_2가 교배 후 보존될 확률

$p_s(H_2)$는 $\frac{8}{9}$이고 $p_s(H_1)$은 $\frac{1}{9}$이므로 교배에 의하여 만들어진 자식이 v_1, v_9를 같은 집합에 포함시킬 확률은 재배열 안했을 때 보다 8배 높아지게 된다. 이와 같은 일이 여러 세대를 거쳐 반복되면 유전자 알고리즘은 높은 적합도를 가진 스키마가 나타내는 해의 공간을 보다 집중적으로 탐색하여 탐색효율을 높이게 된다.

교배만을 고려할 때 어느 세대 t에 스키마 H가 나타내는 m개의 chromosome을 $m(H,t)$라 나타내면 다음세대의 chromosome 수 $m(H,t+1)$는 다음과 같이 나타내 진다.[5]

$$m(H,t+1)=m(H,t)\cdot\frac{f(H)}{\bar{f}}\cdot[1-p_c\cdot\frac{\delta(H)}{n-1}]$$

위의 식에서 $\bar{f}$는 전체 chromosome의 적합도 평균이고 p_c는 교배확률이다. 즉 적합도에 비례하여 선택확률이 커지는 것을 고려하였고, 교배의 경우 defining length가 클수록 스키마의 생존확률이 커지는 것을 볼 수 있다. 위의 식이 의미하는 바는 적합도가 높고 defining length가 짧은 스키마는 다음세대에 생존하여 유전자 알고리즘의 탐색공간에 포함될 확률이 높다는 것이다.[6] 유전자 알고리즘에서 선택은 적합도를 이용한 확률적 방법을 사용하므로 적합도가 높은 스키마가 높은 선택확률을 갖지만 특정 스키마의 적합도와 defining length는 특별한 관계가 없다.

본 논문에서는 스키마의 적합도가 높을수록 defining length가 짧아지도록 유전자를 재배열하여 유전자 알고리즘을 적용하는 전략을 제시한다. 스키마의 적합도와 defining length가 관련을 갖게 유전자를 재배열 하는 것은 매우 어렵고 문제마다 다른 방법을 요구한다. 다음 장에서는 두 개의 전형적인 최적화 문제를 통하여 유전자 재배열이 알고리즘 성능에 미치는 영향을 실험적으로 확인해 본다.

3. III. 실례를 통한 성능평가

3.1 그래프 분할문제

3.1.1 유전자 재배열

그래프 분할문제의 정의와 코드화는 3장에서 설명되었으므로 여기서는 생략한다. 이 문제에서 유전자 재배열의 목표는 c_{ij}가 높은 v_i, v_j를 나타내는 유전자를 chromosome 내에서 가깝게 두는 것이다. 따라서 다음과 같이 널리 사용되는 클러스터링 방법으로 그래프의 점 $v_1, v_2, \cdots, v_N$을 재배열 하였다.

먼저 weight가 가장 높은 간선이 연결하는 점 v_i, v_j를 하나의 점 $v_{[i,j]}$로 묶는다. 기존에 있던 v_i와 v_k 사이에 weight가 c_{ik}인 간선 e_{ik}이 존재하고 v_j와 v_k 사이에 weight가 c_{jk}인 간선 e_{jk}가 존재하면 새로운 점 $v_{[i,j]}$와 v_k를 잇는 간선의 weight는 $c_{ik}+c_{jk}$로 정의한다. 위와 같은 과정을 하나의 점만 남을 때까지 반복하고 남겨진 점 $v_{[i_1,i_2,\cdots,i_N]}$으로부터 새로운 점들의 배열 $v_{i_1}, v_{i_2}, \cdots, v_{i_N}$을 얻게 된다.

이와 같은 과정을 유사코드로 나타내면 다음과 같다.

Procedure Graph_Clustering (graph $G=(V,E)$)
while $(|V|>1)$ do
begin
Let c_{ab} be max $_{v_i,v_j\in V}$ c_{ij}
$V=V-\{v_a,v_b\}$
$V=V\cup\{v_{[a,b]}\}$
for each $k(k\neq a$ and $k\neq b)$
$c_{[a,b]k}=c_{ak}+c_{bk}$
end
Let $V=\{v_{[i_1,i_2,\cdots,i_N]}\}$ then $v_{i_1}, v_{i_2}, \cdots v_{i_N}$ is the new
order of vertices in V.
endProcedure.

3.1.2 실험결과

위에서 제안한 재배열 알고리즘을 적용하였을 때 유전자 알고리즘의 성능변화를 알아보기 위하여 그래프 분할문제를 위한 유전자 알고리즘을 구현하였다. 문제로 주어지는 그래프를 발생시키기 위해 먼저 N개의 점을 정의하고 무작위로 두 점을 골라 이미 간선으로 연결되어 있지 않은 경우 연결한다. 새로운 간선의 weight는 $w^{\min}$과 $w^{\max}$사이의 무작위한 값을 발생하여 할당하였다. 방향성이 없는 그래프의 경우 모든 쌍의 점이 연결된 그래프에서 간선의 개수가 $\frac{N(N-1)}{2}$가 된다. 실험에 사용된 그래프를 발생할 때는 최대간선의 수 $\frac{N(N-1)}{2}$에 1이하의 값을 곱하여 그래프 내 간선의 밀도를 조절하였다. 그래프 발생 과정을 유사코드로 나타내면 다음과 같다.

Procedure Graph_Generation (graph $G=(V,E)$)
Let $v_1, v_2, v_3, \cdots, v_N$ be the N vertices in V
$no\_edges=\frac{N(N-1)}{2}*density$
for $i=1$ to no_edges

select $1 \le i,\ j \le N$ at random
if $e_{ij} \notin E$ then
$E = E \cup \{e_{ij}\}$
c_{ij} = a random number chosen between
w^{min} and w^{max}
endif
endfor
endProcedure.

유전자 알고리즘의 구현에 사용된 각종 매개변수의 값은 다음과 같다. 집단의 크기는 200으로 하였고 부모선택은 가장 흔히 사용되는 룰렛 휠 방법을 사용하였다. 교배연산자는 원 포인트 교배를 사용하였다. 돌연변이 비율은 0.02%로 한정하였다. 본 논문에서 구현한 유전자 알고리즘은 steady-state 알고리즘이다. Steady-state 알고리즘에서는 각 세대에서 두 개의 부모만 선택되어 교배를 한다. 교배의 결과로 만들어진 두 개의 자식들은 집단 내에 최소 적합도를 갖는 개체 두 개를 교체한다. 실험에서는 이러한 과정을 10000 세대 반복하여 충분한 수렴이 이루어지도록 하였다. 표 1에서는 그래프의 크기 즉 점의 수 N을 변화시키면서 유전자 재배열을 적용하지 않은 경우와 같은 문제에 대하여 유전자 재배열을 했을 경우 유전자 알고리즘으로 얻어진 해의 분할비용 C를 비교하였다. 간선에 주어진 weight의 최대값과 최소값인 w^{min}과 w^{max}는 각각 1과 30 이고 그래프 내 간선의 수 즉 간선의 밀도인 *density*는 80%이다. 본 논문의 모든 실험에서는 각 경우에 100개의 문제를 발생시켜 해를 구한 후 평균값을 계산하여 표에 수록하였다. 표 1을 보면 모든 경우에서 차이가 크지는 않지만 유전자 재배열이 보다 낮은 분할비용 C를 얻게 해줌을 알 수 있다. 점의 수가 많아질수록 즉 그래프의 크기가 커질수록 유전자 재배열의 효과가 조금씩 커지는 것을 관찰할 수 있다.

표 1. 그래프 크기에 따른 분할비용 C

Table 1. Cut cost C for Different Graph Sizes

N	유전자 재배열 전	유전자 재배열 후
50	6310.2	6310.0
60	9282.0	9277.1
70	12816.3	12784.7
80	16939.2	16920.9

표 2에서는 간선의 밀도 *density*값을 변화시키면서 표 1과 같은 방법으로 성능을 비교하였다. 그래프 내 점의 수는 60개로 고정하였고 w^{min}과 w^{max}는 각각 1과 30 이다. 이 실험에서도 유전자 재배열후 유전자 알고리즘을 적용했을 때 보다 낮은 분할비용 C를 얻을 수 있었다. 간선밀도가 커질수록 유전자 재배열의 효과가 조금씩 커지는 것을 관찰할 수 있다.

표 2. 간선의 밀도에 따른 분할비용 C

Table 2. Cut cost C for Different Edge Densities

density(%)	유전자 재배열 전	유전자 재배열 후
60	6644.59	6642.71
70	7933.92	7929.57
80	9258.93	9249.67
90	10619.29	10590.79

표 3에서는 간선에 주어진 weight 값의 범위를 변화시키면서 위와 같은 방법으로 성능을 비교하였다. 그래프 내 점의 수는 60개로 고정하였고 *density*는 80%이다. 이 실험에서도 위의 두 경우와 유사한 결과를 관찰할 수 있다.

표 3. Weight 값의 범위에 따른 분할비용 C

Table 3. Cut cost C for Different Ranges of Edge Weights

w^{min} - w^{max}	유전자 재배열 전	유전자 재배열 후
1 - 10	2859.27	2851.74
1 - 20	6085.7	6065.83
1 - 30	9287.4	9274.95
1 - 40	12447.08	12411.09
1 - 50	15658.42	15602.52

위의 실험에서 보면 모든 경우에 유전자 재배열이 유전자 알고리즘의 성능을 향상시키는데 도움이 된다는 것을 알 수 있다. 그러나 성능향상이 그다지 크지 않음도 보여주고 있다. 그 이유로는 실험에 사용된 그래프를 발생할 때 무작위로 선택된 두 점 사이를 간선으로 연결하고 무작위로 선택된 weight 값을 줌으로써 간선과 더불어 간선의 weight 값이 그래프 전체에 걸쳐 고르게 분포하기 때문이라 생각된다. 따라서 상대적으로 유전자 알고리즘에서 유전자의 순서가 덜 중요하게 된다. 아래의 실험에서는 그래프의 점들을 5개의 그룹으로 나누고 두 점을 간선으로 연결할 때 두 점이 동일그룹에 속하지 않으면 1과 30 사이의 값을 weight로 주고 동일그룹에 속하면 보다 더 큰 값을 weight로써 할당하였다. 이렇게 함으로써 weight 값이 고르지 않게 분포된 그래프를 얻게 된다. 대부분의 실제 응용에서 볼 수 있는 그래프에서는 모듈간의 연결이 특정모듈들에 편중된 경향이 있기 때문에 이렇게 얻어진 그래프가 현실을 더 충실하게 반영할 수 있다.

표 4에서는 그룹 내 간선의 weight 값을 변화시키면서 유전자 재배열의 효과를 측정하였다. 그래프 내 점의

수는 60개로 고정하였고 간선밀도 *density*는 80%이다. 이 실험의 결과를 보면 유전자 재배열이 유전자 알고리즘의 성능향상에 훨씬 더 큰 영향을 미치는 것을 볼 수 있다. 각 경우 두 값의 차이가 위의 세 실험에서 보다 크다는 것을 확실하게 알 수 있다. 그룹 내 간선에 주어진 weight가 커질수록 유전자 재배열의 효과가 조금씩 커지는 것도 관찰할 수 있다.

표 4. 그룹 내 Weight 값의 범위에 따른 분할비용 C

Table 4. Cut cost C for Different Ranges of Intra-group Edge Weights

그룹 내 edge weight 범위 (w^{min} - w^{max})	유전자 재배열 전	유전자 재배열 후
20 - 40	11405.97	11285.42
30 - 50	12033.92	11793.98
40 - 60	12566.5	12083.08
50 - 70	13140.05	12597.21
60 - 80	13761.76	13123.23

3.2 Knapsack 문제

3.2.1 문제의 정의와 유전자 재배열

Knapsack 문제는 전형적인 NP-hard 문제들 중 하나로써 다음과 같이 정의된다.[7] N개의 물건 $item_1, item_2, \cdots, item_N$과 용량이 M으로 한정된 knapsack이 주어진다. i번째 물건 $item_i$은 무게가 w_i이고 가치가 p_i이다. 이같이 주어진 물건들과 knapsack을 이용하여 $\Sigma_{i=1}^{N} w_i x_i \leq M$을 만족하면서 $\Sigma_{i=1}^{N} p_i x_i$가 최대가 되도록 하는 $x_1, x_2, \cdots, x_N$의 값을 구하는 것이 knapsack 문제의 목적이다. 여기서 x_i가 소수를 취할 수 있을 경우 fractional knapsack 문제라 부르고 x_i값이 0 혹은 1로 제한될 경우 0-1 knapsack 문제라 부른다. x_i는 knapsack에 담는 $item_i$의 수를 나타낸다. x_i가 소수를 취할 수 있다는 것은 $item_i$를 나누어 일부만 knapsack에 담을 수 있음을 의미한다. Fractional knapsack 문제는 greedy 알고리즘을 이용하여 N과 비례하는 시간내에 해를 구하는 것이 가능하다.

0-1 knapsack 문제의 유전자 알고리즘 코드화는 그래프 분할 문제의 경우와 비슷하게 이루어질 수 있다. 즉 물건의 개수가 N인 knapsack 문제의 특정해는 $b_1, b_2, \cdots, b_N$으로 표현될 수 있고 여기서 b_i의 값은 $x_i = 0$ 이면 0, $x_i = 1$ 이면 1이다. 교배연산자는 그래프 분할의 경우와 마찬가지로 원 포인트 교배를 사용하였다. 유전자 재배열을 위하여 다음과 같은 세 가지 방법으로 물건들을 재배열 한 후 코드화 하였다.

1. O_1 : $item_1, item_2, \cdots, item_N$을 $\frac{p_i}{w_i}$ 값을 이용하여 내림차순으로 정렬
2. O_2 : $item_1, item_2, \cdots, item_N$을 p_i값을 이용하여 내림차순으로 정렬
3. O_3 : $item_1, item_2, \cdots, item_N$을 w_i값을 이용하여 내림차순으로 정렬

3.2.2 실험결과

실험을 위한 knapsack 문제의 발생을 위하여 p^{min}과 p^{max}사이의 무작위한 값을 발생하여 p_i값으로 할당하였고 w^{min}과 w^{max}사이의 무작위한 값을 발생하여 w_i값으로 할당하였다. 유전자 알고리즘의 구현에 사용된 각종 매개변수의 값은 그래프 문제의 경우와 동일하다. Knapsack 문제 실험에서도 각 경우에 100개의 문제를 발생시켜 knapsack에 담아지는 물건들의 가치의 총합 P를 구한 후 평균값을 계산하여 표에 수록하였다. 또한 여기서도 10000 세대를 반복하여 유전자알고리즘이 최적해에 충분히 수렴하도록 하였다. 표 5는 knapsack 문제의 크기 즉 물건의 수 N의 변화에 따른 세 가지 유전자 재배열 방법의 효과를 보여주고 있다. p^{min}과 p^{max} 그리고 w^{min}과 w^{max}의 범위는 모두 1에서 20이다. 표에 수록된 값은 knapsack에 담겨진 물건들의 총 가치이므로 그래프 문제와는 반대로 값이 클수록 바람직한 해임을 의미한다. 결과를 보면 O_1이 가장 좋고 O_3가 가장 좋지 않은 해를 제공함을 알 수 있다. 특히 O_1에 의한 유전자 재배열은 하지 않았을 때 보다 월등히 좋은 해를 제공하고 O_3에 의한 유전자 재배열은 오히려 더 나쁜 결과를 보여주고 있다. 이로부터 문제에 적합한 유전자 재배열 방법을 찾는 것은 유전자 알고리즘의 성능을 향상시키는데 대단히 중요함을 알 수 있다.

표 5. 물건의 수에 따른 총합 P

Table 5. Total Profit P for Different numbers of Items

물건 수	유전자 재배열 전	O_1	O_2	O_3
50	371.97	405.66	399.65	361.01
60	439.09	483.67	474.48	423.3
70	502	556.52	545.93	479.57
80	557.59	624.37	613.74	535.03

표 6은 knapsack 물건의 가치범위 변화에 따른 세 가지 유전자 재배열 방법의 효과를 보여주고 있다. 물건

의 수는 60개이고 w^{min}과 w^{max}는 각각 1과 20 이다. 이 실험에서도 모든 경우에서 위의 실험과 유사한 결과를 보여주고 있다.

표 6. 물건의 가치범위에 따른 총합 P

Table 6. Total Profit P for Different Ranges of Profit Values

p^{min} - p^{max}	유전자 재배열 전	O_1	O_2	O_3
10 - 30	819.61	884.87	857.91	787.9
20 - 40	1204.65	1282.8	1232.62	1151.51
30 - 50	1596.64	1701.56	1625.24	1531.14
40 - 60	1976.44	2119.55	2011.77	1890.71

IV. 결 론

본 논문에서는 유전자의 배열순서가 유전자 알고리즘의 성능에 어떠한 영향을 미치는가를 두 개의 예제를 통하여 실험적으로 고찰하였다. 실험 결과에 따르면 유전자 알고리즘을 적용하기 전에 유전자를 특정 방법으로 재배열함으로써 유전자 알고리즘의 성능을 향상시키는 것이 가능하다. 그래프 분할문제에서 간선의 weight가 고르지 않게 분포되었을 경우 유전자 재배열이 유전자 알고리즘의 성능을 더욱 높여주는 것을 볼 때 유전자 재배열은 재배열에 사용된 매개변수 값이 문제에 어떻게 사용되었는가에 따라 더욱 중요할 수 있다. 두 예제에서 유전자 재배열을 위하여 사용한 클러스터링과 정렬(sorting)은 반복적인 유전자 알고리즘에 비교하면 무시할 수 있을 만큼 아주 작은 수행시간을 필요로 한다. 이와 같이 작은 비용으로 유전자 알고리즘의 성능을 높일 수 있는 것은 충분히 의미가 있다. 또한 본문에서 언급하지는 않았지만 유전자 재배열을 했을 경우 훨씬 더 작은 수의 세대를 거친 후, 유전자 재배열 없이 많은 수의 세대를 거쳤을 때와 비슷한 수준의 해가 얻어지는 것을 많은 경우의 실험에서 관찰하였다. 이는 유전자 재배열이 유전자 알고리즘의 수렴속도에 영향을 줄 수 있다는 것을 의미한다. Knapsack 문제에서 관찰할 수 있듯이 유전자 알고리즘은 유전자 재배열 방법에 따라 다른 성능을 보여준다. 따라서 주어진 문제에 적합한 유전자 재배열 방법을 찾는 것이 대단히 중요하다.

참고문헌

[1] I. J. Ramirez-Rosado, J. L. Bernal-Agustin, "Genetic Algorithms Applied to the Design of Large Power Distribution Systems," IEEE Transactions on Power Systems, Vol. 13, No. 2, May. 1998, pp. 696-703.

[2] P. Mazumder, E. M. Rudnick, "*Genetic Algorithms for VLSI Design, Layout & Test Automation,*" Prentice Hall, Inc., 1999.

[3] S. M. Sait, H. Youssef, "*Iterative Computer Algorithms with Applications in Engineering,*" IEEE Computer Society, 1999.

[4] D. E. Goldberg, "*Genetic Algorithms in Search Optimization & Machine Learning,*" Addition Wesley Publisher Company, Inc., 1989.

[5] S. M. Sait, H. Youssef, "Iterative Computer Algorithms with Applications in Engineering: Solving Combinatorial Optimization Problems," Wiley-IEEE Computer Society Press, 2000.

[6] C. Reeves, editor, "Modern Heuristic Techniques for Combinatorial Problems," McGraw-Hill Book Co., 1995.

[7] M. R. Garey, D. S. Johnson, "*Computers and intractability,*" W. H. Freeman and Company, San Francisco, CA, 1983.

황 인 재(In-Jae Hwang)

1986년 충북대학교 컴퓨터공학과 공학사

1991년 Univ. of Florida 전산학과 공학석사

1994년 Univ. of Florida 전산학과 공학박사

1986년 ~ 1987년 한국전자통신연구원 연구원

1995년 ~ 현재 충북대학교 컴퓨터교육과 교수

관심분야 : 병렬 및 분산처리, 알고리즘