외판원 문제를 위한 다중 에이전트를 이용한 분산 유전 알고리즘

김 정 숙
김포대학 컴퓨터 계열 소프트웨어 개발 전공

Distributed Genetic Algorithm using Multi-agent for the Traveling Salesman Problem

Jung-Sook Kim,
Dept. of Computer Science, Kimpo College
E-mail : kimjs@kimpo.ac.kr

요 약

본 논문에서는 외판원 문제를 분산 시스템 환경에서, 다중 에이전트를 이용해 수행 시간을 단축시키고, 더욱 우수한 근접해를 구할 수 있는 분산 유전 알고리즘을 개발하였다. 다중 후보해를 이용한 분산 유전 알고리즘을 수행할 때, 고려해야 할 중요한 요소는 후보해들 간의 개체들을 어떤 노드의 후보해 개체와 교환할 것인가와 어떤 개체들을 선택해서, 얼마만큼의 개체를 이동시킬 것인가가 중요하게 고려되어야 한다. 따라서 본 논문에서는 교환해야 할 개체의 크기를 동적으로 윈도우 크기를 변경하면서 교환하는 방법을 개발하였고, 교환할 개체들의 위치를 결정하는 새로운 유전 이동 정책 2가지 방법을 개발하고 실험하였다.

1. 서론

외판원 문제는 주어진 n개의 도시들과 그 도시들 간의 거리 비용이 주어졌을 때, 처음 출발도시에서부터 정확히 한 도시는 한번씩만 방문하여 다시 출발도시로 돌아오면서 방문한 도시들을 연결하는 최소의 비용이 드는 경로를 찾는 문제로 최적해를 구하는 것은 전형적인 NP-완전 문제중의 하나이다[5,6,7]. 출발도시에서 시작해서 나머지 (n-1)개의 도시를 한 번씩 거치는 경로의 총수는 (n-1)! 만큼 존재하게 되어 계산복잡도는 지수시간이 된다. 그리고 이 문제는 n개의 도시 방문 순서를 일부분 바꾸더라도 경로의 합이 증가하게 되는 국소 최소점이 많이 있는 문제이다. 이렇게 국소 최소점이 많은 울퉁불퉁한 표면(rugged landscape)에서 전체 최적해를 찾는 문제는 변수 공간 전체에 대한 검색을 하지 않는 한 불가능하다. 그러나 이렇게 복잡한 외판원 문제가 실제 다른 최적화 문제들의 응용이 될 뿐만 아니라, NP-완전 문제를 현실적으로 해결하려는 노력의 일환이 된다. 따라서 외판원 문제를 해결하기 위한 다양한 연구들이 시도되고 있다. 분기 한정법(Branch-and-Bound) 알고리즘이나 동적 프로그래밍(Dynamic Programming) 알고리즘 방법과 같이 최적해를 구하는 연구들[5]이 있고, 확률적 탐색 휴리스틱(probability search heuristic)에 근거해서 근사해(near optimal)를 구하는 유전 알고리즘[3,7] 등 다양한 방법들이 개발되었다. 더불어 요즈음은 통신망과 가격이 저렴하면서 성능 좋은 분산 시스템의 발달로 이러한 분산 환경에서 수행시간을 단축시키면서 성능 향상을 가져올 수 있는 분산 알고리즘들이 많이 개발되고 있다. 특히 분산 인공지능이라는 분야에서 비롯된 에이전트는 네트워크의 부하를 줄이고 비동기적이고 자율적으로 실행되며, 동적으로 적응할 수 있고, 플랫폼에 독립적이라는 장점을 가지며, 네트워크 상에 분산되어 존재하는 호스트들로 구성된 가상의 분산 컴퓨팅 환경을 구축할 수 있다[2]. 이러한 분산 환경에서 다중 후보해를 가진 분산 유전 알

고리즘에서 성능에 영향을 많이 미치는 몇 가지 요소들이 고려되어야 한다[1]. 가장 먼저 노드간에 이동해야 할 후보해의 크기가 결정되어야 하고, 그 선택된 후보해의 일부가 어떤 노드의 후보해와 교환되어야 할 것인가가 결정되어야 한다. 더불어 교환해야 할 후보해는 어떻게 선정되며 어떤 개체를 대체할 것인가와, 후보해의 크기는 얼마로 해야 가장 우수한 해를 구할 수 있을 것인가를 고려해야 한다.

. 그래서 본 논문에서는 이러한 여러 가지 성능에 영향을 미치는 요소들을 진화적인 방법을 사용해서 수행시간을 줄이면서 성능향상을 고려할 수 있도록 분산 유전 알고리즘을 개발하였다. 먼저 노드간의 이동해야 할 후보해의 크기를 동적으로 변화시킬 수 있는 윈도우를 사용하였다. 처음에는 개체의 크기를 하나에서부터 시작해서 교환이 이루어질 때마다 일정하게 증가해 가는데 후보해의 임의의 비율 범위 안에서 증가하는 방법을 모색하였고, 다음은 윈도우 크기를 결정할 때, 유전 연산을 수행해서 전 세대보다 결과가 θ 값보다 크면 변경해야 할 윈도우 크기를 적게 해서 이동하고, θ 값보다 작으면 교환되어야 할 개체의 크기를 많이 선택해서 변경하도록 개발하였다. 그리고 후보해의 개체가 변경되어야 할 위치를 결정하는 새로운 유전 이동 정책을 2가지 개발하였다. 첫째, reciprocal 유전 이동 정책으로 임의의 두 노드를 선정해서 윈도우 크기만큼 우수한 개체를 선정해서, 가장 나쁜 개체와 교환하도록 하는 이동 정책이다. 다음은 환형큐(circular queue) 유전 이동 정책으로 모든 노드가 마치 환형큐가 연산하는 것처럼 서로 이웃하는 노드에게 개체를 각각 교환하는 방법이다.

논문의 구성은 먼저 2장에서 관련 연구를 살펴보고, 3장에서 이동할 개체의 크기를 동적으로 조절하는 윈도우에 대해 기술한다. 그리고 4장에서는 개체들이 이동해야 할 위치를 결정하는 유전 이동 방식 2가지를 설명하고, 5장에서는 실험한 내용과 결과를 분석하고 마지막 6장에서 결론을 내리고 향후 연구과제를 제시한다.

2. 관련 연구

2.1 유전 알고리즘

유전 알고리즘을 특정 문제에 적용시키기 위해서는 개체가 문제의 해결책을 어떻게 표시하도록 할 것인지를 설계해야 하고, 개체의 우수한 정도를 나타내는 적응도를 계산하는 적합성 함수(fitness function)를 어떻게 정의할 것인지, 또 우수한 개체들을 집단에서 어떤 방법으로 선택하고 몇 개로 복제해야 하는지, 교차 연산과 돌연변이 연산을 어떻게 정의할 것인지를 결정해야 한다. 이들 적용 방법이 어느 정도로 문제의 특성을 잘 반영하고 있느냐에 따라 알고리즘의 성능이 크게 달라진다. 또한, 유전 알고리즘은 자체에 병렬성이 내재되어 있어서 수행 시간의 향상과 근접해의 향상을 가져오기 위해 병렬로 많이 처리한다.

외판원 문제를 위한 유전 알고리즘들은 다양한 후보해(candidate population)의 표현과 유전 연산을 제시하고 있다. 후보해의 표현 방법은 정수 표현 방법을 사용하고 있으며, 유전 연산자는 주로 교잡 연산과 돌연변이 연산이 사용되고 있다. 따라서 외판원 문제의 입력자료로서 개체는 모든 도시들을 정수로 매핑하여 n개의 도시들을 임의로 생성 가능한 도시들의 배열로 입력한다. 이렇게 생성된 개체의 예를 들어 도시의 수가 17개인 경우 다음과 같다. 1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9 - 10 - 11 - 12 - 13 - 14 - 15 - 16 - 17, V는 도시 "1"에서 출발하여 도시 2를 방문하고 다음 도시 3을 방문하고 등으로 하여 도시 17을 방문하는 경로를 나타내며, V는 목적 함수 값으로 도시들을 방문하면서 걸린 비용의 합을 나타낸다.

초기 후보 집단은 이렇게 임의의 수들의 배열로 생성한 후 생성된 개체들에 대한 비용을 계산한다. 그 비용들을 정렬한 후 비용이 적은 우성 인자들을 이용하여 교잡 연산자와 돌연 변이 및 역순 연산자를 수행하였다. 교잡 연산자는 순서 교잡(Order Crossover, OX)를 사용했으며, 돌연변이는 reciprocal exchange mutation을 이용하였고, 역순 연산자는 한 개체내에서 임의의 위치를 정하여 그를 기점으로 도시의 배열 순서를 역으로 바꾸는 것이다.

그리고 외판원 문제의 목적 함수는 간단한 함수를 적용하는데, 여행 경로중에 걸린 비용의 합을 목적 함수 값으로 정하여 이들에 대해 평가한다. 또한 우수한 개체들을 생성하기 위한 선택 방법은 순위 선택 방법으로 목적 함수값을 정렬하여 좋은 값을 가지는 개체들을 선택하여 일정한 비율은 다음 세대로 재생산(reproduction)하고 나머지 개체에서 유전 연산들을 행하여 새로운 자손을 만든다. 자손의 세대는 항상 일정하게 유지한다.

2.2 에이전트

분산 인공지능이라는 분야에서 비롯된 에이전트 개념은 컴퓨터 통신의 보편화와 더불어 현재 가장 중요

한 기술의 하나로 많은 연구가 진행되고 있다. 에이전트란 환경에서 얻은 지식과 자신의 내부 지식을 바탕으로 추론하고, 추론의 결과로 환경에 영향을 미치며 또한, 사용자를 포함한 다른 에이전트와 상호 작용하는 소프트웨어를 말한다. 특히 최근에는 인터넷을 비롯한 네트워크의 발달과 다양한 분산처리 기술의 등장으로 인하여 거대한 정보체계 속에서 원하는 정보를 네트워크에서 추출할 수 있는 분산된 멀티 에이전트에 대한 연구가 활발하게 이루어지고 있다. 이러한 분산 에이전트는 크게 인터페이스 에이전트, 태스크 에이전트, 정보 에이전트의 세 가지 유형으로 나누어서 생각할 수 있으며, 멀티 에이전트는 단독 에이전트가 가질 수 있는 성능의 한계를 극복하고 인터넷의 성장과 더불어 발전하고 있는 전자상거래 및 웹 데이터베이스 등의 분야에서 효율적인 정보 관리 및 추출을 목적으로 한다. 이러한 환경에서는 에이전트가 다른 에이전트와 상호통신을 통하여 정보의 공유 및 교환처리를 수행하고 공동작업을 수행하는 것이 주된 관심사이다. 이를 위해서 네트워크의 이질성을 극복하고 에이전트간의 통신을 위한 각종 프로토콜 및 언어 등에 대한 연구가 수행되고 있으며, 네트워크에서 멀티 에이전트를 이용한 응용 시스템에 대한 연구도 수행되고 있다.

3. 동적인 개체 이동 윈도우

분산 환경에서 다중 에이전트들을 이용해서 수행하는 분산 유전 알고리즘의 구성도는 그림 1과 같이 마스터/슬래이브(master/slave) 모델로 구성된다. 마스터 에이전트는 후보해를 생성해서 슬래이브 에이전트들에게 전송하면 슬래이브 에이전트들은 각 자신의 능력에 따라 분산 유전 알고리즘을 수행하여 얻은 결과를 마스터 에이전트에게 전달한다. 슬래이브 에이전트들로부터 전송된 결과값들을 비교하면서 동적으로 개체 이동 윈도우 크기와 유전 이동 정책을 연산하여 얻은 결과를 각 슬래이브 에이전트들에게 알려 준다. 그러면 슬래이브 에이전트들은 윈도우 크기만큼 개체를 유전 이동 정책 결과에 따라 개체를 이동시킨 후 다시 분산 유전 알고리즘을 수행한다. 이렇게 에이전트간에 개체 이동 정책이 유전 이동 정책을 사용해서 진화 연산에서 얻을 수 있는 세대가 진화함에 따라 더욱 좋은 근사해를 구할 수 있도록 설계하였다. 그리고 개체 이동 크기는 후보해 크기의 10% 이하의 범위에서 개체 이동이 일어날 때마다 크기가 점점 증가하도록 동적인 윈도우를 설계하였다. 예를 들어 처음 시작할 때 개체 이동 윈도우 크기가 1이었다면, 개체 이동이 한번 일어나고 나면 다음에 개체 이동해야 할 윈도우 크기는 2가 되고, 이렇게 동적으로 증가하다 후보해 크기의 10%보다 크게 되면 수행을 멈추게 구성하였다. 이동해야 할 개체는 좋은 개체들을 선택해서 결과가 좋지 않은 개체를 대체하였다. 그리고 개체 이동은 매 세대마다 교환이 일어나는 것이 아니라, 통신 연산을 고려해서 가변적인 세대의 수가 지나고 나서 개체 이동을 하였다.

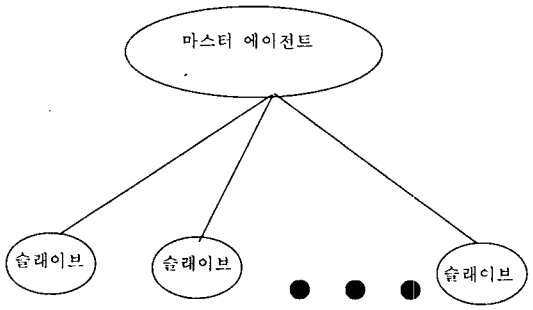

그림 1. 시스템 구성도

4. 에이전트간 유전 이동 정책

앞에서도 설명했지만 다중 후보해를 가진 분산 유전 알고리즘에서 고려해야 할 중요한 요소가 어떤 에이전트들간에 개체의 이동이 일어나야 하는지를 선택하는 것이다. 여기에 대한 많은 연구들이 수행되고 있다. 본 논문에서는 진화 연산이 세대가 변화함에 따라 더욱 좋은 근사해로 수렴하는 것과 같이 개체 이동이 일어나야 할 에이전트를 선정할 때, 유전 연산를 적용하여 에이전트들을 선정하는 2가지 이동 정책, reciprocal 이동과 환형 큐 이동 정책을 개발하였다.

4.1 Reciprocal 이동

유전 연산의 reciprocal mutation 연산과 유사하게 실행되는 연산으로, 마스터 에이전트가 임의의 두 슬래이브 에이전트를 선택하여 그 선택된 슬래이브 에이전트들간에 윈도우 크기만큼 개체 이동이 일어나는 개체 이동 정책이다.

4.2 환형 큐 이동

환형 큐 이동 정책은 후보해 간의 이동이 환형 큐가 연산하는 것과 유사하게 각 슬래이브 에이전트들이 논리적인 링을 형성하여 서로 이웃하는 슬래이브 에이전트간에 윈도우 크기만큼 개체를 이동시키는 정책이다.

5. 실험 방법

본 논문에서 제시한 내용들은 Java를 사용하여 구현하고 있으며, 실험 환경은 LAN으로 연결되어 있는 분산환경을 사용한다. 슬래이브 에이전트의 수를 점차 증가시키면서 수행하는데, 하나의 슬래이브 에이전트의 후보해 크기는 10으로 하였다. 슬래이브 에이전트 간에 이동하는 개체의 윈도우 크기도 변경하면서 실험하였다. 유전 알고리즘에서 사용한 파라미터는 다음과 같다.

표 1 실험에서 사용한 파라미터들

	크기 및 비율
후보해의 크기	가변적(에이전트 당 10)
자손의 세대	가변적
교잡 연산자의 비율	60%
돌연변이 연산자의 비율	10%
역순 연산자의 비율	10%

모든 슬래이브 에이전트에서 사용한 연산의 비율 및 후보해의 크기 그리고 세대수 등 모든 조건은 동일하게 적용하여 수행하였다. 그리고 외판원 문제에 대한 거리의 비용은 외판원 문제를 연구하는 많은 연구자들이 인용하고 있는 TSPLIB[10] 에 있는 값들을 사용하였으며, 일부는 임의로 생성하여 실험하는 중이다. 지금 실험은 윈도우 크기를 1로 시작해서 한번 개체 이동이 있을 때마다 윈도우 크기가 증가하는 실험을 진행중에 있으며, 매 세대마다 개체를 교환하지 않고 통신 연산을 고려하여 10세대를 수행한 후 개체를 교환하도록 실험중이다.

6. 결 론

본 논문에서는 외판원 문제를 분산 환경에서 마스터/슬래이브 모델의 다중 에이전트들을 이용해서 우수한 근접해를 구할 수 있는 분산 유전 알고리즘을 개발하였다. 다중 후보해를 이용한 분산 유전 알고리즘을 수행할 때, 고려되어야 할 요소가 에이전트들 간에 이동해야 할 개체의 크기를 얼마로 해야 할 것인가와, 어떤 방법으로 에이전트를 선택해서 개체들을 이동시키느냐이다. 이에 이동 개체 크기를 동적으로 조정해서 사용할 수 있는 윈도우를 설계하였으며, 개체들을 교환할 위치를 결정하는 새로운 유전 이동 정책 2가지를 개발하고 실험하였다. 앞으로 연구과제는 다양한 실험을 완성해야 하며, 이동해야 할 개체의 크기 곧 최적의 θ를 결정하는 좀 더 효율적인 방법을 모색하는 일이다.

[참고문헌]

[1] Erick Cantú-Paz, "Topologies, Migration Rates, and Multi-Population Parallel Genetic Algorithms", IlliGAL Report No. 99007, January, 1999.

[2] Fazlollahi, B., Vahidov, R.M., Aliev, R.A., "Multi-agent distributed intelligent system based on fuzzy decision making", International Journal of Intelligent Systems, Vol. 15, No. 9, pp. 849-858, 2000.

[3] M. Gen, R. Cheng, Genetic Algorithms and Engineering Design, pp. 118-132, 1997.

[4] D. E. Goldberg, Genetic Algorithms : in Search and Optimization, *Addison-Wesley,* pp. 1-125, 1989.

[5] E. Horowitz, S. Sahni, Computer Algorithms, *Computer Science,* pp. 370-421, 1978.

[6] J. Kim, Y. Hong, "A Distributed Hybrid Algorithm for the Traveling Salesman Problem", *IASTED, AI'99,* Feb. 1999.

[7] Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs, *Springer-verlag,* pp. 209-237, 1995.

[8] G. Reinelt, The Traveling Salesman Computational Solutions for TSP Applications, *Springer-Verlag,* 1994.

[9] T.C. Sapountzis, "The Traveling Salesman Problem", *IEEE Computing Futures,* pp. 60-64, Spring, 1991.

[10] TSPLIB, *http://www.iwr.uni-heidelberg.de/iwr/comopt/soft/TSPLIB95/ATSP.html.*