콤팩트 엘리트 개미 최적화

조진선° 장형수
서강대학교 컴퓨터공학과
{jeyescho,hschang}@sogang.ac.kr

Compact elitist Ant Optimization

JinSun Cho° , Hyeongsoo Chang
Department of Compute Science and Engineering, Sogang University

요 약

본 논문에서는 개미 집단 최적화(Ant Colony Optimization, ACO)의 시간적 공간적 효율성을 향상시키기 위해 ACO에 엘리트 콤팩트 유전 알고리즘(Elitist compact Genetic Algorithms, elitist cGAs)의 아이디어를 적용한 콤팩트 개미 최적화(Compact elitist Ant Optimization, CAO)를 제안한다. CAO는 elitist cGAs에서 각 세대마다 염색체의 수를 둘로 고정하고 우월한 염색체를 유지하여 최적의 해를 찾는 방식을 적용하여 개미의 수를 하나로 고정하고 전이 확률식과 페로몬 갱신 규칙을 변형하고 특정 문제에 적용할 수 있는 타부 규칙을 추가한 알고리즘이다. 이 알고리즘의 공간 효율성이 ACO보다 좋다는 것을 증명하고 스테이너 트리 문제(Steiner Tree Problem)에 적용하여 제안된 알고리즘의 시간 효율성이 ACO보다 좋다는 것을 보인다.

1. 서 론

조합 최적화 문제인 순회 판매원 문제(Traveling Salesman Problems, TSP)를 풀기 위해 Dorigo와 Maniezzo에 의해 처음 제안된 메타 휴리스틱 기법인 개미 집단 최적화(Ant Colony Optimization, ACO)[1]는 실제 개미들이 집에서 먹이까지 가장 짧은 경로를 찾는 능력을 모방하여 만든 알고리즘이다.

기존의 개미 집단 최적화 알고리즘은 그 성능이 개미의 수와 페로몬의 증발 정도에 영향을 받는다. 하지만 개미의 수를 증가시키거나 페로몬의 증발 정도를 작게 하면 성능은 좋아지더라도 계산 비용이나 메모리 요구량이 증가한다. 이러한 단점을 극복하기 위해서 본 논문에서는 Ahn과 Ramakrishna에 의해 제안된 엘리트 콤팩트 유전 알고리즘(Elitist compact Genetic Algorithms, elitist cGAs)[2]의 아이디어를 ACO에 적용하여 새로운 알고리즘 콤팩트 엘리트 개미 최적화(Compact elitist Ant Optimization, CAO)를 제안한다.

유전 알고리즘(Genetic Alogorithm, GA)[3] 역시 ACO와 비슷한 단점을 가지고 있다. ACO에서 최적의 해를 찾는데 좋은 성능을 위해서 개미 수를 증가시키거나 페로몬의 증발 정도를 작게 하는 것과 비슷하게 GA는 최적의 해를 찾는데 좋은 성능을 위해서 개체군(population)의 크기를 충분히 크게 한다. 하지만 ACO와 마찬가지로 GA 역시 개체군의 수를 증가시키면 계산 비용과 메모리 요구량이 증가한다. GA의 이러한 단점을 보안하기 위해 Compact Genetic Algorithms(cGAs)[4]는 염색체의 길이를 n으로 하면, 염색체를 $\{0,1\}^n$으로 표현하고, GA에서처럼 해들의 집합으로 개체군을 형성하는 것이 아니라 각 유전자가 1 값을 가질 확률을 유지하고 이 확률을 통해 두 개의 염색체를 생성하여 더 우월한 염색체의 성향을 따르는 방식으로 해답을 찾는다. 이러한 방식으로 cGAs는 GA보다 계산 비용과 메모리 요구량을 감소시킬 수 있지만 비용을 제외한 성능은 GA와 비교해 크게 떨어진다. 이러한 단점을 보안하기 위해 elitist cGAs가 제안되었다. 이 알고리즘은 cGAs에서 매 세대마다 두 개의 염색체를 생성하는 것과 달리 이전 세대에 우월했던 염색체와 새로 생성한 하나의 염색체를 비교하는 방식을 통해 최적의 해답을 찾도록 한다. 이 알고리즘은 우월한 염색체가 항상 살아남아 결론적으로 찾은 최적의 해답을 놓치지 않는 방식으로 cGAs의 단점을 보안하면서 기존의 GA의 단점인 계산 비용과 메모리 요구량도 감소시킬 수 있었다.

기존의 ACO에 이러한 elitist cGAs의 아이디어를 적용하여 ACO의 단점을 보안하는 알고리즘 CAO는 elitist cGAs에서 염색체의 수를 세대마다 둘로 고정하는 것을 적용하여 개미의 수를 하나로 고정하고 우월한 염색체를 다음 세대로 유지하여 새로 생성되는 염색체와 비교한 아이디어를 적용하여 하나의 개미에 알맞게 전이 확률식과 페로몬 갱신 규칙을 변형하고 마지막으로 특정한 문제가 주어졌을 경우 적용할 수 있는 "타부 규칙"을 추가하여 만든 알고리즘이다. CAO의 효율성을 증명하기 위해서 공간비용 측면에서 더 효율적이라는 것을 증명하고 스테이너 트리 문제(Steiner Tree Problem, STP)에 적용하고, 그 성능에 대해 기존의 ACO 방법과 비교하여 시간비용측면에서 더 나은 결과가 나온다는 것을 실험을 통하여 보인다.

2. Compact elitist Ant Optimization

2.1 Background

X가 가능한 해의 유한 집합이고 $f: X \rightarrow R^+$는 양의 목적 함수(objective function)일 때,

$$\min_{x \in X} f(x)$$

인 최적화 문제를 생각해보자. 문제의 목표는 $\min_{x \in X} f(x)$ 값을 얻게 하는 최적의 해 $x^* \in X$를 찾는 것이다.

ACO[5]는 조합 최적화 문제를 해결하기 위해 최근에 제안된 메타 휴리스틱 탐색 방법이다. ACO에서는 x^*를 찾기 위해 주어진 문제를 construction graph $G(C, \varphi)$로 나타내는데, $C(V, A)$는 노드의 집합 V와 간선의 집합 A를 요소로 가지는 directed

graph이고 φ는 주어진 그래프 G에서 찾은 경로를 $x \in X$로 대응시켜주는 함수이다. 개미의 수를 S라고 하면, ACO에서는 S개의 개미가 그래프 G위를 이동하면서 각 경로에 페로몬(pheromone)을 분비하고 그 이후에 지나가는 개미들이 그 경로에 있는 페로몬 정보를 이용해 다음 경로를 선택하는 원리를 휴리스틱 탐색에 적용시킨 기법이다.

Iteration m에서 개미 s가 노드 i에 도착하고 i와 연결된 모든 노드 r에 대해 $(i,r) \in A$에 있는 페로몬 정보를 이용해 다음 노드를 선택할 때 이미 지난 노드들의 sequence를 $u^s = (u_0^s, \cdots, u_{t-1}^s)$라고 하면 $j \notin u^s, (i,j) \in A$경우 다음 노드로 노드 j가 선택될 확률은

$$p_{ij}(m,u) = \frac{[\tau_{ij}(m)]^{\alpha} \times [\eta_{ij}(u^s)]^{\beta}}{\sum_{r \notin u^s,(i,r)\in A} [\tau_{ir}(m)]^{\alpha} \times [\eta_{ir}(u^s)]^{\beta}}$$

이고, $j \in u^s$이거나 $(i,j) \notin A$인 경우 다음 노드로 노드 j가 선택될 확률은 0이다. 이와 같이 노드 현재 노드 i에서 노드 j로의 전이 확률(transition probability)은 iteration m에서 간선 (i,j)의 페로몬 값 $\tau_{ij}(m)$과 desirability values $\eta_{ij}(u^s)$에 의해 결정된다[5].

페로몬 값은 ACO의 중요한 요소 중 하나로 초기 페로몬의 값은 $\tau_{ij}(1) = \frac{1}{|A|}$로 동일하게 주어진다. A_s를 현재 iteration m까지 개미 s가 찾은 경로라고 하고 f_s를 그에 대응하는 해의 목적 함수 값(objective function value)이라 하고 ϕ는 증가하지 않는(nonincreasing) 함수일 때, 각각의 간선 (i,j)에 대해

$$C = \sum_{(i,j)\in A} \sum_{s=1}^{S} \Delta\tau_{ij}^{s}, where\ \Delta\tau_{ij}^{(s)} = \begin{cases} \phi(f_s), & (i,j) \in A_s, \\ 0, & otherwise, \end{cases}$$

로 한다. 여기서 $C=0$일 경우, 페로몬은 아래 식과 같이 다음 iteration의 페로몬 값을 현재 iteration에서의 페로몬 값으로 한다.

$$\tau_{ij}(m+1) = \tau_{ij}(m)$$

하지만 $C>0$일 경우, 페로몬은 다음 아래의 규칙에 따라 변경된다.

$$\tau_{ij}(m+1) = (1-\rho) \times \tau_{ij}(m) + \rho\Delta\tau_{ij},$$
$$where\ \Delta\tau_{kl} = \frac{1}{C}\sum_{s=1}^{S} \Delta\tau_{kl}^{(s)}.$$

$\rho \in (0,1)$는 페로몬의 증발정도를 나타내는 파라미터로 이전의 페로몬이 증발되는 정도와 새로 할당되는 페로몬의 양을 결정하는 파라미터이다.

매 iteration마다 페로몬이 위와 같이 갱신되어 개미들은 각 노드에서 이전 iteration과는 다른 확률로 이동을 하게 된다. 그리고 충분한 iteration 이후엔 최적의 경로에 속하는 간선의 페로몬의 양이 다른 간선에 비해 월등히 많아 결국 최적의 경로를 찾을 확률이 1로 수렴한다[5].

ACO는 x^*가 유일하고 개미 수를 제외한 다른 파라미터가 고정되어 있는 경우 $\epsilon > 0$에 대해 개미수가 충분히 클 때 어떤 iteration $m_0(\epsilon)$ 이후에 x^*를 찾을 확률이 $1-\epsilon$ 이상이 되는 것을 보장한다[5]. 혹은 x^*가 유일하고 ρ가 충분히 0에 가까운 값을 가질 때 개미 수를 충분히 크게 한 것과 마찬가지로 iteration $m_1(\epsilon)$ 이후에 x^*를 찾을 확률이 $1-\epsilon$ 이상이 되는 것을 보장한다[5]. 하지만 개미의 수를 충분히 크게 하면 계산 비용과 요구되는 메모리가 증가하고 ρ 값을 0에 가까운 값으로 페로몬의 증발정도가 작아 느리게 수렴하게 되므로 최적의 해 x^*로 수렴하기 위해서는 계산 비용이 증가한다. 이러한 이유로 실시간으로 의사 결정을 해야 하는 에이전트나 작은 메모리를 가지고 있는 에이전트의 경우엔 ACO를 적용하기가 힘들다.

이러한 단점은 GA에서도 비슷하게 발견된다. GA는 자연계에 있는 생물의 진화과정에 있어서, 개체군(population) 중에서 환경에 대한 적합도(fitness)가 높은 개체가 높은 확률로 살아남아 재생(reproduction)할 수 있게 되는 적자생존의 원칙에 입각한 알고리즘이다. GA에서 중요한 이슈가 되고 있는 개체군의 크기(population size)를 고려해 보면, 개체군의 크기가 클수록 GA는 일반적으로 더 좋은 해를 주지만, 그와 비례해서 계산 비용과 메모리 요구량 또한 증가한다[2].

이러한 단점을 보안하면서 기존의 성능을 유지하기 위해서 제안된 알고리즘이 elitist cGAs[2]이다. Elitist cGAs는 persistent elitist cGAs와 nonpersistent elitist cGAs로 나뉘는데 본 논문에서는 nonpersistent elitist cGAs의 특징을 ACO에 적용하였다. Elitist cGAs는 GA처럼 여러 해들의 집합인 개체군을 유지하는 것이 아니라 cGAs와 마찬가지로 i번째 유전자가 1 값을 가질 확률을 i번째 값으로 가지는 확률 벡터를 통해 생성하는 두 개의 염색체만을 각 세대마다 유지한다. 첫 세대에서는 이 벡터의 각 요소의 값은 0.5로 하여 랜덤하게 유전자가 0 또는 1의 값을 가지도록 하여 두 개의 염색체를 생성하고 두 번째 세대부터는 이전 세대에 우월한 염색체와 확률 벡터의 i번째 확률을 이용해 생성된 염색체, 이렇게 두 개의 염색체를 유지한다. 각 세대마다 이 두 개의 염색체를 비교해서 우월한 염색체와 열등한 염색체로 나누고 두 염색체에서 각기 다른 값을 가지는 유전자에 대해 우월한 염색체의 유전자가 1의 값을 가진다면 그 유전자가 1의 값을 가질 확률을 증가시키고 0의 값을 가진다면 확률을 감소시킨다. 그리고 확률 벡터의 모든 값들이 0 또는 1의 값을 가지면, 그 확률 벡터에 의해 생성되는 염색체를 알고리즘의 결과로 하고, 이 염색체에 대응하는 해를 주어진 문제의 해로 한다. 이 알고리즘은 x^*를 찾으면서 시간 비용과 메모리 요구량을 감소시킨다는 것이 실험적으로 증명되었다[7].

ACO에 이러한 elitist cGAs의 아이디어를 적용하여 계산 비용과 메모리 요구량을 감소시키고 성능을 유지하는 알고리즘을 다음 장에서 논한다.

2.2 알고리즘

앞서 설명한 것과 동일하게 주어진 문제를 construction graph $G(C,\varphi)$로 나타낼 때, ACO는 S개의 개미가 주어진 그래프 C에서 이동하면서 경로를 찾는 알고리즘이지만CAO는 elitist cGAs에서 각 세대마다 염색체의 수를 둘로 고정하는 것과 같이 하나의 개미만이 경로를 찾는다. S개의 개미를 이용하여 경로를 찾는 ACO의 경우에는 C^1, C^2, C^3이 각각 상수일 때, 주어진 그래프 $G(V,A)$의 정보와 페로몬 값을 저장하는데 $2|V|^2 + C^1$의 메모리가 필요하고 S개 개미의 경로를 저장하기 위해서 $S|V|^2 + C^2$의 메모리가 필요하므로 공간 복잡도가 $O(S|V|^2)$이다. 하지만 CAO의 경우에는 그래프이 정보와 페로몬 값을 저장할 때는 동일한 메모리가 요구되고 하나의 개미의 경로와 이전까지 찾은 경로 중에 대응하는 해의 목적 함수 값이 가장 작은 경로만 저장하면 되기 때문에 $2|V|^2 + C^3$의 메모리가 필요하므로 공간 복잡도가 $O(|V|^2)$이 된다. 그러므로 CAO가 ACO보다 공간적으로 효율적이다.

CAO가 하나의 개미만으로 최적의 해 x^*를 찾기 위해 ACO의 전이 확률 식과 페로몬 갱신 규칙을 다음과 같이 변경한다.

CAO에서도 iteration m에서 개미가 노드 i에 도착하면 i와 연결된 모든 노드 r에 대해 $(i,r)\in A$에 있는 페로몬 정보를 이용해 다음 노드를 선택한다. ACO와 마찬가지로 CAO도 개미가 이미 탐색한 노드를 다시 탐색하는 것을 막기 위하여 이미 지난 노드들의 정보를 저장하고 있어야 하는데 CAO에서는 하나의 개미만이 경로를 탐색하기 때문에 이미 지난 노드들의 sequence를 $u^1=(u_0^1,\cdots,u_{t-1}^1)$라고 표현할 수 있다. 이 때 CAO에서 $j\notin u^1$, $(i,j)\in A$인 경우 다음 노드로 노드 j가 선택될 확률은

$$p_{ij}(m,u^1)=\frac{\tau_{ij}(m)}{\sum_{r\notin u^1,(i,r)\in A}\tau_{ir}(m)} \qquad (1)$$

이고 $j\in u^1$이거나 $(i,j)\notin A$인 경우 다음 노드로 노드 j가 선택될 확률은 0이다. ACO와는 달리 CAO에서 전이 확률은 오직 페로몬 정보에 의해서만 결정된다.

페로몬을 변경하는 페로몬 갱신 규칙(pheromone update rule)에 대해 살펴보면, ACO에서 각 간선 (i,j)의 초기 페로몬 값은 $\tau_{ij}(1)=\frac{1}{|A|}$로 동일하게 하는 것과 달리 CAO에서는 ACO에서 전이 확률을 결정하는 요소 중에 하나였던 간선 (i,j)의 desirability values $\eta_{ij}(u^1)$을 이용하여 $\tau_{ij}(1)=\frac{C}{\eta_{ij}(u^1)}$로 값을 결정한다. 여기서 초기 페로몬 값을 결정할 때 개미는 탐색을 하기 이전이기 때문에 $\eta_{ij}(u^1)$는 η_{ij}로 한다.

Elitist cGAs에서는 두 개의 염색체 중에서 서로 다른 값을 가진 유전자에 대해서만 우월한 염색체의 유전자 값이 1일 경우 확률 값을 증가시키고 0일 경우 확률 값을 감소시킨다. 이 아이디어를 적용해서 CAO에서는 안 좋은 경로를 찾으면, 최적의 경로에 속하지 않은 간선에 대해서만 페로몬 값을 감소시킨다.

이전 iteration까지 찾은 경로 중에서 그에 대응하는 해의 최소의 목적 함수 값을 가지는 경로를 A^*라고 하면, f^*는 그 해의 목적 함수 값이라고 하자. A_1을 현재 찾은 경로라고 하고 f_1을 그에 대응하는 해의 목적 함수 값이라고 하면 매 iteration마다 페로몬의 양은 다음 아래의 규칙에 따라 변경된다.

$$\tau_{ij}(m+1)=\begin{cases}(1+\rho_m)\times\tau_{ij}(m), & f_1<f^*\\ (1-\rho_m)\times\tau_{ij}(m), & f_1>f^* \text{ and } (i,j)\in A_1 \text{ and } (i,j)\notin A^*\\ \tau_{ij}(m), & otherwise\end{cases} \qquad (2)$$

$\rho_m\in(0,1)$는 페로몬의 증발정도를 나타내는 파라미터로 m_1은 일정 iteration 구간이고 C는 상수 값일 때,

$$\rho_{m+1}=\begin{cases}\rho_m+\frac{C}{m} & \text{if } m\%m_1=0\\ \rho_m & otherwise\end{cases},\ where\ \rho_0=0.0$$

위 식과 같은 규칙을 따른다. 이 규칙에 따라 ρ_m은 iteration이 커질수록 값이 커진다. 알고리즘 수행 시 초기 개미의 탐색은 $\rho_0=0.0$고 초기 페로몬의 값이 간선의 desirability value의 역수이기 때문에 탐욕적(greedy) 탐색을 한다. 하지만 일정 구간 m_1마다 페로몬의 증발량이 증가하면서 개미는 과거의 경험을 고려해서 탐색을 하게 된다. 처음엔 페로몬의 변화량이 작기 때문에 충분한 탐색을 하고 시간이 지나면서 매 iteration 사이의 페로몬 변화량이 커진다. 하지만 이 규칙으로만 페로몬을 변화시킬 경우 빠른 속도로 페로몬을 강화하고 다양화는 없기 때문에 지역해에 빠질 가능성이 크다. 이러한 단점을 보안하기 위해 elitist cGAs에서 일정 세대마다 우월한 염색체를 랜덤하게 생성시킨 염색체로 대체 했던 아이디어를 적용하여 일정 주기로 정해진 구간동안 초기 페로몬을 유지시켜 준다.

CAO의 전체적인 알고리즘은 다음과 같다.

V, A(input): V은 노드의 집합, A는 간선의 집합
MAX_m(input): 종료 iteration
A^*: 찾은 경로 중 최적의 경로
A_1: 현재 찾은 경로
f^*: A^*에 대응하는 목적 함수 값
f_1: A_1에 대응하는 목적 함수 값
u^1: 이미 지난 노드 정보
ρ_m: 증발량을 나타내는 파라미터
τ_{ij}: 노드 (i,j)의 페로몬 값

Step1:
/* 알고리즘 수행 전, 변수 초기화 */
$f^*=MAX\_f$
$m=0$
$\rho_m=0.0$
For all $(i,j)\in A$,
if $(i!=j)$ then $\tau_{ij}(1)=\frac{C}{\eta_{ij}}$
else $\tau_{ij}(1)=0$

Step2:
if $(m==MAX_m)$ goto step6
initialize u, A_1, f_1
/* 페로몬 증발량 업데이트 */
if $(m\%m_1==0)$ then $\rho_{m+1}=\rho_m+\frac{C}{m}$
else $\rho_{m+1}=\rho_m$

Step3:
/* 경로 찾기 */
i=srart node
while ($\varphi(A_s)$ is feasible solution){
j= next node
when $p_{ij}(m,u^1)=\frac{\tau_{ij}(m)}{\sum_{r\notin u^1,(i,r)\in A}\tau_{ir}(m)}$
/* 이제까지 찾은 경로에 간선 (i,j) 추가 */
A_s = add(A_s,(i,j))
/* 방문한 노드의 sequence에 j 추가 */
u^1 = add(u^1, j)
$i=j$

}

Step4:

/* 페로몬 업데이트 */

if $(f_1 < f^*)$

then $\tau_{ij}(m+1) = (1+\rho_m) \times \tau_{ij}(m)$, $A^* = A_1$

else if $(f_1 > f^* \text{ and } (i,j) \in A_1 \text{ and } (i,j) \notin A^*)$

then $\tau_{ij}(m+1) = (1-\rho_m) \times \tau_{ij}(m)$

else $\tau_{ij}(m+1) = \tau_{ij}(m)$

Step5:

$m = m+1$

goto step2

Step6:

return A^*

추가적으로 찾은 경로의 대응하는 해의 목적 함수 값이 경로의 간선들의 weight 값의 합으로 표현되는 경우, CAO에 "타부 규칙"을 추가한다. 타부 규칙은 모든 간선 (i,j)에 대해 (i,j)의 weight 값이 지금까지 찾은 경로 중에 대응하는 목적 함수 값이 가장 작은 값보다 클 경우, 간선 (i,j)의 페로몬 값을 0으로 하는 것을 의미한다. 이 타부 규칙은 주어진 그래프 $C(V,A)$에서 불필요한 간선이나 노드를 제거하는 기능을 한다. 기존의 ACO에서는 특정한 간선의 페로몬 값이 0에 가까운 값으로 수렴하기 전까지 계속 적으로 선택될 수 있다. 하지만 이 규칙을 이용하여 불필요하다고 판단된 간선을 A에서 제외시킬 경우 개미는 필요 없는 경로를 찾지 않을 수 있다.

3. Steiner Tree Problem에 적용

3.1 Steiner Tree Problem

Jakob Steiner에 의해 이름 붙여진 Steiner Trree Problem (STP)[6]는 조합 최적화 문제 중 하나이다. STP는 그래프 $G(V,E)$와 간선들의 weight를 결정하는 함수 $w: E \rightarrow R^+$와 V의 부분집합 T가 주어졌을 때, 아래 두가지 조건

1) $V_s \subset V,\ E_s \subset E$

2) $T \subset V_s,$

을 만족하고 모든 간선의 weight 값의 총 합이 가장 작은 minimum weight tree $G_s(V_s, E_S)$를 찾는 것을 목적으로 한다.

T는 보통 터미널 노드(terminal node)라고 불리며 Steiner tree에 반드시 속해야 하는 노드를 일컫는다. STP는 minimum spanning tree 문제와 비슷해 보이지만 그와는 다르게 STP의 경우, steiner point 혹은 Steiner vertices라고 불리는 $V-T$에 속하는 노드들은 Steiner tree에 속하지 않아도 된다.

3.2 알고리즘 적용

기존의 ACO[5]를 변형하여 STP를 해결하기 위해서 주로 사용되는 ACO[7][8]를 STP-ACO라 하면, STP-ACO는 터미널 노드보다 하나 적은 수의 개미를 사용하여 모든 개미들을 각기 다른 터미널 노드에서 출발시키는 방식이다. 이 방식에서 각각의 개미들은 다른 터미널 노드나 다른 개미들이 방문한 node를 방문하게 될 경우, 탐색을 중지하게 되고, 모든 개미들이 탐색을 중지하면 하나의 트리가 만들어진다. 그리고 ACO의 페로몬 갱신 규칙에 의해 페로몬을 갱신한다. 이 방식은 기존의 ACO[5]를 STP를 풀기에 적합하게 개선한 방식으로 그 효율성은 실험적으로 증명되었기 때문에 CAO와 비교하기로 한다.

CAO의 경우 임의의 터미널 노드에서 개미를 출발시킨다. 다른 터미널 노드를 찾게 되면 지나온 노드와 간선으로 연결된 노드 중에서 지나지 않은 노드를 식(1)의 확률로 방문하게 된다. 모든 터미널 노드를 방문하게 되면 탐색을 중지하고 식(2)을 따라 페로몬을 갱신한다.

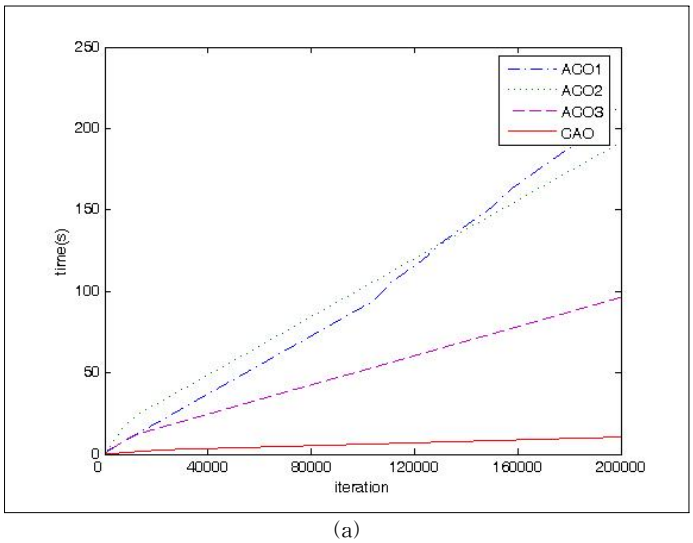
3.3 결과 및 분석

본 논문에서는 ACO를 적용할 때 사용하는 파라미터 중에서 α, β 값은 다른 논문[8]에서 실험에 사용한 값과 마찬가지로 각각 5, 1로 결정하였고 ACO1과 ACO2에 동일하게 사용하였다. ρ 값은 ρ 값에 따른 성능 비교를 위해서 ACO1에서는 0.0001로 하고 ACO2에서는 0.00001로 다른 값을 사용한다. 그리고 ACO3에서는 ACO2와 동일한 파라미터 값을 사용하지만, 각 개미가 경로를 찾는 것을 병렬적으로 처리한 경우이다. CAO에서는 m_1, C 값을 모두 100으로 하였다.

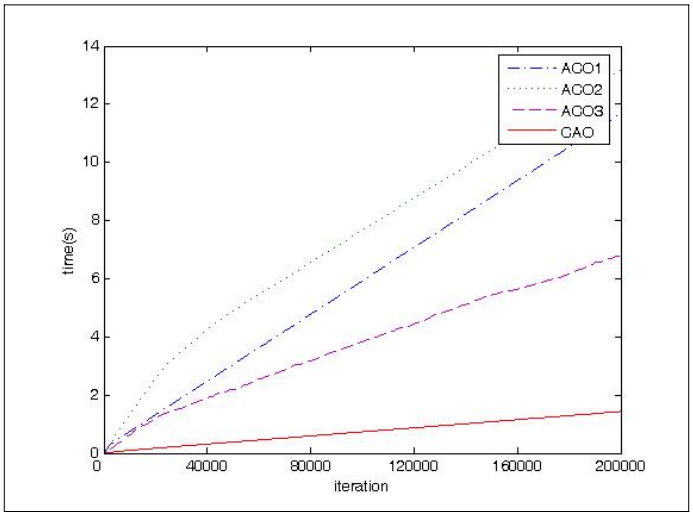
모든 알고리즘은 종료 iteration을 500000으로 해서 종료 iteration까지 해를 찾도록 하였다. Tree를 구성하는 모든 간선들의 weight의 총합을 tree의 weight라고 하고 매 iteration에서 이제까지 찾은 tree의 weight 중에서 가장 작은 값의 weight를 가지는 tree를 최적의 tree라고 할 때, 매 100 iteration마다 걸리는 시간과 찾은 tree의 weight 값, 그리고 최적의 tree의 weight 값을 측정하였다.

실험을 위해 그래프 $G(V,E)$는 undirected complete graph $G(V,E)$와 모든 노드 $i \in E$에 대해 w_{ii}값은 0으로 하고 그 외 모든 값은 0과 100사이의 실수 값을 랜덤하게 가지는 weight function $w: E \rightarrow R$이 주어진다.

그림 1과 그림 2과 그림 3의 (a)는 노드의 수가 50개 이고 터미널 노드의 수는 10개인 경우의 실험 결과이고 (b)는 노드의 수가 20개이고 터미널 노드의 수가 5개인 경우의 실험 결과로 각 실험은 두 경우 모든 간선의 랜덤하게 생성된 weight 값이 각각 다른 세 가지 그래프에 대해 각각 20회씩 수행되었고 최종적인 결과 값은 각 실험 결과의 평균값이다.



(a)



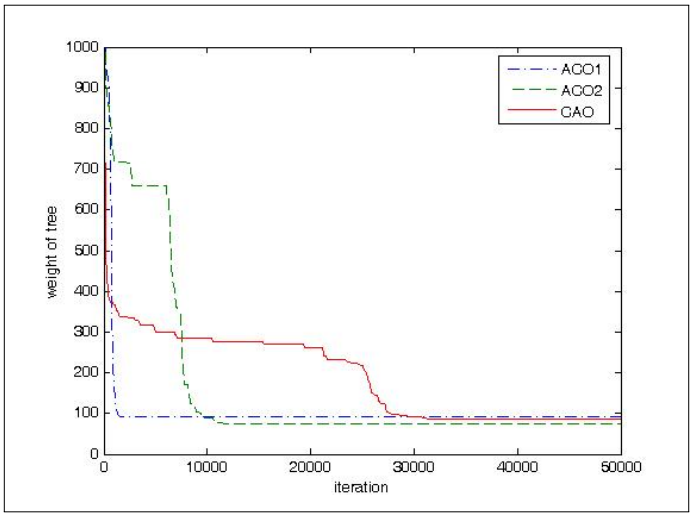
(b)

그림 1. 각 iteration까지 걸리는 시간. (a) 노드의 수가 50개 이고 터미널 노드의 수가 10개인 경우. (b) 각 iteration까지 걸리는 시간노드의 수가 20개 이고 터미널 노드의 수가 5개인 경우.

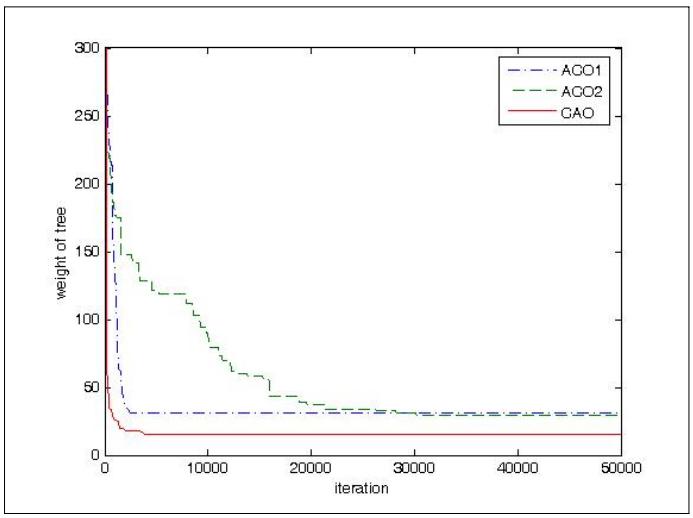
그림 1은 첫 번째 iteration에서부터 각 iteration까지 time() 함수를 통해 시간을 측정한 것으로 ACO1과 ACO2의 경우엔 각 개미들이 순차적으로 경로를 탐색한 것으로 하여 시간을 측정한 것이고 ACO3인 경우에는 각 개미들이 병렬적으로 경로를 탐색한 것으로 하여 측정한 것이다.

그림 1을 보면 각 iteration이 반복될수록 ACO의 시간 비용이 CAO에 비해 크게 증가한다는 것을 확인할 수 있다. 이것은 ACO의 경우 각 iteration에서의 시간 비용이 CAO보다 크기 때문에 iteration이 반복될수록 그 차이가 누적되기 때문이다. 이렇게 ACO1, ACO2, ACO3 세 방식보다 CAO의 시간 비용이 효율적이라는 것을 실험을 통하여 증명하였다.

Tree를 구성하는 모든 간선들의 weight의 총합을 tree의 weight라고 하고 매 iteration에서 이제까지 찾은 tree의 weight 중에서 가장 작은 값의 weight를 가지는 tree를 최적의 tree라고 할 때, 그림 2는 매 iteration의 최적의 tree의 weight를 그래프로 나타낸 것이고 그림 3은 각 iteration에 찾은 tree의 weight를 그래프로 나타낸 것이다.

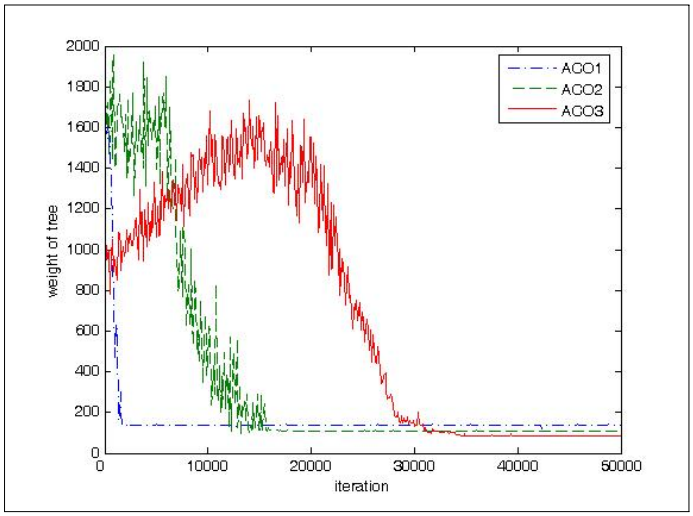


(a)

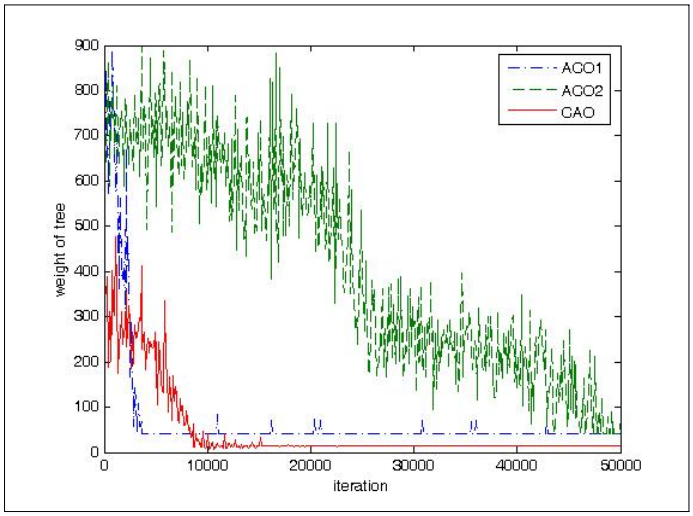


(b)

그림 2. 각 iteration까지 찾은 tree의 weight값 중에 가장 작은 값. (a) 노드의 수가 50개 이고 터미널 노드의 수가 10개인 경우. (b) 노드의 수가 20개 이고 터미널 노드의 수가 5개인 경우.



(a)



(b)

그림 3. 각 iteration에서 찾은 tree의 weight 값. (a) 노드의 수가 50개 이고 터미널 노드의 수가 10개인 경우. (b) 노드의 수가 20개 이고 터미널 노드의 수가 5개인 경우.

그림 2와 그림 3을 통해서 CAO는 초기에 탐욕적 탐색과 비슷하지만 초기 페로몬의 변화량이 작고 일정 iteration마다 초기 페로몬 값을 이용하여 탐색하기 때문에 충분한 탐색이 가능하다는 것을 확인할 수 있다. 그리고 CAO가 충분히 탐색을 한 이후엔 찾은 최적의 해로 빠르게 수렴해 간다는 것을 그림 3에서 일정 iteration 이후에 각 iteration마다 찾은 tree의 weight 값이 일정하다는 것을 통해 또한 확인할 수 있다. 그리고 그림 2와 3의 최종적으로 얻게 되는 tree의 weight가 비슷하다는 것을 통해 CAO가 ACO만큼 최적의 해 x^*를 찾는데 좋은 성능을 보일 수 있다는 것을 확인할 수 있다.

표 1과 표 2는 노드가 50개이고 터미널 노드의 수가 10개인 경우 각 알고리즘 별로 특정 iteration에서의 측정 결과를 표로 나타낸 것이다. 각각의 알고리즘 수행 후, 각 알고리즘이 종료될 때까지 얻은 tree 중에 가장 작은 weight를 가지는 tree를 x^*_{ACO1}, x^*_{ACO2}, x^*_{CAO}라고 한다.

표 1. 노드의 수가 50개이고 터미널 노드의 수가 10개인 경우 각 알고리즘에서 찾은 최적의 tree의 정보.

	ACO1	ACO2	CAO
weight	91.412	77.25	85.49
iteration	2200	10700	33300
time(s)	3.11	21.66	3.19

표 1에서 weight는 x^*_{ACO1}, x^*_{ACO2}, x^*_{CAO}의 weight 값이고 iteration은 x^*_{ACO1}, x^*_{ACO2}, x^*_{CAO}를 최초로 찾은 iteration이다. 그리고 마지막으로 time은 해당 iteration까지 걸린 시간이다. CAO는 비록 ACO2보다는 해가 좋지 않지만 ACO1보다는 좋고, 알고리즘의 최적의 해를 ACO1과는 비슷한 시간에 그리고 ACO2보다는 매우 빠른 시간에 찾을 수 있다는 것을 표 1을 통해 확인할 수 있다.

4. 결론

ACO는 충분한 개미의 수를 사용하거나 ρ값을 0에 근접한 값으로 할 때 일정 iteration 이후에 최적의 해를 찾는 확률이 0보다 크다. 하지만 개미의 수를 충분히 하거나 ρ를 0에 근접한 값으로 하게 되면 계산 비용과 필요한 메모리를 증가시키는 단점이 있다. 이러한 단점을 보안하기 위하여 GA의 단점을 보안하여 제안된 알고리즘 elitism cGAs의 기본 아이디어를 ACO에 반영하여 새로운 알고리즘 CAO를 제안하였고 본 알고리즘이 성능의 저하 없이 ACO보다 시간 효율성이 높다는 것을 실험을 통해 증명하였다. 향후에 CAO를 적용하여 조합 최적화 문제에서 최적의 해를 찾는다는 것을 이론적으로 증명하고 STP를 제외한 다른 조합 최적화 문제에도 CAO를 적용하여 그 효율성을 보이고자 한다.

참 고 문 헌

[1] M. Dorigo and L. M. Gambardella, “Ant colony systems: A cooperative learning approach to the traveling salesman problem,” *IEEE Trans. on Evolutionary Computation*, vol. 1 pp. 53 - 66, 1997.

[2] Ahn, C. W. and Ramakrishna, R. S. (2003). “Elitism-based compact genetic algorithms.” *IEEE Trans. on Evolutionary Computation*, 7(4):367-385.

[3] G. Rudolph, "Convergence Analysis of Canonical Genetic Algorithms," *IEEE Trans. on Neural Networks*, vol. 5, no. 1, 96-101, 1994.

[4] G. Harik, F. G. Lobo, and D. E. Goldberg, “The compact genetic algorithm,” *IEEE Trans. on Evolutionary Computation*, vol. 3, pp. 287 - -297, Nov. 1999.

[5] W. J. Gutjahr, "A Graph-based Ant System and Its Convergence," *Future Generation Computer Systems*, vol. 16, no. 8, 2000.

[6] F. Hwang and D. Richards, "Steiner Tree Problems," *Networks*, Vol.22, pp. 55-89, 1992

[7] S. Das, S. V. Gosavi, W. H. Hsu, and S. A. Vaze, "An Ant Colony Approach for the Steiner Tree Problem," *In: Proc. of Genetic and Evolutionary Computing Conference*, New York City, New York, 2002.

[8] Yu Hu, Tong Jing, Xianlong Hong, Zhe Feng, Xiaodong Hu, and Guiying Yan, "An efficient rectilinear steiner minimum tree algorithm based on ant colony optimization," In *Proceedings of IEEE ICCCAS*, pages 1276--1280, Chengdu, China, 2004.