비용 제약조건을 이용한 병렬 O(n!) 서치 스페이스 탐색 기법의 구현[1)]

이 정 훈
제주대학교 전산통계학과
e-mail: jhlee@jejunu.ac.kr

Implementation of a parallel traversal scheme for O(n!) search space exploiting cost constraint

Junghoon Lee
Dept of Computer Science and Statistics, Jeju National University

요 약

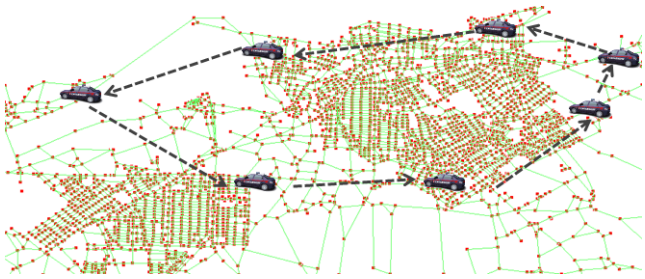
DualCore 혹은 MultiCore 플랫폼의 보급에 따라 높은 시간복잡도를 갖는 응용들도 사용자의 컴퓨터나 단말에서 수행되어 다양한 서비스를 제공할 수 있게 되었다. 본 논문에서는 관광 스케줄을 효율적으로 결정하기 위한 다중목적지 방문 문제에 대해 이중 쓰레드에 기반한 서치 스페이스 탐색 알고리즘을 구현한다. 이는 Traveling Salesman Problem의 한 종류로서 O(n!) 시간 복잡도를 갖고 있으며 검색시의 독립성때문에 각 쓰레드는 병렬적으로 최적의 스케줄을 탐색할 수 있다. 또 현재까지 발견된 최적값을 기반으로 부분 경로의 비용이 이미 최적값을 넘는 경우는 하위 탐색을 제거하여 상당한 성능의 향상을 가져온다. 2.4 GHz Intel(R) Core DuoCPU와 3 GB 메모리로 구성된 플랫폼 상에서 구현된 서비스는 11개의 목적지에 대한 방문 스케줄을 생성함에 있어서 단일 쓰레드 버전은 14.196초, 이중 쓰레드 버전은 6.411초, 제약조건을 포함한 이중 쓰레드 버전은 0.14초에 최적의 스케줄을 찾아낼 수 있다.

1. 서론

최근 Dual-Core나 Multi-Core 개인용 컴퓨터 혹은 이동 단말기들이 보급됨에 따라 하나 이상의 CPU를 탑재하는 멀티프로세서 시스템이 일반적인 작업 혹은 프로그램 환경이 되어가고 있다. 이는 단순히 여러 개의 응용을 병렬적으로 수행시켜 전반적인 수행시간을 단축하는데 그치지 않고 NP 문제들도 병렬 프로그램에 의해 사용자들이 충분히 기다릴 수 있는 시간 내에 수행될 수 있어서 보다 다양한 서비스를 제공할 수 있음을 의미한다. 최근 이동 단말기의 보급은 경로 탐색, 방문 스케줄 결정 등 다수의 지리기반 응용들이 많이 사용되도록 하고 있다 [1].

방문 순서 혹은 일정을 결정하는 스케줄링 문제는 O(n!) 복잡도를 가지며 일반적인 사양의 개인용 컴퓨터에서 n이 11을 넘어가는 경우 이미 10 여초의 수행시간을 필요로 하기 때문에 사용자가 기다릴 수 있는 범위를 넘고 있다 [2]. (그림 1)은 여러 목적지가 주어져 있을 때 이를 순서대로 방문하는 TSP (Traveling Salesman Problem)를 보이고 있으며 택배 기사의 배달지 방문 순서의 결정, 관광객의 관광지 방문 순서의 결정 등 다양한 분야에 적용될 수 있다 [3].

1) 본 연구는 지식경제부 및 정보통신산업진흥원의 대학 IT연구센터 지원사업의 연구결과로 수행되었음 (NIPA-2010-(C1090-1011-0009))



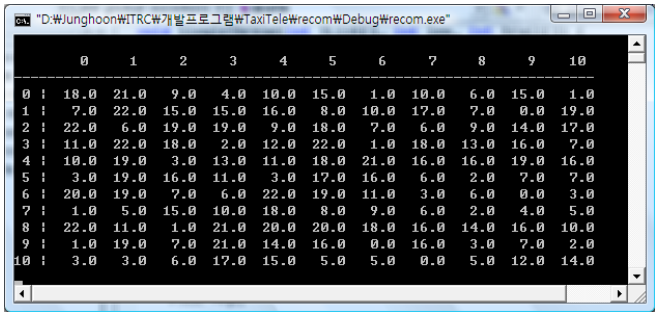
(그림 1) TSP 기반 O(n!) 문제의 정의

2. 분할 검색 방법

먼저 각 목적지들간의 이동 비용이 (그림 2)와 같이 행렬형태로 주어져 있다고 가정한다. 실용적으로 현재의 교통정보 시스템이나 다양한 통행속도 산출 기법에 의해 두 지점간 예상 소요 시간을 다익스트라의 최소경로 기법이나 A* 알고리즘에 의해 계산할 수 있으며 이 소요 시간을 비용으로 설정할 수 있다. (그림 2)는 임의로 비용이 생성된 예로서 동일 노드간 비용은 의미가 없으며 두 지점간 각 방향의 비용은 독립적이라고 가정한다.

Dual-Core 플랫폼에서는 각 CPU에 프로세스 혹은 쓰레드 단위로 작업을 할당하여 병렬적으로 수행이 되도록 한다. 쓰레드 작업시 주의할 사항은 공유 변수가 있는 경우 이에 대한 접근을 중재하여야 하며 이를 위하여 현대의 운영체제들은 다양한 동기화 객체를 제공한다. 분할 검색

방식은 우선적으로 서치 스페이스를 프로세서의 수로 나누어 각 서브트리를 쓰레드가 검색하고 이후 WaitForSingleObject 함수 호출에 의하여 메인 쓰레드가 결과를 수합하도록 한다.



(그림 2) 비용 행렬의 예

Dual-Core 플랫폼에서 메인 쓰레드는 프로그램이 시작되면 2 개의 쓰레드를 생성하고 각 쓰레드의 수행을 위한 자료구조를 초기화한다. 각 쓰레드가 독립적으로 사용하는 자료구조는 Permuation을 구할 때 변경되는 각 원소들의 위치를 저장하기 위한 ThList1, ThList2와 각 쓰레드가 구한 최소 비용의 방문 스케줄을 저장하기 위한 ThTrail1, ThTrail2가 포함된다. 각 쓰레드는 (그림 3)에서 보는 바와 같이 최상위 레벨의 n 개 서브트리에 대해 앞부분과 뒷부분을 병렬적으로 검색하며 이 과정에서 perm 함수를 구동시킨다.

```
// Thread for concurrent operation
int ThList1[Max], ThList2[Max];
int ThTrail1[Max], ThTrail2[Max];
void part_mul1(DWORD *ThreadArg) {
        int i, len = (int) ThreadArg;
        for (i=0; i<len/2; i++) {
                ChangePos(&(ThList1[0]), &(ThList1[i]));
                perm(ThList1, 1, len, ThTrail1, &currun1);
                ChangePos(&(ThList1[0]), &(ThList1[i]));
        }
}
void part_mul2(DWORD *ThreadArg) {
        int i, len= (int) ThreadArg;
        for (i=len/2; i<len; i++) {
                ChangePos(&(ThList2[0]), &(ThList2[i]));
                perm(ThList2, 1, len, ThTrail2, &currun2);
                ChangePos(&(ThList2[0]), &(ThList2[i]));
        }
}
```

(그림 3) 쓰레드 루틴

각 쓰레드가 호출하는 perm 루틴은 (그림 4)에서 보는 바와 같이 일차적으로 가능한 모든 순서, 즉 스케줄을 나열해보는 permutation 기능을 갖고 있으며 각 순서에 대해 비용을 계산하고 최소의 비용을 갖는 스케줄을 찾는다. 이 과정에서 d 레벨에서 호출된 perm 함수는 d 위치에 각 원소를 위치시키고 나머지 len-d 부분에 대해 다시 재귀적으로 perm 함수를 호출한다. d가 len이 되면 스케줄이 완성되어 비용을 계산한다. 이 과정에서 하위 서브트리 검색을 시작하기에 앞서 제약조건을 검사한다. 본 논문에서는 현재까지의 지역 최저값을 currun1, currun2에 저장한 후 스페이스 서치 과정에서 부분합이 이미 지역 최저값보다 커지는 경우 하위 서브트리 검색을 취소한다. 이후 새로운 제약조건이 CheckConstraint 함수에 추가될 수 있다.

```
CheckConstraint(int a[], int d, double *currun, int len) {
        int i;
        double sum = 0;
        if (d < 1) return (0);
        for (i=0; i<d; i++)
                sum += cost[a[i]][a[(i+1) % len]];
        if (sum > *currun) return (1);
        return (0);
}

void perm (int aa[], int d, int len, int trail[], double *currun) {
        int i;
        if (d == len) {
            Dump(aa, len, trail, currun);
            return;
        }
        for (i=d; i <len; i++) {
            ChangePos(&(aa[d]), &(aa[i]));
            if (! CheckConstraint(aa, d, currun, len))
                perm(aa, d+1, len, trail, currun);
            ChangePos(&(aa[d]), &(aa[i]));
        }
}
```

(그림 4) Permutation과 제약조건의 처리

3. 수행 결과 및 결론

본 논문에서는 2.4 GHz Intel(R) Core DuoCPU와 3 GB 메모리로 구성된 플랫폼 상에서 이상의 다중목적지 방문 프로그램을 작성하였다. 11개의 목적지에 대한 방문 스케줄을 생성함에 있어서 단일 쓰레드 버전은 14.196초, 이중 쓰레드 버전은 6.411초, 제약조건을 포함한 이중 쓰레드 버전은 0.14초에 최적의 스케줄을 찾아낼 수 있다.
본 논문에서 개발된 응용은 관광 스케줄 결정에 있어서 다양한 제약조건, 예를 들면, 점심 시간에 식당 근처를 방문, 숙박지에서 가까운 목적지부터 방문시작 등을 추가할 예정이며 불필요한 하위트리 방문을 사전에 제거하기 위한 쓰레드간 동기화 기업을 추가할 예정이다.

참고문헌

[1] R. Healey, S. Dowers, B. Gitting, M. Mineter, Parallel Processing Algorithms for GIS, Taylor&Francis, 1998.
[2] J. Lee, E. Kang, G. Park, "Design and implementation of a tour planning system for telematics users," LNCS, Vol. 4707, 2007, pp.179-189.
[3] http://www.tsp.gatech.edu/concorde/downloads/codes/src/co031219.tgz