多處理機平行程式設計 作業六 report

F74109032 陳均哲

Problem 1

1. 實作說明

讀檔

```
int a, cnt=0;
    queue<int> input;
fp = freopen(filename, "r", stdin);
while (cin >> a) { //先讀入所有數字
    cnt++;
    input.push(a);
}
fclose(fp);
size = sqrt(cnt); //計算城市數量
vector<vector<int>> dis(size, vector<int>(size));
vector<vector<float>> P(size, vector<float>(size));
for (int i = 0; i < size; i++) {
    for (int j = 0; j < size; j++) { //城市間距離與機率初始化
        dis[i][j] = input.front();
        input.pop();
        if (i != j) P[i][j] = 10000.0 / dis[i][j];
    }
}
```

我這邊會讓每個電腦都讀入距離資訊，讀檔的方式是利用freopen再cin的方式，先把所有數字都讀進去，再計算城市數量是多少。然後前面我先用一個queue紀錄所有數字，再放進去dis陣列，同時也計算出史機率陣列，我設定是10000.0 / edge，讓距離跟機率有成反比的關係。

選擇隨機城市

```
float total = 0;
int unvisited = 0;
for (int j = 0; j < size; j++){
    if (Sp[j]) { //把沒走過城市都塞到輪盤
        unvisited++;
        total += P[city][j];
    }
}

if ((int)total == 0) { //如果機率總和太小 就隨機取一個
    int num = rand() % unvisited;
    for (int j = 0; j < size; j++) {
        if (Sp[j]) {
            num--;
            if (num <= 0) {
                city = j;
```

```
                break;
            }
        }
    }
}
else {
    int num = rand() % (int)total;
    for (int j = 0; j < size; j++) {
        if (Sp[j]) {
            num -= P[city][j];
            if (num <= 0) {
                city = j;
                break;
            }
        }
    }
}
```

這邊每隻螞蟻選擇下一個城市的方法是把沒走過的塞進輪盤裡面，然後把所有機率相加，再隨機出一個數字出來後對機率總和取餘數，然後對每個塞進來的城市機率相減，撿到數字<=0為止，就可以選擇出隨機的城市。而為了避免機率加起來太小，如果加起來total取整數還是0的話，就以相同機率隨機選擇下一個城市。但是以我寫的程式碼應該是不太可能發生這樣的情況，因為我把算出來的機率都乘上10000讓他比較好看，但還是以防萬一加上去。

每隻螞蟻的路徑紀錄

```
vector<bool> Sp(size, true); //代表某隻螞蟻有沒有造訪過某個城市
int city = rand() % size; //隨機初始城市
T[k][0] = city; //紀錄路徑
L[k] = 0; //紀錄走過距離
Sp[city] = false;
for (int i = 1; i < size; i++) {
    city = random_city(); //這邊的程式碼如同上一步
    Sp[city] = false;
    L[k] += dis[T[k][i-1]][city];
    T[k][i] = city;
}
//回到最一開始的城市
T[k][size] = T[k][0];
L[k] += dis[T[k][size-1]][T[k][size]];

//從這次計算選出每隻螞蟻中最短路徑
if (L[k] < Lbest) {
    Lbest = L[k];
    memcpy(Tbest, T[k], sizeof(int) * (size+1));
}
```

這邊的程式碼是每隻螞蟻會走的迴圈程式碼，初始會完全隨機產生一個初始城市，然後用T跟L記錄路徑跟距離，這樣從頭到尾走完一輪後跟Lbest比較，LBest是每個thread private的紀錄變數，跳出每隻螞蟻的路徑迴圈後，每個thread都會有一個LBest。

更新最短距離(Bonus)

```
if (Lbest < Lglobal) {
    #pragma omp critical
    {
        if (Lbest < Lglobal) { //再次檢查 以免race condition
            Lglobal = Lbest;
            memcpy(Tglobal, Tbest, sizeof(int) * (size+1));
        }
    }
}
```

每個thread都會記錄的最短距離，當一整個蟻窩跑完的時候會再進入上面這串程式並利用critical section更新globla的最短距離。

更新費洛蒙與機率

```
//更新費洛蒙濃度
for (int i = 0; i < size; i++) {
    for (int j = 0; j < size; j++) {
        pheromone[i][j] *= (1 - evap);
    }
}

for (int k = 0; k < M; k++) {
    for (int i = 0; i < size; i++) {
        pheromone[T[k][i]][T[k][i+1]] += (Q / L[k]);
    }
}

#pragma omp barrier

//更新每個edge選擇的機率
for (int i = 0; i < size; i++) {
    for (int j = 0; j < size; j++) {
        if (i != j) P[i][j] = (pow(pheromone[i][j], Alpha) * pow(1.0f / dis[i]
[j], Beta)) * 10000;
    }
}
```

費洛蒙基本上就是按照hackmd上面的方式更新。
而機率的計算方法，把除了自己跟自己以外的edge都按照說明的方式更新，但就像前面提到的，我把算出來的機率值都乘上10000比較好看也可以避免上面說到總和太小的問題。
而因為機率值會被費洛蒙影響，為了避免在費洛蒙還沒計算完成的情況下就更新機率，因此我在中間放一個barrier。

2. 執行結果

以下最短距離結果皆為np=4的條件測試

gr17_d.txt	fri26_d.txt	dntzig42_d.txt	att48_d.txt
2085	937	700	34482

3. 結果分析

這算是我稍微微調一點參數能得到稍微好一點的結果了，前兩個比較小的測資都能得到正確的解答，而後面兩個測資都比較大，沒辦法得到最佳解，但我認為我算出的也還算接近了。我前面修改參數的方向大概是，增加費洛蒙濃度的影響(alpha)，減少edge距離對費洛蒙的影響(beta)。另外則是稍微增加evaporation rate，讓新的路徑影響費洛蒙程度高一點，還有更改Q稍微高一點
，讓新的距離對費洛蒙的影響高一點。最後再適量的增加計算數量以及螞蟻數量，但畢竟增加太多會讓執行時間太久，因為我就放在一個我覺得還算適合的數值。但因為後面兩個測資都太多edge的關係要得到最短路徑的機率比較難，我目前的程式還收斂不夠快。因此有可能是要再調整參數或是乾脆直接增加蟻群或計算數量才有更高機率得到最佳解。至於我不是統計學的專家，我覺得我已經得到夠接近的答案了，開心就好。