탐색개미-일개미 군집화 알고리즘[1)]

A Search-ant and Labor-ant Algorithm for Clustering Data

심규석, 이희상, 김윤배, 박진수, 김재범

성균관대학교 산업공학과

{gsshim, leehee, kimyb}@skku.edu, jsf001@paran.com, kjbnhd@skku.edu

Abstract

데이타 마이닝, 특히 군집화 분야는 개미기반 알고리즘(ant-based algorithm) 관련 연구가 주목받는 영역 중 하나이다. 그러나 개미기반 알고리즘은 실제 군집화 문제를 해결하기에 많은 어려움을 가지고 있다. 본 논문에서는 기존의 알고리즘들을 고찰하고, 보다 실제적인 개미의 행태를 고려하여, 개선된 알고리즘을 제안한다. 실제 개미의 먹이 수집 과정에서 일어나는 초기 탐색개미의 움직임과 저장소를 향한 일개미들의 움직임을 반영하여, 보다 빠르고 효율적인 군집화를 이끌어 낸다. 인공자료와 실제 자료에 적용하여, 제안하는 알고리즘이 성능을 향상시킬 수 있고, 보다 다양한 자료에 적용될 수 있다는 것을 실험결과로 보였다.

1. 서론

최근, 새로운 기술과 생활환경의 변화로 인해, 대용량 데이타에 대한 군집화가 필요해지고 의미 있게 되었다. 현재 다양한 군집화 기법들이 연구되어 실제 사례에 적용되고 있는데, 잘 알려진 군집화 기법으로는 K-평균 군집화, CART(Classification And Regression Trees), SOM(Self-Organizing Map) 등이 있다. 이러한 군집화 기법들은 크게 4종류로 나눌 수 있다 : 분할 기법(partitioning method), 계층적 기법(hierarchical method), 밀도기반 군집화(density-based clustering) 및 격자기반 군집화(grid-based clustering). 개미기반 군집화(ant-based clustering)는 격자기반 휴리스틱 군집화 기법(grid-based heuristic clustering method) 중 하나이다.

개미 기반 군집화은 실제 개미의 행동으로부터 얻어진 영감(inspiration)을 기반으로 한다. 이 군집화 기법은 각각의 데이타들을 시체로 보고, 개미들의 시체 쌓기 과정에서 일어나는 두 가지의 종류의 행동, 줍기(pick-up)와 내려놓기(drop), 을 이용하여 군집화를 수행한다. 이러한 접근은 개미군락 최적화(ant colony optimization)[1] 또는 군중지능(swarm intelligence)[2]적인 측면으로 바라볼 수 있으며, 개미 알고리즘(ant algorithm)[3]의 넓은 범위에 속한다. 개미기반 군집화는 자기 조직적인 군집(self-organizationally cluster)을 형성해가며, 시각화(visualization)면에서 뛰어나다. 현재 문서에 대한 텍스트 마이닝(text mining)등 여러 실제 사례 적용에 대한 연구들도 활발하게 이루어지고 있다.

본 논문에서는 보다 실제적인 개미의 행태를 고려하여, 새로운 격자기반 개미 군락 휴리스틱을 제안한다. 제안하는 알고리즘은 실제 개미의 식량, 시체 수집과정에서 일어나는 초기 탐색개미의 움직임을 반영하여, 탐색개미와 일개미라는 두 종류의 개미들을 사용한다. 또한, 각각의 일개미들이 시체 또는 식량을 들고 자신들의 보금자리(nest)로 향한다는 점을 적극적으로 반영하여, 개미들의 식량, 시체 저장 창고인 저장 둥우리(storage nest)를 초기에 구성하고, 식량을 짊어진 모든 일개미들이 자신에게 맞는 저장 둥우리를 향해 이동하도록 한다. 한 종류의 개미를 사용하는 것에 비해, 역할이 분담된 두 종류의 개미들은 보다 빠르고 효율적인 군집화 과정을 이끌어 내며, 고정된 저장 둥우리를 향해 이동하도록 함으로써, 혼잡하고 길었던 일개미의 이동선이 간결해지고 짧아진다.

본 논문의 절차는 다음과 같다. 2절에서는 개미기반 군집화에 대해 간단히 설명하고, 3절에서는 우리의 알고리즘을 소개한다. 4절에서는 실험과 평가가 이루어지고, 5절에서 결론을 맺는다.

2. 개미 기반 군집화(ant-based clustering)

1990년, Deneubourg et al. [4]은 실제 개미의 시체 쌓기 과정을 반영하여 에이전트들이 데이타들을 주변과의 유사성에 따라 줍기(pick-up) 또는 내려놓기(drop) 행위를 하도록 하는, 최초의 개미 기반 군집화 알고리즘을 제안했다. 이후 Lumer and Faieta [5]는 비유사성을 측정하는 공식을 소개했고, 단기 기억 메모리(short term memory)를 개선하는 등 Deneubourg의 알고리즘을 개량했다. Kunz et al. [6]은 Lumer and Faieta의 알고리즘을 개선하였으며, 그래프 분할(graph partitioning), 그리고 그것과 관련된 문제들에 대해 적절하게 적용하였다. Handle et

1) 본 논문은 2004년 성균학술연구비의 지원으로 이루어졌습니다.

al. [7][8]은 Lumer and Faieta의 알고리즘을 텍스트 마이닝에 적용하였고, 임계함수(threshold function) 수정, 이웃함수(neighborhood function) 수정, 바로 가는 점프(jump), 활동기반 α-적응 (activity-based α -adaptation) 등 여러 시각에서 알고리즘 개선을 시도하였다.

한편, Deneubourg의 시체 쌓기 모형을 사용하지 않은 개미 군집화 알고리즘도 연구되었다. Labroche et al. [9][10]은 화학적인 냄새(chemical odor)와 몇몇의 행동 규칙을 이용하여 AntClust라는 새로운 알고리즘을 제안하였고, Azzang et al.[11]은 의사결정나무 형식의 AntTree를 제안하여 문서(document)에 적용하였다.

Deneubourg의 모형은 개미와 시체(데이타 개체)로 점유되는 격자평면에서의 움직임으로 이루어진다. 각 단계에서 개미들은 현재의 위치에서 주변 이웃 위치(위, 아래, 오른쪽, 왼쪽)로의 임의적인 이동을 하게 된다.

개미가 이동 중 시체를 만나게 되면,(그리고 그 개미가 아직 시체를 짊어지고 있지 않다면) P_p의 확률로 줍는다. 줍기에 성공한다면, 주변 이웃 위치로 임의적 이동을 하면서 P_d의 확률로 내려놓는다.

Deneubourg [4]는 해당 위치에서의 줍기(pick-up), 내려놓기(drop) 확률을 다음과 같은 임계함수(threshold function)로 제안했다.

$$P_p(i)=\left(\frac{k^+}{k^+ + f(i)}\right) \quad (1)$$

$$P_d(i)=\left(\frac{f(i)}{k^- + f(i)}\right) \quad (2)$$

여기서 k^+, k^-는 파라미터이다.(일반적으로 k^+=0.1, k^-=0.3을 사용한다.)

Lumer and Faieta[5]는 이웃함수(neighborhood function)를 다음과 같이 제안했다. 이웃함수는 한 개체와 주변 개체들 간의 비유사성을 측정한다.

$$f(i)=\max\left(0.0,\frac{1}{\sigma^2}\sum_{j\in L}\left(1-\frac{\delta(i,j)}{\alpha}\right)\right) \quad (3)$$

i는 현재 위치한 장소의 시체(데이타 개체)이고, j는 주변 이웃 L(local neighborhood L)의 시체들이다. $\delta(i,j)\in[0,1]$는 비유사성 함수이다. (연속형 속성의 경우 거리로 계산한다.) $\alpha\in[0,1]$ 은 데이타-종속 크기조정 파라미터(data-dependent scaling parameter)이다. σ^2은 L의 크기이다(σ는 L의 지름). 이웃함수를 고려하며, 줍기, 내려놓기 확률을 고찰하면 주변 이웃에 비유사한 시체가 많을수록 줍기 확률이 높아지고, 반대로 유사한 시체가 많을수록 내려놓기 확률이 높아진다는 것을 알 수 있다. 본 논문이 제안하는 알고리즘에서는 Deneubourg의 임계함수와 Lumer and Faieta의 이웃함수를 사용한다.

Lumer and Faieta[5]에 의해 개선된 단기 기억 메모리는 임의적 이동(random walk)의 방향을 잡아준다. 각각의 개미들은 가장 최근에 옮겼던 시체 몇 개와 그 위치를 기억한다. 새로운 시체를 줍게 되면, 단기 기억 메모리에 저장된 시체들과의 유사성 검사를 통해(연속형의 경우 거리를 사용) 가장 유사한 시체의 위치를 알아낸다. 그리고 그 위치를 이용하여 임의적 이동의 방향을 정한다. 이런 이동은 개미들의 이동선을 길고 복잡하게 만들며, 군집화 수행에 필요한 루프 반복(iteration) 수를 증가시킨다. 본 논문이 제안하는 알고리즘에서는 개미 각각에게 메모리를 부여하지 않고, 저장 둥우리(storage nest)들에 대한 정보를 담은 메모리를 모든 개미가 공유한다. 자신에게 맞는, 위치가 고정된 저장 둥우리를 향해 이동하도록 함으로써, 일개미의 이동선이 간결해지고 짧아지게 되며, 기존방법에 비해 훨씬 적은 루프 반복을 필요로 한다.

기존 개미기반 군집화에서의 개미와 데이타 개체들의 움직임을 고찰하면, 우선 데이타 개체들이 크게 한 덩어리로 뭉쳐진 형태를 이룬 후, 격자평면을 크게 이동하면서 점차적으로 작은 덩어리를 분류시키면서 군집화가 이루어진다는 것을 알 수 있다. 그러한 작은 덩어리들은 추후에 다시 뭉칠 수도 또 다시 나눠질 수도 있다. 이러한 과정 속에서 개미들의 긴 이동거리를 갖게 되고, 군집화는 대용량 데이타를 다룰 수 없을 만큼 상당히 늦게 완료된다. 또한, 군집화를 통해 몇 개의 군집이 생기게 될지 사용자는 미리 알 수 없다. 적절한 루프 반복(iteration) 횟수로 군집화가 적절하게 완료되었는지 여부도 확인하기 어렵다. 루프 반복 횟수에 따라, 즉 시간에 따라 군집의 개수가 증가하거나 감소하게 되는데, 수백만의 루프 반복동안 같은 수의 군집을 유지하다가, 어느 시점을 기준으로 군집의 수가 변할 수 있다. 사용자는 루프 반복 횟수를 늘려서 추후의 과정을 확인해야 하는지, 횟수를 늘린다면 얼마나 늘려야 하는지에 대해 고민해야 한다.

3. 탐색개미-일개미 군집화 알고리즘 (SLAC : Search-ant and Labor-ant Clustering Algorithm)

3.1 주요 착상

제안하는 알고리즘에서는 군집의 개수를 미리 정하게 하여, 사용자가 목적에 따라 유용하게 활용할 수 있도록 하였다. 이러한 접근은 루프 반복 횟수에 따라 군집의 수가 변하는 기존의 알고리즘에 비해, 보다 명확하게 군집화 완료 시점을 정할 수 있다는 장점도 가진다. 단, 비슷한 군집이 형성되어 합치게(merge) 될 경우 기대했던 군집 개수보다 적은 수를 얻게 될 수도 있다.

군집의 수가 정해지면 그 수만큼의 저장 둥우리를 평면상에 균일한 크기로 생성하여 할당한다. 할당된 영역에 각각 중심점을 배치하고 이 점들을 기준으로 저장 둥우리가 형성된다. 탐색개미들은 각각의 저장 둥우리 중심점 주변에 데이타 개체를 떨어뜨린다. 이때, 같은 저장 둥우리의 데이타 개체들은 유사하게 되고, 서로 다른 저장 둥우리의 데이타 개체들끼리는 되도록 비유사하게 되도록 데이타 개체를 떨어뜨린다. 탐색개미들에 의해저장 둥우리에 떨어진 데이타 개체들은 둥우리 메모리에 저장된다. 이후의 과정에서 둥우리 메모리에 저장된 데이타 개체들은 옮겨지지 않고 저장 둥우리의 기반을 지킨다.

각각의 둥우리 메모리는 탐색개미와 일개미들에게 페로몬 역할을 한다. 방향설정의 길잡이 역할을 하게 되며, 적절한 식량들이 쌓이도록 이끌어준다.

3.2 메인 알고리즘(Main Algorithm)

메인 알고리즘은 아래의 Algorithm 1과 같다. 우선 데이타 개체를 임의적으로 격자평면에 산포한다. 먼저 탐색개미를 활성화 하여 각 저장 둥우리를 구성한다. 이후 모든 일개미들은 둥우리 메모리에 속해있지 않은 데이터 개체 중 임의의 데이터 개체 위치로 이동하여 그 개체를 줍는다. 이후 일개미들은 그들의 일을 시작하게 된다. 정해진 횟수만큼 일을 하고 나면 알고리즘이 종료된다.

Algorithm 1 Main Algorithm

```
procedure Main
  //Activate search ant
  call ActiveSearchAnt
  for all agents do
      randomly select a data item
      move to the item
      pick up the item
  end for
  for all iteration do
     //Activate labor ant
     call ActiveLaborAnt
  end for
end procedure
```

3.3 탐색개미

각 저장 둥우리는 3.1절에서 설명한 바와 같이 초기 배치된다. 그리고 각 저장 둥우리에 둥우리 메모리(nest memory)를 준비한다.(Algorithm 2의 initialize 둥우리 메모리부분). 둥우리 메모리는 탐색개미에 의해 채워질 데이타 개체 수만큼 준비한다. 데이타 개체가 할당되는 대로 해당 저장 둥우리의 둥우리 메모리에 그 속성값을 저장한다. 이 데이타 개체의 수는 사용자에 의해 정해지며, 저장 둥우리의 중심점 주변에 떨어뜨려진다. 탐색개미들의 임무는 이 둥우리 메모리를 다 채우는 것이다.

둥우리 메모리가 채워지는 과정은 다음과 같다.

1) 초기에, 탐색개미 한 마리가 우선 임의의 데이타 개체를 임의의 저장 둥우리에 떨어뜨린다. 떨어진 데이타 개체는 해당 저장 둥우리의 둥우리 메모리에 저장된다. (algorithm 2의 initial selection of random data 부분).

2) 탐색개미는 임의의 데이타 개체를 선택하고, data가 1개 이상 있는 저장 둥우리에 대하여 저장 가능 여부를 확인한다. 데이타 개체이 있는 저장 둥우리가 두 개 이상일 경우 각 저장 둥우리의 데이타 개체들과의 이웃함수값을 구하여, 가장 높은 쪽을 저장장소로 고려한다.

3-1) 해당 저장 둥우리의 둥우리 메모리에 빈자리가 있다면, 식(2)의 내려놓기 확률로 내려놓기 여부를 결정한다. 내려놓기 결정이 나오면 내려놓는다.

3-2) 3-1)과정에서 내려놓기에 실패하거나 해당 저장 둥우리의 메모리에 빈자리가 없다면, 데이타 개체가 하나도 없는 저장 둥우리가 있는지 확인한다. 그런 저장 둥우리가 없다면, data를 이동시키지 않고 그대로 둔 채 해당 루프 반복(iteration)을 마친다.

3-3) 3-2)의 과정에서, 만약 데이타 개체가 하나도 없는 저장 둥우리가 존재한다면, 현재의 저장 둥우리에서, 이미 저장 둥우리에 있는 데이타 개체들을 이웃 데이타 개체들로 하여 식(1)의 줍기 확률로 줍기를 시도한다. 즉, 해당 저장 둥우리에 있지도 않은 데이타 개체에 대해, 그곳에 있다고 가정한 후, 줍기를 시도하는 것이다. 이는 비유사성을 고려하며 확률적으로 다른 저장 둥이리의 초기 데이타 개체로서 적합한지를 확인하기 위함이다. 만약 줍기에 성공한다면, (이 줍기는 당연히 저장 둥우리 데이타 개체들과의 비유사성이 높을수록 성공 가능성이 높다), 완전히 비어있는 저장 둥우리에 현재 잡고 있는 데이타 개체를 내려놓는다.

3-4) 3-3)의 줍기에 실패한다면 데이타 개체를 이동시키지 않고 원래 위치에 그대로 둔 채 해당 루프 반복(iteration)을 마친다.

4) 위의 과정을 모든 저장 둥우리에 정해진 수만큼의 data가 찰 때까지, 즉 모든 저장 둥우리의 둥우리 메모리가 다 찰 때까지 반복한다. 다 차지 않았다면 2)번 절차로 간다. (algorithm 2의 filling the nest 부분)

Algorithm 2 Search-Ant Algorithm

```
procedure ActiveSearchAnt
  //initialize nest memory
  for all clusters do
       initialize nest and its memory
  end for
  //initial selection of random data
  randomly select an item and pick it up
  move to an empty nest and drop the item
  memorize the item and the position to nest memory
  //filling the nest
  while all memory is not full do
       randomly select an item
       get the best matched nest
       get f value of the item from the Nest memory
       get drop probability and pick-up probability
       if the nest memory is not full then
            pick up the item and move to the nest
            drop it with the drop probability
      end if
      if dropping is failed or the nest memory is full then
            pick up the item and move to an empty nest
            drop it with the pick-up probability (with best matched nest data items)
      end if
      if all dropping trials are failed then
```

```
        restore the item to the original place
      end if
    end while
  end procedure
```

3.4 일개미

임의로 하나의 개미를 선택하고, 모든 저장 둥우리에 대하여 식(3)의 이웃함수 값을 구한 후, 가장 값이 큰 쪽으로 방향을 잡는다. 그 방향으로 정해진 거리(stepsize)만큼 이동한 후, 식(2)의 내려놓기 확률로 내려놓기 여부를 결정한다. 내려놓기에 실패하면, 해당 루프 반복(iteration)을 마친다. 내려놓기에 성공하면 (개미들이 들고 있지 않은 데이타 개체 중) 임의의 데이타 개체에 대하여, 줍기((1)식의 확률을 반영)가 성공할 때까지 시도한 후, 성공하면 해당 루프 반복을 마친다.(Algorithm 3)

Algorithm 3 Labor-Ant Algorithm

```
procedure ActiveLaborAnt
  randomly select an agent
  step to the best matched nest with step size
  get drop probability
  drop its item with probability
  if dropping is not failed then
      while the agent has no item do
          randomly select an item
          pick up probability
        pick up the item with pick-up probability
      end while
  end if
end procedure
```

4. 실험 및 결과

4.1 실험 설계

실험에는 실제 데이타 세트와 2차원 정규분포(two-dimensional normal distribution $\mathcal{N}(\mu,\sigma)$)에 의해 생성된 인공 데이타 세트가 사용되었다. 인공 데이타는 Table 1과 같이 구성되었다. K는 cluster의 수, N은 데이타 개체의 수, Dim은 속성의 수를 나타낸다.

Table 1. 인공 데이타 세트

Name	K	N	Dim	Source
Art1	4	4×250	2	$\mathcal{N}([0,0],[2,2])$ $\mathcal{N}([10,10],[2,2])$ $\mathcal{N}([0,10],[2,2])$ $\mathcal{N}([10,0],[2,2])$
Art2	4	4×250	2	$\mathcal{N}([0,0],[2,2])$ $\mathcal{N}([6,6],[2,2])$ $\mathcal{N}([0,6],[2,2])$
Art3	4	769,77, 77,77	2	$\mathcal{N}([6,0],[2,2])$ $\mathcal{N}([0,0],[2,2])$ $\mathcal{N}([10,10],[2,2])$ $\mathcal{N}([0,10],[2,2])$ $\mathcal{N}([10,0],[2,2])$

Art1, Art2, Art3는 각각 Handl et al.[8]이 사용했던 square1, square5, sizes 5 data set과 같다.

실제 데이타로는 Machine Learning Repository(http://www.ics.uci.edu/~mlearn/MLRepository)의 붓꽃 데이타(Iris data set)를 사용하였다.(Table 2)

Table 2. 실제 데이타

Name	K	N	Dim
Iris	3	3×50	4

본 논문에서는 최근에 개선된 개미기반 일고리즘인 Handl et al.[12]의 알고리즘(ATTA)과 비교 평가하였다. ATTA는 기존의 알고리즘에 비해 속도가 크게 향상되고, 강인성(robustness)이 증가된 알고리즘이다. 평가는 오분류율과 수행시간 등으로 이루어졌다. Lumer and Faieta의 알고리즘과도 비교가 이루어졌으나, 대부분의 평가에서 탐색개미-일개미 군집화 알고리즘이나 ATTA에 크게 뒤지는 것으로 나와, 결과 Table에서는 제외했다. Lumer and Faieta의 알고리즘은 ATTA에 비해 군집간의 간격도 덜 벌어지게 되어, 군집 인식 면에서도 크게 뒤지 것으로 확인되었다.

탐색개미-일개미 군집화 알고리즘에서는 탐색개미와 일개미가 서로 다른 α를 사용한다. (탐색개미의 α가 더 작다.) 이는 탐색개미의 내려놓기에 보다 더 신중을 기울이려는 의도이며, 일개미의 빠른 내려놓기를 유도하기 위함이다. 실험 시 우리 알고리즘의 경우 초기에 입력해야하는 군집의 수는 이미 알려진 적절한 수를 사용하였다. ATTA와 우리 알고리즘 모두 군집화 시행 후, 군집들의 유사성을 고려하여 필요 시 군집들을 합병하였다. 만약 군집의 수가 이미 알려진 수와 다르다면, 그 사례는 오분류율의 평균, 오분류율의 표준편차, 평균 수행시간 계산 시 고려대상에서 제외시켰다. 실험 전체를 통틀어 군집의 수가 알려진 수 보다 많게 나온 경우는 없었으므로, 본 실험 결과에서 평균 군집의 수는 높은 값 일수록 좋다고 볼 수 있다. 실험환경은 Pentium 4 2.8G, 256M이었다. 수행시간은 두 알고리즘 모두 프로그램상의 시각적 효과는 제외된 상태로 측정되었다. 실험은 데이타 세트별로 각각의 알고리즘이 30번씩 시행되었고 그 결과를 토대로 Table 3~6이 작성되었다.

4.2 결과

Table 3. Art1 실험 결과

	ATTA	SLAC
루프 반복 횟수(iteration)	300,000	50,000
군집 수의 평균	4	4
4개의 군집이 나온 횟수	30	30
오분류율의 평균	0.014	0.016
오분류율의 표준편차	0.007	0.004
평균 수행시간(초)	6.79	2.92

개미기반 군집화는 데이타 개체들의 군집간 중첩(overlap)의 정도에 따라 성능이 크게 달라진다고 알려져 있다. Art1 데이타 세트는 비교적 분명하게 4갈래로 나눠지는 데이타 세트이다. Table 3을 보면, ATTA와 SLAC(탐색개미-일개미 군집화 알고리즘, Search-ant and Labor-ant Clustering Algorithm)이 모두 30번의 시도에서 전부 적절한 군집 수를 내었다는 것을 알 수 있다. 또한 둘 다 적은 오분류율을 보였다. ATTA의 루프 반복 횟수는 선행 시도를 통해 결정한 값이다. 반복 횟수에서 SLAC이 훨씬 적지만 비슷한 오분류율을 보였다. ATTA에 비해 훨씬 짧은 시간이 소요된 것을 볼 수 있다.

Table 4. Art2 실험 결과

	ATTA	SLAC (둥우리 메모리=20)	SLAC (둥우리 메모리=50)
루프 반복 횟수 (iteration)	1,000,000	50,000	50,000
군집 수의 평균	3.36	3.87	3.93
4개의 군집이 나온 횟수	21	26	28
오분류율의 평균	0.171	0.157	0.152
오분류율의 표준편차	0.009	0.031	0.048
평균 수행시간(초)	14.68	2.98	5.16

Art2 데이타 세트는 Art1 데이타 세트에 비해 군집 간 중첩부분이 많은 데이타 세트이다. 따라서 오분류율 값의 작은 차이는 큰 의미를 갖지 못한다. Table 4에서 보여지는 대로, ATTA는 1,000,000번의 반복 횟수에도 불구하고 평균3.34개의 군집밖에 못 만들어냈다. 특히 전산 상 군집을 2개로 인식하는 경우가 많았다. SLAC의 경우는 두 개의 군집이 합병되어 3개를 내놓은 경우가 있었다. 각 둥우리 메모리의(둥우리 메모리 하나당) 크기를 20에서 50으로 늘리면, 군집 수의 평균은 증가하는 반면, 오분류율의 표준편차가 늘어나고, 시간이 더 소요되는 것을 볼 수 있다. 탐색개미에 의해 초기 할당되는 데이타 개체 수가 많을수록 해당 군집이 합병되지 않고 남게 되는 경우가 많아지지만, 초기에 할당이 잘못될 경우, 그 여파가 군집화 과정 동안 지속적으로 작용하여, 오분류율이 크게 증가된다. 전반적으로 SLAC의 경우가 ATTA에 비해 오분류율의 표준편차가 크게 나왔다. 이는 군집의 수가 이미 알려진 수와 다르다면 그 사례는 오분류율의 평균, 오분류율의 표준편차 계산 시 고려대상에서 제외시켰기 때문이기도 하다. ATTA는 21번의 사례만으로 계산했지만, SLC는 각각 26번, 28번의 사례로 계산했다.

Table 5. Art3 실험 결과

	ATTA	SLAC
루프 반복 횟수(iteration)	1,000,000	50,000
군집 수의 평균	4	3.8
4개의 군집이 나온 횟수	30	24
오분류율의 평균	0.015	0.021
오분류율의 표준편차	0.005	0.022
평균 수행시간(초)	14.52	4.41

Art3 data set은 각 군집에 속하는 데이타 개체의 수가 각각 769,77,77,77로 이루어진 데이타 세트이다. 적절히 군집화 할 경우 커다란 군집 1개에 작은 군집 3개가 나오게 된다. Table 5는 SLAC의 취약점을 보여준다. 탐색개미들은 임의적으로 식량을 선택하기 때문에 전체 데이타 개체 수에서 차지하는 비율이 크다면 그만큼 선택될 가능성이 높다. 선택된 데이타 개체는 둥우리 메모리의 저장 개체로 고려되는데, 만약 둥우리 메모리 형성 초반에 두 저장 둥우리가 (작은 확률이지만) 유사한 데이타 개체로 채워져 간다면, 합병의 대상이 될 가능성이 높다. ATTA는 Art3 데이타 세트에 강한 면모를 보였지만 SLAC는 군집의 수를 지키지 못한 경우가 있었다.

Table 6. 붓꽃 실험 결과

	ATTA	SLAC
루프 반복 횟수(iteration)	1,000,000	5,000
군집 수의 평균	2	3
3개의 군집이 나온 횟수	0	30
오분류율의 평균	-	0.08
오분류율의 표준편차	-	0.012
평균 수행시간(초)	6.25	0.15

붓꽃 데이타 세트에 대한 결과(Table 6)에서는 ATTA가 단 한번도 3개의 군집을 형성하지 못하는 약한 모습을 보였다. 개미들의 초기 α값을 0.3~0.8까지 변동시키며 실험해봤지만 3개의 군집을 이루는 모습은 볼 수 없었다. 150개의 데이타 개체로 구성된 붓꽃 데이타 세트에 대해, SLAC는 평균0.15초라는 짧은 시간 안에 8%의 오분류율을 일으키며 군집화를 수행했다.

Fig 1. 붓꽃 데이타 세트를 두개의 군집으로 나눈 화면

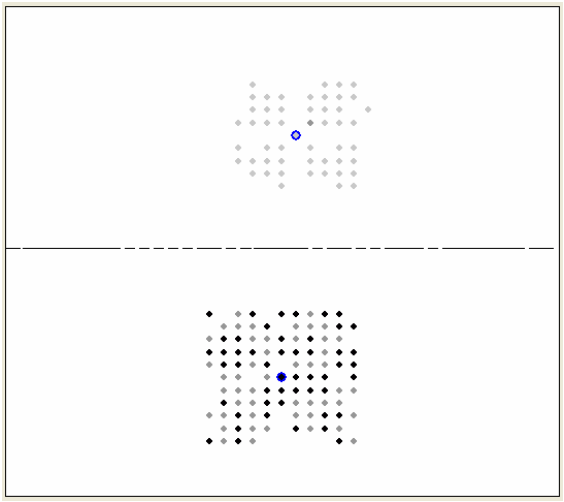

Fig 1. 은 붓꽃 데이타 세트가 2개의 군집을 갖도록 SLAC으로 군집화를 수행한 결과 화면이다. 위쪽으로 분류된 군집의 데이타 개체들은 Table 6의 실험에서도 유난히 높은 순도로 군집화 되었던(오분류된 경우가 별로 없었던) 데이타 객체들이다. 세 군집 중 다른 군집들과의 비유사성이 가장 높았던 군집의 데이타 개체들이 Fig 1.에서 두 군집 중하나를 이루고, Table 6의 실험에서도 간간히 섞여서 오분류율을 높였던(즉 상대적으로 비유사성이 낮았던) 두 군집의 데이타 개체들이 또 하나의 군집을 이룬 것이다. 본래 3개의 군집으로 나눠지게 되어 있는 데이타 세트에 대해 2개의 군집을 갖도록 수행한 결과도 역시 의미 있는 정보를 주고 있다. 이는 SLAC에서 군집의 수를 정해주는 방식이 사용자의 특정 목적에 따라 활용될 수 있음을 말해준다.

Fig 2. ATTA의 진행과정(Art1 데이타 세트)

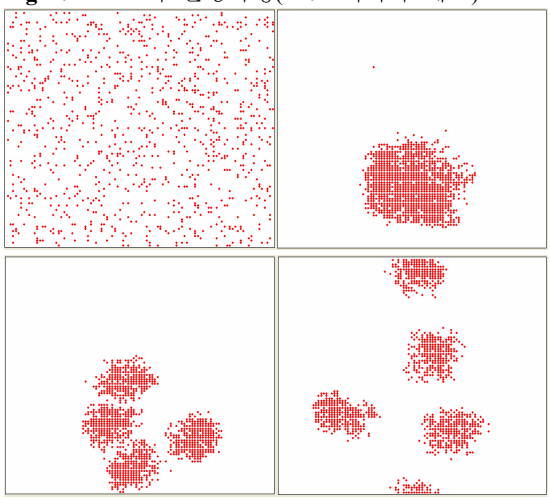

Fig 2. 는 ATTA의 진행과정을 나타내고 있다. 임의적으로 격자평면에 뿌려진 데이타 개체들이 우선 하나의 큰 덩어리로 뭉쳤다가 갈라지는 모습을 볼 수 있다. 토러스(Torus) 공간을 사용하기 때문에, 마지막 화면의 윗부분 군집과 아랫부분 군집은 서로 이어져있는 하나의 군집이다.

Fig 3. SLAC의 진행과정(Art1 데이타 세트)

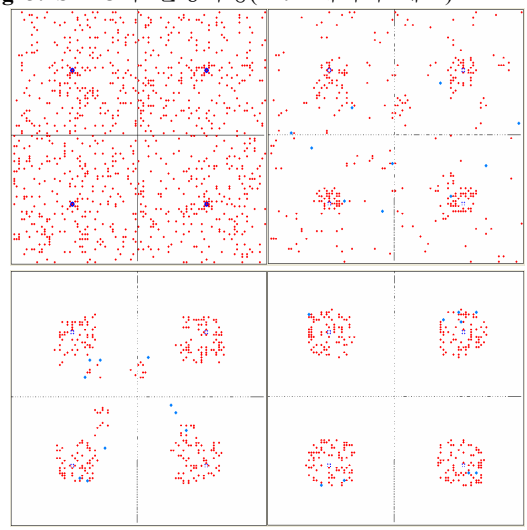

Fig 3. 은 SLAC의 진행과정이다. 첫 화면에 4개의 저장 둥우리 중심점이 보인다. 두 번째 화면은 탐색개미에 의해 저장 둥우리 주변에 데이타 개체들이 놓여져 있는 상황에서 일개미들이 약간의 일을 한 화면이다. 이후 일개미들에 의해 데이타 개체들은 각 둥우리에서 촘촘히 밀집된다.

Fig 4. ATTA의 Art2 데이타 세트 군집화 결과 사례 (반복 횟수 : 1,000,000)

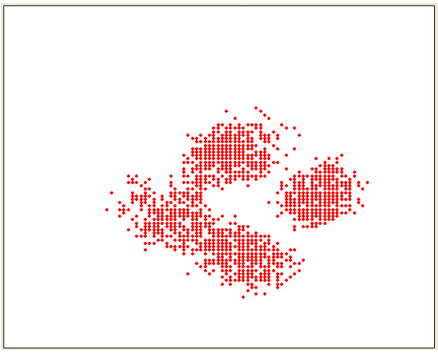

Fig 4. 는 ATTA의 Art2 군집화 결과 사례 중 하나이다. 시각적으로는 3개 또는 4개로 보이지만, 전산 상의 결과 파일을 확인한 결과 놀랍게도 군집은 1개로 인식되어 있었다. 화면상의 데이타 개체 분포를 분석한 결과 군집화는 적절하게 진행 중이었다. 즉, 반복횟수를 더 늘려주면 적절한 군집의 수가 나올 수도 있는 상황이었다. 하지만 얼마나 늘려야 하는지 그리고 과연 늘리면 적절한 결과가 나오는지에 대해서 확답을 구하기는 어렵다. 기존의 전산 상 군집 인식이 각 데이타 개체들 간 격자평면상의 거리를 기반으로 하기 때문에, Fig 4.와 같은 문제가 발생되는 것이다. 이런 문제점을 피하려면, 4개의 cluster가 좀 더 순도를 높이면서, 격자평면상에서 서로에게 멀리 떨어지는 방향으로 나아가야 한다. 하지만 덩어리들이 움직이다 보면, 떨어졌다가도 다시 가까워지는 모습을 보일 때가 있다.(이는 토러스 공간을 사용하기 때문이기도 하다. 그러나 토러스 공간을 쓰지 않으면 데이타 객체들의 움직임이 상당히 제한된다.) 이 경우 군집 인식을 언제 하느냐에 따라 군집의 수가 달라 질 수 있다.

즉, 반복 횟수를 늘린다고 해서 무조건 더 좋아지는 것은 아니다. 하지만 SLAC에서는 이런 상황이 발생하지 않는다. 군집 인식 방법이 다르기 때문이다. Fig.3에서도 볼 수 있듯이 촘촘히 모이면서 확연히 구분되므로 둥우리 중심점과의 거리를 이용하여 군집을 인식한다. 토러스 공간을 쓰지 않아도 문제가 생기지 않는다. 다만 반복 횟수가 적어서 진행과정 중에 끝나게 되면, 적절하지 않은 모습을 보일 수 도 있다. 그러나 반복 횟수를 늘려주면 반드시 Fig.3의 마지막 화면처럼 확연하게 뭉친다.

5. 결론

본 논문은 기존 개미 기반 군집화에 대한 고찰과 개미의 생태학적 행태에 대한 모방을 기반으로, 탐색개미-일개미 군집화 알고리즘(SLAC)을 제안하였다. 제안한 알고리즘은 탐색개미와 일개미라는 두 종류의 개미들을 사용하여 역할을 효율적으로 분담시켰고, 일개미의 이동선을 크게 줄였다.

SLAC은 실험을 통해서 기존의 개미기반 군집화 알고리즘에 비해 빠르고, 더 적용성이 높은 알고리즘으로 판단되었다. SLAC은 대용량의 데이타를 무리 없이 처리할 수 있을 만큼 빠르고 간결한 구조를 가졌다. 또한, 사용자의 특정한 목적을 위하여 군집의 수를 적절하게 정해 줄 수 있다. SLAC의 처리 속도라면, 사용자는 적절한 군집의 수를 찾기 위해 여러 번의 시도를 부담 없이 할 수 있을 것이다. 단, 탐색개미에 의해 배치되는 초기 데이타 개체들에 의해 이후 과정이 크게 영향을 받는다는 문제점과 함께 일부 데이타 세트에 대해 다소 높은 오분류율 표준편차를 가지게 된다는 점, 그리고 α 값과 둥우리 메모리의 크기 설정에 대한 합리적인 기준이 제시되지 않았다는 점은 앞으로 계속 개선되어야 할 부분들이다.

Reference

[1] Dorigo, M. & Di Caro, G. (1999). The ant colony optimization metaheuristic. In Corne, D., Dorigo, M., and Glover, F., editors, New Ideas in Optimization (pp. 11−32). London, UK: McGraw Hill.

[2] E. Bonabeau, M. Dorigo, and G. Theraulaz. Swarm Intelligence − From Natural to Artificial Systems. Oxford University Press, New York, NY, 1999.

[3] Dorigo, M., Bonabeau, E., & Theraulaz, G. (2000). Ant algorithms and stigmergy. Future Generation Computer Systems, 16(8), 851−871.

[4] Deneubourg, J.-L., Goss, S., Franks, N., Sendova-Franks, A., Detrain, C.,& Chr´etien, L. (1991). The dynamics of collective sorting: Robot-like ants and ant-like robots. In Proceedings of the First International Conference on Simulation of Adaptive Behaviour: From Animals to Animats 1 (pp. 356−365). Cambridge, MA: MIT Press.

[5] Lumer, E. & Faieta, B. (1994). Diversity and adaptation in populations of clustering ants. In Proceedings of the Third International Conference on Simulation of Adaptive Behaviour: From Animals to Animats 3 (pp. 501−508). Cambridge, MA: MIT Press.

[6] Kuntz P., Layzell P., Snyder D., A colony of ant-like agents for partitioning in VLSI technology, in: P. Husbands, I. Harvey(Eds.), Proceedings of the Fourth European Conference on Artificial Life, MIT Press, Cambridge, MA, 1997, pp.412~424.

[7] Handl, J. and Meyer, B. Improved ant-based clustering and sorting in a document retrieval interface, PPSN VII, LNCS 2439, 2002.

[8] Handl, J., Knowles, J., & Dorigo, M.(2004). Strategies for the increased robustness of ant-based clustering. In Engineering Self-Organising Systems, Vol. 2977 of Lecture Notes in Computer Science (pp. 90−104). Heidelberg, Germany: Springer-Verlag.

[9] N. Labroche, N. Monmarch´e, and G. Venturini, "A new clustering algorithm based on the chemical recognition system of ants," in Proc. of 15th European Conference on Artificial Intelligence (ECAI 2002), Lyon FRANCE, pp. 345−349, 2002.

[10] N. Labroche, C. Guinot, and G. Venturini "Fast Unsupervised Clustering with Artificial Ants" PPSN VIII, LNCS 3242, pp. 1143−1152, 2004.

[11] N. Azzag, H. Monmarch´e, M. Slimane, G. Venturini, and C. Guinot. Anttree : a new model for clustering with artificial ants. In IEEE Congress on Evolutionary Computation, Canberra, Australia, 08-12 December 2003.

[12] Handl, J., Knowles, J., & Dorigo, M.(2004). Ant-based Clustering and Topographic Mapping, , Artificial Life, Vol.12(1)