Journal of KIIT. Vol. 20, No. 2, pp. 37-46, Feb. 28. pISSN 1598-8619, eISSN 2093-7571 
http://dx.doi.org/10.14801/jkiit.2022.20.2.37

# GPU 기반 병렬 Ant Colony System의 구현과 성능 분석

이 윤 석*

# Implementation and Performance Analysis of a GPU-based Parallel Ant Colony System

Yunseok Rhee*

본 연구는 2020년도 한국외국어대학교 교내학술연구비의 지원에 의하여 이루어진 것임


## 요 약

Ant Colony System은 계산 복잡도가 높은 조합 문제의 근사 최적해(near optimal solution)를 구하는 메타 휴리스틱 기법이다. ACS가 갖는 막대한 계산량으로 인해 알고리즘의 효과적인 병렬화가 요구된다. 본 논문에서는 GPU가 제공하는 데이터 병렬성을 최대한 활용한 알고리즘을 제안하고, 이를 Traveling Salesman Problem에 적용하여 그 성능을 분석하였다. 특히 데이터 병렬성을 높이기 위해 병렬 쓰레드의 수와 쓰레드 블록의 수를 최대화하고, 이들의 연속 메모리 동시 접근 효과를 충분히 활용하였다. 또한 효과적인 TSP 해결을 위해 노드 간 거리를 이용한 근접성을 활용해 최적해를 탐색하는 과정을 제안한다. 본 실험은 Nvidia Titan RTX GPU와 Intel i9-9900K를 사용하여 구현하고 공개된 주요 TSPLIB 데이터를 대상으로 실험을 통해 성능 개선 효과를 확인했다.

## Abstract

Ant Colony System is a metaheuristic technique that seeks a near optimal solution for a combinatorial problem with high computational complexity. Effective parallelization of algorithms is required due to the enormous amount of computation of ACS. In this paper, we propose an ACS algorithm that makes the most of data parallelism provided on GPU, and analyze its performance with the Traveling Salesman Problem to In particular, in order to increase data parallelism, the number of parallel threads and thread blocks were maximized, and their cocurrent continuous memory accesses were fully utilized. In addition, for effective TSP resolution, a process of searching for optimal solutions using proximity by distance between nodes is proposed. This experiment was implemented using Nvidia Titan RTX GPU and Inteli9-9900K and showed the effectiveness of performance improvement through experiments on public TSPLIB data.


Keywords
ant colony system, traveling salesman problem, graphic processing unit, metaheuristic, parallelization

* 한국외국어대학교 컴퓨터공학부 교수
- ORCID: http://orcid.org/0000-0002-0824-6852

· Received: Jan. 26, 2022, Revised: Feb. 12, 2022, Accepted: Feb. 15, 2022
· Corresponding Author: Yunseok Rhee
Division of Computer Engineering, Hankuk University of Foreign Studies, Oedae-ro 81, Mohyeon, Yongin 17035, Korea
Tel.: +82-31-330-4259, Email: rheeys@hufs.ac.kr

## Ⅰ. 서 론

ACS(Ant Colony System)는 Dorigo와 Gambardella 등에 의해 제안된 방법으로 수많은 조합에서 최적해를 찾아내는 메타휴리스틱(Metaheuristic) 기법이다[1]. 이 기법은 특정한 문제를 해결하는 정교한 알고리즘을 기술하는 대신, 문제를 해결하는 전략을 제시하고 이를 반복적으로 적용한다[2]. 전통적인 방법에 비해 메타휴리스틱 기법은 최적해를 보장하지는 않지만, 일정한 시간 내에 충분히 좋은 근사 최적해를 구하는 장점이 있다. 또한 이 기법은 특정 문제에 국한되지 않고 과학 및 공학분야의 다양한 최적화 문제들에 비교적 쉽게 적용될 수 있다.

TSP(Traveling Salesman Problem)는 모든 도시를 오직 한번씩만 방문하여 최단 경로를 구하는 최적화 문제로서, 컴퓨터 네트워크, 센서 배치, 차량 라우팅, 물류 분배, 회로도 배선 등의 다양한 문제에 널리 활용된다[3]. TSP의 계산복잡도는 NP-hard로 잘 알려졌고 대규모 데이터에 대해 전통적인 최적화 알고리즘이 효과적이지 않아서 ACS와 같은 메타휴리스틱 알고리즘이 자주 활용된다.

유전자 알고리즘이나 시뮬레이티드 어닐링 기법 등도 TSP의 근사 최적해를 구하는데 많이 활용되고 있지만[4][5], 대부분의 기존 연구들은 문제 해결 과정에서 네트워크 상황이 변하지 않는 정적 TSP를 다룬다. 그러나, 현실 상황의 TSP는 도로의 교통량 변화와 같은 동적인 환경을 반영하게 되는데, 차량 라우팅 문제 등에서 같은 동적 TSP에 ACS가 매우 효과적인 것으로 알려져 있다[6][7]

ACS 기법은 본래 수많은 개체를 생성하고, 이들의 협업을 통해 최적해를 찾아가는 과정을 설계한 것으로, 이 협업 과정이 병렬화에 매우 적합한 구조를 갖는다. 따라서, 네트워크 기반 다중 컴퓨터 상에서 MPI(Message Passing Interface)를 이용한 병렬화하거나 OpenMP를 이용해 다중 프로세서에서 ACS를 구현하고 TSP를 해결하는 병렬화 연구들이 있었고[8]-[10], 최근 GPU(Graphic Processing Unit)를 이용한 병렬화 연구들이 수행되었다[11]-[13]

GPU 기반 병렬화가 이 문제에 적합한 이유는 네트워크를 구성하는 각 노드에서 인접한 노드에 대해 경로 선택 적합성(Fitness)을 계산하는 과정이나 에이전트인 개미가 방문한 경로에 대해 페로몬의 영향을 업데이트하는 과정들이 모든 노드에서 SIMD(Single Instruction Multiple Data) 형태의 동일한 계산과정이 GPU의 SIMD 다중 코어에서 효과적으로 수행될 수 있기 때문이다.

2장에서는 TSP 해결을 위한 ACS 알고리즘을 소개하고 3장에서 본 논문에서 구현한 GPU 기반 병렬 알고리즘을 각 부분별로 상세 기술한다. 그리고 4장에서 CPU 구현 알고리즘 대비 성능 개선 결과와 ACS의 반복 실행 횟수, 근접성 적용, 병렬화 정도에 따른 실험 결과를 분석하고 5장에서 결론을 맺는다.

## Ⅱ. TSP에 적용된 ACS 알고리즘

ACS 알고리즘을 TSP에 적용하기 위해 개미라고 불리는 다수의 에이전트들을 생성하고, 이들을 통해 주변 경로를 탐색하는 과정을 반복해 간다.

특히 ACS 알고리즘에서는 개미가 이동하면서 각 에지에 페로몬 흔적을 남기는 계산 모델을 통해 개미 간의 정보 교환을 지원하며, 궁극적으로 비교적 좋은 경로를 개미들이 선호하도록 유도한다. 이를 위해 도시 그래프의 어느 간선 $e_{i,j}$에 남긴 페로몬의 값을 $\tau_{i,j}$로 표현하고 $d_{i,j}$는 두 도시 $v_i$와 $v_j$ 간의 거리를 나타낸다.

우선 TSP 데이터를 사용하여 ACS의 개미의 위치, 페로몬 값 등을 초기화한다. 문제 설정 파라미터로서, 일반적으로 개미의 수(num_ants)는 도시의 수(num_nodes)와 같거나 그보다 적은 수로 설정하고, 그 위치는 랜덤하게 배치해 왔다. 그러나, 본 논문에서는 GPU의 고병렬성을 활용하여 도시의 수보다 더 많은 개미들을 생성하여 문제를 해결할 수 있다.

이제 생성된 개미들이 찾아낸 탐색 경로 정보를 교환하고 최적값을 업데이트하는 다음의 과정을 일정한 횟수만큼 반복해 간다. 이때 반복 횟수는 상수로 고정될 수도 있고 동일한 결과가 일정 횟수 반복될 경우 종료하는 조건을 설정할 수도 있다.

각 개미는 자신의 현재 도시에서 연결된 인접 도시들에 대해 확률적 선택을 통해 다음 방문도시를 선택하고, 모든 도시를 방문할 때까지 이와 같은 과

정을 반복하여 하나의 여정(Tour)을 생성하게 된다.

ACS에서 다음 도시의 선택 적합도는 아래 식 (1)과 같이 페로몬의 양($\tau_{i,j}$)에 비례하고 도시 간 거리($d_{i,j}$)에 반비례한다. 여기에, 이 두 값의 영향력을 통제하기 위해 적당한 양수 $\alpha$, $\beta$를 사용하여 다음과 같이 적합도 $f_{i,j}$, 즉 도시 $v_i$에서 $v_j$를 선택하는 확률적 적합도를 구한다.

$$f_{i,j} = (\tau_{i,j})^{\alpha}(1/d_{i,j})^{\beta} \quad (1)$$

또한, 각 인접 도시 $v_j$에 대해 얻은 $f_{i,j}$를 모두 합한 값 $F$를 구한 후, 이를 사용하여 다음 식 (2)와 같이 각 도시에 대한 확률을 설정한다.

$$F = \sum_{j} f_{i,j} \quad (2)$$
$$p_{i,j} = f_{i,j}/F$$

이렇게 구한 각 확률을 바탕으로 확률값에 비례한 면적들로 구성된 룰렛 원판(Roulette-wheel)에서 임의 위치를 선택함으로써 다음 방문지를 선택한다. 이 방법을 통해 적합도가 높은 도시가 확률적으로 선택 가능성이 높지만 이것이 항상 선택되는 것은 아니다.

각 개미가 생성한 경로들을 현재까지 구해진 최단 경로와 비교하여 업데이트한다. 이때 새롭게 선택된 global_best 경로의 간선에 대해 전체 경로의 길이에 반비례, 즉 $1/d_{i,j}$ 만큼 페로몬을 증가시킨다.

또한 ACS에서는 이미 경로 상에 배출된 페로몬은 시간이 지남에 따라 증발하여 그 양이 다음과 같이 점차 약화된다고 가정한다. 이때 $\rho$는 실험에 의해 결정된 증발율 상수이고 대개 0.5로 설정된다.

$$\tau_{i,j} = (1-\rho)\tau_{i,j} \quad (3)$$

앞서의 페로몬 증감 조건에 따라서 다음 순간 (t+1)에 페로몬의 양은 식 (4)와 같이 결정된다.

$$\tau_{i,j}(t+1) = (1-\rho)\tau_{i,j}(t) + 1/d_{i,j} \quad (4)$$

## III. GPU-기반 병렬 ACS 알고리즘

### 3.1 GPU 병렬구조와 메모리 접근

GPU는 데이터 병렬성을 지원하는 아키텍쳐로서 ACS 알고리즘과 같이 에이전트들이 서로 다른 데이터에 대해 동일한 연산을 수행하는 형태의 알고리즘에 적합한 구조이다. 특히, TSP 해결에 필요한 인접 도시의 적합도 판정(Fitness calculation), 여정 생성(Tour construction), 페로몬 업데이트(Pheromone update) 등은 대규모 행렬 데이터에 대한 GPU 병렬처리가 성능의 관건이다.

GPU 장치는 다수의 SM(Streaming Multiprocessor)로 구성되고, 각 SM은 내부에 SIMD 형태로 실행되는 많은 수의 stream processor(또는 core)를 갖는다. 특히 GPU 메모리 시스템은 전역 메모리(Global memory)와 공유 메모리(Shared memory)의 계층 구조를 갖는데, 전역 메모리는 CUDA 커널의 모든 쓰레드들이 접근 가능한 메모리이나 매우 느린 반면, 공유 메모리는 각 쓰레드 블록(Thread block) 내의 모든 쓰레드가 공유하는 메모리로 빠른 접근시간을 지원한다.

따라서, 데이터 병렬성이 높은 알고리즘을 GPU에서 구현할 때 전역 메모리 접근 횟수를 줄이고, 공유 메모리를 적극 활용하는 것이 필요하다. 특히 GPU에서 전역 메모리 접근이 차지하는 오버헤드가 매우 크므로 같은 블록 내의 쓰레드들이 전역 메모리의 연속된 주소들에 동시에 접근할 때 GPU에서는 이를 한꺼번에 처리하여 메모리 접근의 대역폭을 최대한 활용하는 병합 접근(Coalesced access)을 지원하도록 코드를 구성한다.

본 논문에서는 Nvidia에서 제공하는 CUDA (Compute Unified Device Architecture) 프레임워크를 사용해 병렬 프로그램을 개발했다. CUDA는 다수의 쓰레드들의 집합인 쓰레드 블럭의 개념을 지원하는데, 한 쓰레드 블럭은 한 개의 SM에 배치되고, 한 쓰레드는 그 안에서 하나의 코어에서 실행된다.

한편 SM 내에서는 각 쓰레드 블럭을 구성하는 t 쓰레드들을 워프(Warp)라고 부르는 32개의 쓰레드 단위로 나누어 이들을 차례로 스케줄링을 한다. 따

라서, 각 SM에 충분한 블럭들이 배치되지 않는 경우 자연히 스케줄링 워프의 개수도 적어지게 되어 SM 자원을 효과적으로 활용하지 못하는 문제가 나타나므로, 프로그램 개발자는 GPU의 전체 SM에 고르게 많은 쓰레드 블럭들이 생성되도록 알고리즘을 설계해야 한다.

## 3.2 병렬 ACS 알고리즘 설계

본 연구에서 제안하는 알고리즘은 여정 초기화(Tour initialization), 방문 적합도 계산(Visit fitness calculation), 여정 구성(Tour construction), 여정 정리(Wrap-up tour), 페로몬 업데이트(Pheromone update)의 5개 커널로 구성되고 다음에 각각 상세히 소개된다.

ACS 알고리즘은 만족할 만한 최적해가 구해질 때까지 이 커널들을 반복 실행함으로써 TSP의 최단 경로를 계속 업데이트하게 된다. 본 연구에서 사용한 도시 그래프에서는 임의의 두 도시 간에 경로가 존재한다고 가정한다. 즉, 어떤 도시 $v_i$로부터 임의의 다른 도시 $v_j$로 가는 경로가 존재하며 이 커널은 각 도시 $v_i$의 위치가 $(x_i, y_i)$로 표현된 $n$개의 도시에 대해 임의의 구간 거리 $d_{i,j}$ ( > 0)를 구하고 해당 구간의 페로몬 $\tau_{i,j}$를 적당한 초기 상수(여기서는 $1/n$)으로 설정한다.

(1) 여정 초기화 커널

각 개미별 초기 도시 위치를 정하고 여정의 경로 방문 정보와 경로 길이를 초기화한다. 본 구현에서는 블록 크기(=블럭 내 쓰레드의 수)를 32개로 설정하고 개미의 수($n_{ant}$)를 32로 나눈 수, 즉 $\lceil n_{ant}/32 \rceil$ 개의 블록들을 생성하는 CUDA 커널을 작성한다.

(2) 방문 적합도 계산 커널

임의의 도시 $v_i$에서 나머지 모든 도시 $v_j$에 대해 방문 적합도(fitness)를 다음과 같이 계산한다. 이 식에서 $\alpha, \beta$는 페로몬과 거리의 영향력을 각각 통제하기 위한 상수이다.

$$fitness = (\tau_{i,j})^{\alpha}(1/d_{i,j})^{\beta}$$

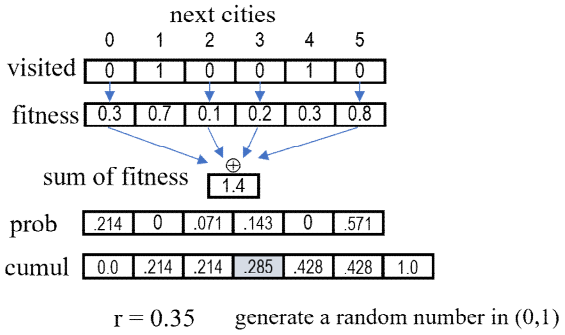



그림 1. 다음 방문도시 선정 과정 예
Fig. 1. Selection process example for a next visiting city

이 커널은 전체 도시 행렬 $n \times n$에 대해 각 $32 \times 32$ 크기의 블록을 $\lceil n/32 \rceil \times \lceil n/32 \rceil$ 개 병렬 생성한다. 이 때 생성된 블록의 각 쓰레드는 자신이 담당한 도시 구간 $(i,j)$에 대해 전역 메모리의 배열에서 해당 페로몬 값 $\tau_{i,j}$와 구간 거리 $d_{i,j}$를 읽어 위의 식을 계산하므로 자연스럽게 메모리 병합 접근이 활용될 수 있다.

(3) 여정 구성 커널

각 개미 쓰레드는 자신이 탐색할 여정을 확률 과정(Stochastic process)을 통해 구성한다. 그림 1은 6개의 인접 도시에서 1개를 선택하는 과정을 예로 보여준다. 우선 visited 배열은 각 도시에 대해 이미 방문했는지 여부를 기록하고, fitness 배열은 (2)의 과정을 통해 각 도시의 선택 적합도(또는 강도)를 나타낸다.

선택 과정에 앞서 visited, fitness 배열을 통해 각 도시에 대한 선택 확률을 구한다. 이를 위해 이미 방문한 도시는 제외하고, 나머지 도시들의 fitness 값의 합한 후, 이 합계(sum of fitness)에 대한 각 fitness의 비율을 해당 도시의 선택 확률(prob)을 계산한다. 당연히 이미 방문한 도시의 확률은 0이다. 그리고, 다음과 같이 누적확률분포(Cumul)를 구한다.

$$cumul[i] = cumul[i-1] + prob[i-1] \quad (5)$$

다음 단계에서 (0, 1) 사이의 난수를 생성하고, cumul 배열에서 이 값이 속하는 구간을 구해 다음

도시를 결정하게 된다. 이 때 난수 생성 함수는 CUDA가 제공하는 cudaRandom 함수를 사용하는데, 이 함수는 각 개미 쓰레드마다 서로 다른 seed를 설정하고 이를 curandState 객체를 통해 관리한다.

현재 구현한 알고리즘은 개미별로 여정을 생성하기 때문에 개미의 수가 병렬성을 제한한다. 또한, 각 여정 생성 시 다음 도시를 선택하는 과정이 순차적으로 실행됨으로써 병렬 효과를 기대하기 어려운 문제가 있다. 그럼에도 현재 각 블록을 32개의 쓰레드로 구성하여 전체 개미들을 블록으로 나누어 구성함으로써 GPU의 블록 점유도(Block occupancy)를 비교적 높게 유지했다. 이에 반해, 만일 한 여정의 병렬화를 극대화하기 위해 도시의 수만큼의 쓰레드로 구성된 블록을 생성하는 경우, 이 블록이 하나의 SM(Streaming Multiprocessor)의 스케쥴 자원을 과도하게 사용하여 오히려 성능이 크게 느려짐을 확인할 수 있었다.

(4) 여정 정리 커널

앞서의 (2)에서 얻어진 각 여정의 길이(여정에 속한 경로의 합)는 그때까지 발견된 최단 경로와 비교되고 더 짧은 경로일 경우 갱신된다. 이 과정 역시 최단 경로를 저장한 전역 변수에 대한 모든 쓰레드의 동시 접근이 이루어지므로, 이의 동기화를 제어하기위해 CUDA에서 제공하는 atomicMin 함수를 사용한다. 그러나, 이 함수의 사용 시 쓰레드의 병행성(Concurrency)이 낮아져 병렬화에 의한 성능 향상을 제약하고 있다.

또한, 이번에 이동한 여정을 구성하는 각 경로에 포함된 구간 $(v_i, v_j)$에 페로몬의 증가량을 기록하는 2차원 배열 원소 $\delta_{i,j}$에 기록한다. 이 커널 역시 각 블록을 32개의 쓰레드로 구성하고 전체 개미들을 이 크기의 블록들로 나누어 구성하였다.

(5) 페로몬 업데이트 커널

여정 정리 커널을 통해 각 개미가 선택한 여정에 대한 평가가 이뤄지면, 전체 경로에 대해 페로몬 영향력을 업데이트해야 한다. 여기에는 크게 (i) 시간이 지남에 따른 페로몬의 감소와 (ii) 앞서 각 개미가 선택한 경로에 더해진 페로몬의 양($\delta[i][j]$)을 다음과 같이 반영한다.

(i) $\tau_{i,j} = (1-\rho)\,\tau_{i,j}$ ($\rho$는 페로몬의 증발율 상수)
(ii) $\tau_{i,j} = \tau_{i,j} + \delta_{i,j}$

이 커널 역시 전체 도시 행렬 $n \times n$에 대해 각 $32 \times 32$ 크기의 블록을 $\lceil n/32 \rceil \times \lceil n/32 \rceil$개 병렬 생성한다. 이 때 생성된 블록의 각 쓰레드는 자신이 담당한 도시 구간 $(i,j)$에 대해 전역 메모리의 배열에서 해당 페로몬 값 $\tau_{i,j}$와 구간 페로몬 증가량 $\delta_{i,j}$를 읽어 위의 식을 계산하므로 병합 접근이 효과적으로 활용될 수 있다.

### 3.3 근접성을 이용한 ACS 제안

(1) 도시 간 평균 거리와 편차를 구하는 커널

일반적인 ACS 알고리즘에서 여정 생성 시 다음 도시는 확률적 사건으로 선택한다. 그러나, 밀집된 도시 분포 그래프의 경우, 인접 도시는 대개 일정 거리 내에 위치할 확률이 매우 높다. 따라서, 본 연구에서는 다음 도시를 일정 거리 내에서 우선 선택하도록 알고리즘을 수정 구현하였다.

이를 위해 모든 도시 간 거리를 합하고 이의 평균($\mu$)과 표준편차($\sigma$)를 구하는 커널을 작성한다. 이 때 $n \times n$ 도시 행렬에 대해 도시 간 거리 행렬 D를 구한 후, 쓰레드 i가 도시 i로부터 모든 다른 도시 j간의 합을 구하는 작업 병렬성을 이용하는 쓰레드 블록을 생성한다. 이 때 각 쓰레드는 자신이 구한 local_sum을 atomicAdd를 사용해 global_sum에 더하고, 최종적으로 한 쓰레드가 평균값을 계산한다. 블록의 크기를 32로 설정하고 도시의 수($n$)를 32로 나눈 수, 즉 $\lceil n/32 \rceil$개의 블록들을 생성하는 커널을 작성한다. 편차를 구하는 커널 역시 이와 동일한 방법을 사용하며, 각 쓰레드는 계산된 평균과 각 도시 간 거리의 차와 그 제곱의 합을 구한 후 종적으로 편차를 계산한다.

(2) 여정 생성 시 근접성을 활용하는 커널

여정 생성을 위해 앞서 3.2절의 (3)에서 기술한 확률 과정을 동일하게 사용된다. 다만, 근접성을 활용하기 위해, 현재 도시 i에서 확률 시행을 통해 선택된 도시 j와의 거리 $d_{i,j}$의 값을 살펴보고 근접 조건을 확인한다.

첫 시행에서의 근접 조건 $|d_{i,j}-\mu| \leq \sigma$인 경우 해당 도시 j를 다음 선택지로 채택한다. 만일 근접 조건을 만족하지 않을 경우 재시행하고, 2차 근접 조건 $|d_{i,j}-\mu| \leq 2\sigma$를 만족하면 다음 선택지로 채택한다. 만일 2차 근접 조건을 만족하지 않는 경우, 아직 방문하지 않은 도시 중 가장 큰 fitness를 갖는 도시를 다음 방문지로 선택한다.

## Ⅳ. 실험 및 성능 분석

본 절에서는 GPU 기반 병렬 알고리즘의 성능을 살펴보기 위해 우선 본래 ACS 알고리즘의 CPU 버전에 대한 성능을 살펴보고, 개미의 수에 따른 실행 시간과 구해진 해답의 품질을 살펴본다. 그리고, GPU 실행 시간에서 각 부분이 차지하는 실행 시간의 비율을 분석한다.

CPU에서 실행될 순차 ACS 알고리즘은 C 언어로 구현된 공개 코드를 참고하여 구현하였으며[14], GPU 병렬 알고리즘은 Nvidia에서 제공하는 CUDA library(version 7.5)를 사용하여 Windows Visual Studio 2019 환경에서 개발하였고, 표 1에 보인 하드웨어 시스템에서 실험하였다.

ACS 알고리즘의 동작을 통제하는 파라미터는 관련 연구에서 제시한 값들을 참조하여 다음과 같이 설정하였다. 페로몬과 거리에 따른 선택 적합도를 제어하는 $\alpha$, $\beta$는 각각 0.2와 3.0, 페로몬의 증발율 상수 $\rho$는 0.01로 설정하였다.

표 1. 실험에 사용된 시스템 사양
Table 1. System specification for experiments

| | CPU system | GPU system |
|---|---|---|
| Processor | Intel core i9-9900K | Nvidia titan RTX (Turing architecture) |
| Clock | 3.6 GHz | 1.750 GHz |
| # Cores | 8 | 4608 (72 SMs x 64 cores/SM) |
| L1 cache | 64 KB (per core) | 64 KB (per SM) |
| L2 cache | 256 KB (per core) | 6 MB |
| Memory bandwidth | 41.6 GB/s | 672 GB/s |
| Memory capacity | 64 GB (on-board) | 24 GB (on-device) |
| Performance | 49.45 GFLOPS | 16312.32 GFLOPS |

또한, 실험에 사용된 TSP 맵은 TSPLIB[15]으로부터 bier127, lin318, ali535, pr1002 등 4개 데이터를 선택하였고(각 데이터 이름에 표시된 숫자는 도시의 수), 개미의 수는 도시의 수와 같은 수로 설정하였다.

### 4.1 병렬화 효과

ACS를 이용한 TSP 알고리즘은 여러 차례의 반복 실행을 통해 근사 최적해를 구하지만, 알고리즘의 성능 비교를 위해서는 1회 반복에 소요된 실행 시간을 측정하였다.

표 2에 보이는 바와 같이, bier127을 제외하고 대체로 10.13~11.37배의 성능 향상을 보였는데, 이는 도시 간 거리 정보와 페로몬 잔량을 저장하는 $N_{city} \times N_{city}$ 행렬에 대해 32×32 병렬 블록에 의한 데이터 분산처리 효과와 연속 메모리 공간에 대한 병합 접근 효과로 분석된다.

다만, bier127의 경우 4.82배의 비교적 낮은 성능 향상을 보인 것은 앞서 언급한 병렬처리 효과에 비해 여정 생성 과정에서 다음 방문 도시를 선정하는 순차적 과정이 차지하는 비중이 상대적으로 높고, 페로몬 업데이트 과정에서 사용한 atomicMin 함수가 상당한 비용을 요구하기 때문으로 분석된다.

표 2. 병렬화 성능
Table 2. Parallelization performance [time unit: ms]

| Data | CPU | GPU | Speedup |
|---|---|---|---|
| bier127 | 9.02 | 1.87 | 4.82 |
| lin318 | 23.84 | 2.35 | 10.13 |
| ali535 | 42.72 | 3.91 | 10.92 |
| pr1002 | 209.12 | 18.39 | 11.37 |

### 4.2 근접성을 적용한 실험 결과

앞서 제안한 대로 ACS 알고리즘에 노드 간 근접성을 활용한 버전을 구현하고 실험하였다. 실험 방법은 (1) 근접성에 따른 재시도가 없는 방법(No retry), (2) 1차 근접성(평균과의 차이 $\sigma$ 이하)을 확인 후 1회 재시도를 하는 방법 (1 retry), (3) 1차 근접성과 2차 근접성(평균과의 차이 $2\sigma$ 이하) 확인

후 재시도하는 방법(2 retries)으로 나누어 실험하였다. 실험 데이터로는 노드 수와 노드 간 분포의 복잡도가 중간 정도인 TSPLIB의 lin318 데이터를 선정하고, 반복 횟수는 30회로 제한하였다.

3가지 방식에서 두드러진 차이는 초기 결과값의 업데이트 속도이다. 근접성을 적용하지 않은 경우에 매우 큰 값에서 초기해가 시작되고, 전체적으로 근접성을 강화한 알고리즘의 초기해가 더 작은 값부터 시작해 최적해를 향해 빠르게 수렴한다.

또한, 그림 2의 결과에 보이는 바와 같이 근접성을 고려한 경우에 최종 최단경로(근사 최적해)가 더 우수함을 볼 수 있다. No retry의 경우에 최종 결과는 45038, 1 retry의 경우는 43421, 2 retries의 경우는 42857로 각각 나타났다. 다만, 3 방법 모두에서 15회 반복 이후에는 최단경로가 갱신되지 않는 한계를 보인다.

그림 3은 3 방법에 의해 각각 구해진 최종 결과의 경로 연결 상태를 그림으로 비교한 것이다. 근접성을 고려하지 않은 경우 먼 거리의 노드 간 연결로 인해 최단 경로 탐색이 제약을 받는데 반해, 근접성을 고려한 경우 인근 노드로의 우선 연결로 점진적인 최단 경로 탐색이 이뤄짐을 볼 수 있다.

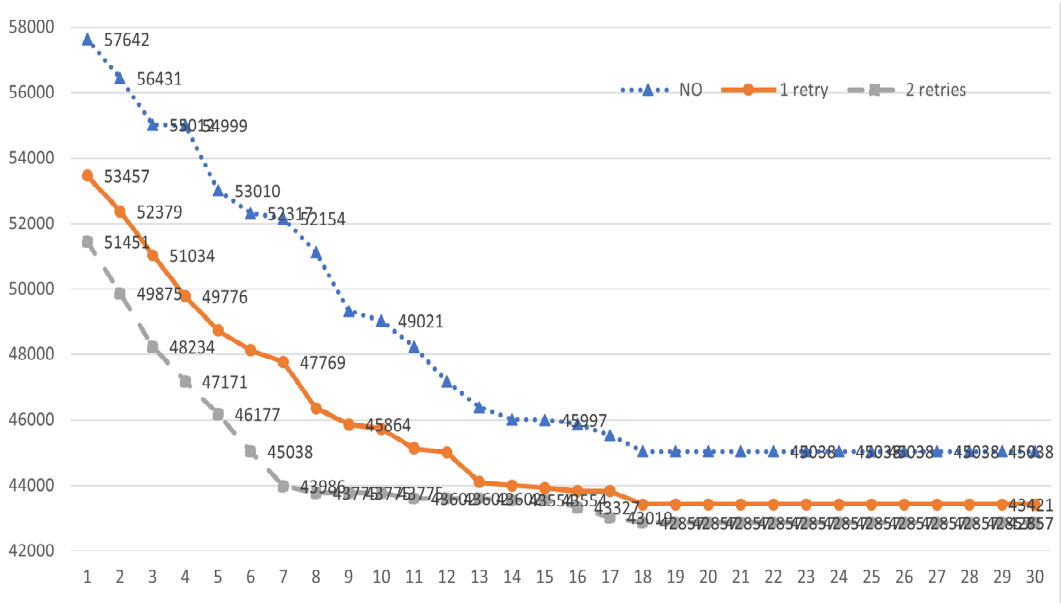
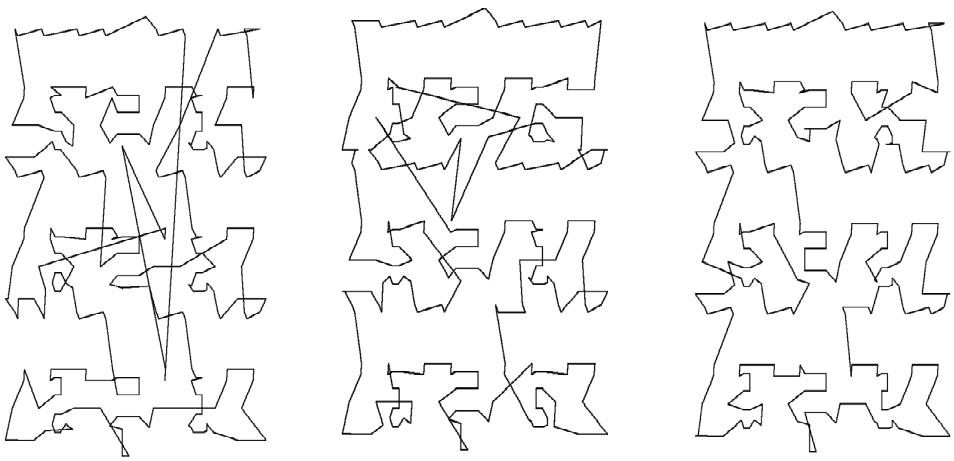



그림 2. 근접성을 활용한 실험 결과
Fig. 2. Experiment results by exploiting proximity



(a) No retry (final: 45038) (b) 1 retry (final: 43421) (c) 2 retries (final: 42857)

그림 3. 근접성을 활용한 최종 결과 비교
Fig. 3. Comparison of final results by exploiting proximity

### 4.3 개미의 수에 따른 성능 비교

개미의 수, 즉 병렬처리 쓰레드의 수에 따른 성능을 살펴보기 위해 도시의 수와 동일한 수의 개미와 2배수, 4배수, 8배수의 개미를 각각 생성하는 실험을 앞서의 4 데이터에 대해 수행하고, 각 개미의 수에 대해 동일한 실험을 10회 반복하였다.

이전의 ACS 연구에서는 도시의 수 이하의 개미를 생성하여 실험했지만, GPU의 고성능으로 수십만 개 이상의 병렬 쓰레드를 생성하는 것이 가능하다.

표 3의 결과는 10회 실행에서 best 결과와 평균 결과를 보여준다. 전체적으로 개미의 수를 증가시키는 것이 근사 최적해의 품질을 높이는데 기여한 것으로 나타났다. 또한 개미의 수가 많을 수로 빠른 속도로 최적값을 향해 수렴해 가는 경향을 보였다. 이것은 높은 병렬성이 순차적 반복 계산을 일정 부분 보상하는 것으로 분석된다.

한편 bier127과 lin318과 같이 규모가 크지 않은 데이터의 경우 4배수와 8배수 개미를 각각 투여한 결과는 근소한 차이를 보였다. 이는 주어진 데이터 크기에 비해 일정 정도 이상의 병렬성은 성능 개선에 한계를 갖는 것으로 보인다.

표 3. 개미의 수에 따른 실험 결과
Table 3. Experiment results with the number of ants

| | Ants | Result | | Execution time (ms) |
|---|---|---|---|---|
| | | Best | Average | |
| bier127 | x 1 | 123406 | 125712 | 56.2 |
| | x 2 | 121535 | 122313 | 78.7 |
| | x 4 | 119828 | 121172 | 109.1 |
| | x 8 | 119124 | 120314 | 220.9 |
| lin318 | x 1 | 45232 | 46754 | 73.8 |
| | x 2 | 45205 | 45874 | 91.2 |
| | x 4 | 44661 | 45279 | 143.7 |
| | x 8 | 43632 | 43928 | 302.3 |
| ali535 | x 1 | 212337 | 213235 | 121.3 |
| | x 2 | 209825 | 212328 | 227.3 |
| | x 4 | 206331 | 207102 | 454.6 |
| | x 8 | 202339 | 204113 | 1002.5 |
| pr1002 | x 1 | 272145 | 273207 | 597.2 |
| | x 2 | 268827 | 269132 | 1203.3 |
| | x 4 | 263224 | 265321 | 2507.3 |
| | x 8 | 261353 | 262748 | 5012.6 |

이상의 실험을 통해 ACS 알고리즘이 일정한 품질의 결과를 얻기 위해 적어도 데이터 크기에 상응하는 개미와 일정 횟수 이상의 반복 계산이 절대적임을 확인할 수 있다. 특히 일정 수준 이상의 병렬처리 성능이 제공되어야 고품질의 결과를 보장받을 수 있다는 점은 CPU 기반으로 ACS를 구현하는 것이 성능 면에서 한계에 다다랐음을 보여준다. 더욱이 수백 또는 수천 개의 데이터에 대해 ACS 알고리즘을 수행할 때, CPU에서 개미의 수를 그 수준으로 생성, 활용한다는 것은 그 수 만큼의 반복 연산을 일으켜 결국 GPU 기반 병렬 알고리즘과 성능 차이는 더욱 크게 벌어질 것으로 예상된다.

## V. 결론 및 향후 과제

본 논문에서는 GPU를 기반으로 ACS 기법을 병렬화하고 이를 TSP에 적용하여 성능 개선 효과를 분석하였다. 특히 TSP 해결 방법으로 노드 간 거리를 이용한 근접성을 활용함으로써 보다 효과적으로 근사 최적해에 도달하는 과정을 보였다. 비교 실험을 위해 Nvidia Titan RTX GPU 상에서 병렬 알고리즘을 구현하고, 순차 알고리즘은 Intel i9-9900K CPU 상에서 구현되었다.

공개된 주요 TSP 데이터를 대상으로 실험한 결과, 단일 CPU에서 동작하는 알고리즘에 비해 GPU 병렬 알고리즘은 약 10.13~11.37배의 성능 개선 효과를 보였다. 다만, bier127와 같이 비교적 작은 규모의 데이터에 대해 4.82배의 비교적 낮은 성능 향상을 보인 것은 여정 생성 과정에서 다음 방문 도시를 선정하는 순차적 과정이 차지하는 비중이 상대적으로 높았던 것으로 보인다. 또한, 근접성을 이용한 알고리즘의 경우 인근 노드로의 점진적 경로 탐색을 통해 더욱 빠르게 최적해를 향해 수렴하는 것을 확인했다.

한편 개미의 수에 따른 실험 결과를 통해 일정한 품질의 결과를 얻기 위해 ACS 알고리즘은 적어도 데이터 크기에 상응하는 병렬성과 일정 횟수 이상의 반복 계산이 절대적임을 확인했고, GPU 구조는 ACS의 병렬화에 최적화된 구조임을 확인했다.

특히 대규모 TSP 맵 데이터에서 다음 방문 도시의 확률적 선택 적합도를 계산하거나 개미들의 움

직임에 따른 페로몬의 자취를 업데이트하는 과정은 GPU의 데이터 병렬구조에 아주 적합한 계산과정으로 성능 개선에 주요한 부분이었다. 다만, 개미들이 매번 다음 여정을 생성하는 과정은 순차적 확률 과정의 연속으로 병렬화에 한계가 있었다. 따라서, 향후 연구에서는 이 부분의 알고리즘 속성을 세분화하고 더 적극적으로 병렬화 방법을 적용할 필요가 있다.

## References


[1] M. Dorigo, "Optimization, learning and natural algorithms", Ph.D. thesis, Dipartimento di Elettronica, Politecnico di Milano, 1992.

[2] M. Dorigo, V. Maniezzo, and A. Colorni, "The ant system: Optimization by a colony of cooperating agents", IEEE Transactions on Systems, Man, and Cybernetics-Part B, Vol. 26, No. 1, pp. 29-41, Feb. 1996. https://doi.org/10.1109/3477.484436.

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, "Introduction to Algorithms", 2nd ed., The MIT Press, 2001.

[4] J. Kanda, A. D. Carvalho, and E. Hruschka, "Meta-learning to select the best meta-heuristic for the Traveling Salesman Problem: A comparison of meta-features", Neurocomputing, Vol. 205(C), pp. 393-406, Sep. 2016. https://doi.org/10.1016/j.neucom.2016.04.027.

[5] X. Chen, Y. Zhou, and Z. Tang, "A hybrid algorithm combining glowworm swarm optimization and complete 2-opt algorithm for spherical travelling salesman problems", Applied Soft Computing, Vol. 58, pp. 104-114, Sep. 2017. https://doi.org/10.1016/j.asoc.2017.04.057.

[6] C. Groba and A. Sartal, "Solving the dynamic traveling salesman problem using a genetic algorithm with trajectory prediction: an application to fish aggregating devices", Computers & Operations Research, Vol. 56, pp. 22-32, Apr. 2015. https://doi.org/10.1016/j.cor.2014.10.012.

[7] Michalis Mavrovouniotis, Shengxiang Yang, Mien Van, Changhe Li, and Marios Polycarpou, "Ant Colony Optimization Algorithms for Dynamic Optimization: A Case Study of the Dynamic Travelling Salesperson Problem", IEEE Computational Intelligence Magazine, Vol. 15, No. 1, pp. 52-63, Feb. 2020. https://doi.org/10.1109/MCI.2019.2954644.

[8] M. Manfrin, M. Birattari, T. St¨utzle, and M. Dorigo, "Parallel Ant Colony Optimization for the Traveling Salesman Problem", in Proc. of5th International Workshop on Ant Colony Optimization and Swarm Intelligence, Vol. LNCS 4150, Springer-Verlag, Brussels, Belgium, pp. 224-234, Sep. 2006.

[9] P. Delisle, M. Krahecki, M. Gravel, and C. Gagn´e, "Parallel implementation of an ant colony optimization metaheuristic with OpenMP", in Proc. of the 3rd European Workshop on OpenMP, pp. 1-7, Jan. 2001.

[10] Breno A. de Melo Menezes, Nina Herrmann, Herbert Kuchen, and Fernando Buarque de Lima Neto, "High-Level Parallel Ant Colony Optimization with Algorithmic Skeletons", International Journal of Parallel Programming, Vol. 49, pp. 776-801, Dec. 2021. https://doi.org/10.1007/s10766-021-00714-1.

[11] A. Del´evacq, P. Delisle, M. Gravel, and M. Krahecki, "Parallel Ant Colony Optimization on Graphics Processing Units", Journal of Parallel and Distributed Computing, Vol. 73, No. 1, pp. 196-202, Jan. 2013. https://doi.org/10.1016/j.jpdc.2012.01.003.

[12] Menezes, B. A., Kuchen, H., Neto, and de Lima Neto, "Parallelization strategies for GPU-based ant colony optimization solving the traveling salesman problem", In Proc. of IEEE Congress on Evolutionary Computation, Wellington, New Zealand, pp. 3094-3101, Jun. 2019. https://doi.org/10.1109/CEC.2019.8790073.

[13] Junqi Yu, Ruolin Li, Zengxi Feng, Anjun Zhao, Zirui Yu, Ziyan Ye, and Junfeng Wang, "A Novel Parallel Ant Colony Optimization Algorithm for Warehouse Path Planning", Journal of Control Science and Engineering, Hindawi, pp. 1687-5249, Vol. 2020, Aug. 2020. https://doi.org/10.1155/2020/5287189.

[14] Stützle Thomas. Ant colony optimization - http://iridia.ulb.ac.be/~mdorigo/ACO/aco-code/public-software.html. [accessed: Dec. 05, 2021]

[15] TSPLIB, "Symmetric traveling salesman problem", http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsplib.html. [accessed: Nov. 22, 2021]

## 저자소개

이 윤 석 (Yunseok Rhee)

1988년 : 서울대학교 계산통계학(학사)
1995년 : KAIST 정보통신공학 (석사)
1999년: KAIST 전산학 (박사)
1988년 ~ 1993년 : 시스템공학 연구소 연구원
1999년 : IBM Watson 연구소 방문연구원
1999년 ~ 현재 : 한국외국어대학교 컴퓨터공학부 교수
관심분야 : 분산병렬시스템, 운영체제, 인터넷 서비스