Optimization of Travelling Cost through a Vertex-Weighted Undirected Graph

Jade Cheng

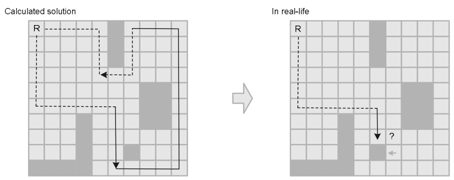
December 2008

Introduction

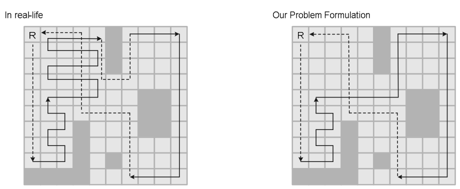
The paper is concerted with Roomba's novel feature, which optimizes the summation of priorities through a vertex-weighted, undirected mesh graph. The paper presents a brut-force algorithm to solve this problem. The paper provides a comprehensive problem formulation, algorithm design, and the complexity analysis for the algorithm. We further prove the correctness of the algorithm and the fact that it is unlikely for us to find a polynomial algorithm that solves this problem because the worst exists can be converted into to a well-known NP-complete—the longest path problem. The algorithm proposed in this paper runs at an exponential time.

Preliminaries and Problem Formulation

Simplification 1: Roomba stores the complete map; Computation is done off-line



Simplification 2: Roomba does not switch between cleaning and traveling modes within one trip



Given: A function $f(\sigma) = \sum_{k=1}^{r} p_{i_k j_k}$: $V = \{v_{i,j} \mid 1 \le i \le m \wedge 1 \le j \le n\}$ denotes the vertices of a $m \times n$ mesh graph, $\mathcal{U}_{m,n}$; $E = \{(v_{i,j}, v_{k,l}) \mid 1 \le i \le m \wedge i \le j \le n \wedge 1 \le k \le m \wedge 1 \le l \le n \wedge [(i = k \wedge |j - l| = 1) \vee (|i - k| = 1 \wedge j = l)]\}$ denotes the edges of $\mathcal{U}_{m,n}$; $p_{i,j} \in \mathbb{N}$ for each $v_{i,j} = V$ denotes the priorities of the vertices, where $\mathbb{N}$ denotes the set of natural numbers.

Constraint: One inequality $g = \sum_{k=1}^{r} c_{i_k,j_k} + d(v_{a,b}, v_{i_1,j_1}) + \sum_{k=1}^{r-1} d\left(v_{i_k,j_k}, v_{i_{(k+1)},j_{(k+1)}}\right) + d(v_{i_r,j_r}, v_{a,b})$: $g \le C$, where $1 \le i_1 \le m \wedge 1 \le j_1 \le n$; C as a constant denotes to battery capacity.

Sought: A sequence σ of r *distinct* vertices $v_{i_1,j_1}, v_{i_2,j_2}, \cdots, v_{i_r,j_r}$ which maximizes the objective function f(σ), subject to the following constraint g, where $1 \le i_1 \le m \wedge 1 \le j_1 \le n$.

Algorithm Design and Optimization

Key idea: Design a brute-force algorithm that checks all feasible paths when battery constraint allows.

Loop through all possible starting points. For each, try moving all possible directions.

Base Case 1: The front most vertex of the path is surrounded by vertices that are already covered on the same path or the boundary of the map.

Base Case 2: Current battery life cannot sustain the cleaning of the vertex examining.

```
Graph input;
BooleanMatrix matrix;
int maxGoodness;
GraphSolution output;
Stack<Vertex> sequence;

JADE-MESH-OUTERLOOP(Graph graphInput)
    input ← graphInput;
    matrix ← initialize as the size of graphInput and populate with false;
    maxGoodness ← the minimum integer;
    output ← null;
    sequence ← initialize as a new object;
    for int i ← 0 to i ← graphInput.width; i++ {
        for int j ← 0 to j ← graphInput.height; j++ {
            JADE-MESH-RECURSION (i,j, 0, graphInput.capacity
            – capacityToBase(i,j));
        }
    }
    return output;

JADE-MESH-RECURSION(int x, int y, int goodness, int capacity)
    if isBlocked(x, y) = true or isBatteryExhausted(x, y, capacity) = true {
        If output = null or goodness > maxGoodness {
            mxGoodness ← goodness;
            Output ← new GraphSolution(sequence, goodness);
        }
        return;
    }
    sequence.push(new Vertex(x, y));
    matrix.mark(x, y);
    int newGoodness ← goodness + input.priority(x, y);
    int newCapacity ← capacity – input.comsumption(x, y) – 1;
    JADE-MESH-RECURSION(x – 1, y, newGoodness, newCapacity);
    JADE-MESH-RECURSION(x + 1, y, newGoodness, newCapacity);
    JADE-MESH-RECURSION(x, y – 1, newGoodness, newCapacity);
    JADE-MESH-RECURSION(x, y + 1, newGoodness, newCapacity);
    sequence.pop();
    matrix.unmark(x, y);
    }
```

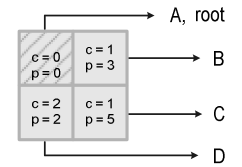
The helper methods used above:

```
CAPACITYTOBASE(int x', int y')
    return distance(input.base.x , input.base.y, x', y');

ISBATTERYEXHAUSTED(int x, int y, int capacity)
    if capacityToBase(x, y) + input.consumption(x, y) + 1> capacity {
        return true;
    }
    return false;

ISBLOCKED(int x, int y)
    if x < 0 or x <= matrix.width or y < 0 or y >= matrix.height {
        return true;
    }
    if x = input.base.x and y = input.base.y {
        return true;
    }
    return matrix.isMarked(x, y);
```

Improvement Attempts

The worst case of this problem can be addressed as the Longest Path Problem, which is a known NP-complete problem. The worst case happens when the input battery capacity is sufficient of traveling all over the map and clean as much as the map allows. We cannot prevent this worst case from happening. Therefore, it is unlikely for us to find an algorithm that runs significantly faster—

Simplified Example



Input Graph

All candidate solutions if battery capacity is enough

Path	c Consumption	p Summation
D, C, B	9	10
B, C, D	9	10
C, B	7	8
B, C	7	8
D, C	8	7
C, D	8	7
C	6	5
B	4	3
D	5	2

Solutions for all possible battery capacity ranges

Battery Capacity	Final Solution
$C \ge 9$	D, C, B or B, C, D
$7 \le C < 9$	B, C or C, B
$C = 6$	C
$4 \le C < 6$	B
$C < 4$	null

Complexity Analysis

Time complexity: Jade-Mesh-OuterLoop runs at $O(V)$. Jade-Mesh-Recursion runs at $O(4^v)$. Therefore, the designed brute-force algorithm made up of these two methods runs at exponential time.

$$O(V) \times O(4^v) = O(V \times 4^v)$$

Space complexity: The space usages of this algorithm are mainly associated with the several data collector ADTs. They all take spaces linearly proportional to the size of V from the input mesh graph. The over all space complexity is therefore $O(V)$.