Wavefront Propagation

Luis Alberto Ballado Aradias
Cinvestav Unidad Tamaulipas
luis.ballado@cinvestav.mx

April 20, 2023

Abstract

The task of building a map of an unknown environment and concurrently using that map to navigate is a central problem in mobile robotics research. This document trys to replicate one of the knows algorithm (wavefront expansion) to explore a map using plan C code in the first phase and parallelizate it to perform its calculations trying to reduce the total computation time. Parallel wavefront expansion can significantly improve the computational performance and speed of the algorithm, particularly when processing large environments or complex obstacles. By utilizing multiple processing units, the algorithm should process the environment, reducing the total computation time. However, parallel software also has some limitations, such as increased complexity in implementation and increased communication overhead between processing units. It is important to consider these limitations.

I. Introduction

Wavefront expansion is a popular path planning algorithm used in mobile robotics for determining the optimal path for a robot to move from its starting position to its goal while avoiding obstacles. The algorithm starts with the goal position and creates a wavefront that expands outward, considering the surrounding environment and updating the cost of reaching each cell. The final path is determined by selecting the lowest cost path from the starting position to the goal.

Beginning with mid-1960s, the path planning has attracted interests from a lot of scholars. The path planning problem can be described in the following: given a robot and its working environment, the mobile robots searches for an optimal or suboptimal path from the initial state to the target state according to a certain performance criteria. Good path planning technology of mobile robot can not only save a lot of time, but also reduce the wear and capital investment of mobile robot.

Wavefront expansion is well-suited for real-time applications, as it can be executed efficiently in terms of computation time. The algorithm can handle complex environments with many obstacles and can produce an optimal path in a relatively short amount of time. Additionally, the algorithm can be modified to incorporate additional objectives, such as time constraints or energy consumption, by incorporating different cost functions.

One of the key benefits of wavefront expansion is its simplicity, which makes it easy to understand and implement. The algorithm is also highly flexible, as it can be adapted to various types of robots and environments, from ground robots to aerial robots, and from indoor to outdoor environments.

However, wavefront expansion has some limitations that need to be considered. For example, the algorithm does not consider the dynamics of the robot or the motion constraints of the environment. As a result, the final path produced by the algorithm may not be feasible for the robot to follow.

II. Serial Algorithm

The typical wavefront expansion algorithm uses the implementation of an adjacency list representation of graphs, storing the lists of adjacent nodes and the queue of nodes needed for BFS traversal.

At this first phase I am not using the BFS algorithm (because of my data-structure issues), not discarding also develop the bfs version of the algorithm and compare the results applying the parallelism techniques.

The serial main program consists of the invocation of five functions:

- **map_init:** Initialize the map
- **propagate_wavefront:** main function of the algorithm in charge of propagating and coordinating the exploration of new nodes to visit
- **clear_map:** just crear the map for another experiments
- **explore_neighbors:** explore and returns the min node cost
- **show_map:** prints out the actual map to see whats going on

i. map_init

This part just initialize the map putting the elements at it.

```
system("clear");
showmap();
print("Se comienza wavefront");
sleep(1);
put robot at the map;
put goal at the map;
show map();
print("Se incluyo el objetivo y el robot")
sleep(1);
```

Algorithm 1: Map initialize

ii. propagate_wavefront

This functions controls the propagations between the differents nodes at the matrix, every time a min neighbor is found and the value is not considered SPECIAL ITEM such as WALL or ROBOT is the min neighbor

```
$MAX\_EXP \leftarrow 20$ ;                    /* MAX iter for exploration, if no solution */
$min\_neigbor \leftarrow 0$;
$counter \leftarrow 0$;
while $counter \leq MAX\_EXP$ do
    $x \leftarrow 0$;
    $y \leftarrow 0$;
    while $((x < SIZE\_MATRIX\_X)\&\&(y < SIZE\_MATRIX\_Y))$ do
        if $((map[x][y]! = WALL)\&\&(map[x][y]! = ROBOT))$ then
            $show\_map()$;
            $min\_neighbor \leftarrow explore\_neighbors(x, y)$;
            if $((min\_neighbor < SPECIAL\_ITEM)\&\&(map[x][y] == GOAL))$ then
                $show\_map()$;
                $print(''termine'')$
            else if $min\_node! = SPECIAL\_ITEM$ then
                $map[x][y] = min\_node + 1$;
        end
        MAKE ROW EXPLORATION (y++);
        if $((y == SIZE\_MATRIX\_Y)\&\&(x! = SIZE\_MATRIX\_X))$ then
            $x \leftarrow x + 1$
            $y \leftarrow 0$
        end
    end
    $show\_map()$;
    $counter \leftarrow counter + 1$
end
```

Algorithm 2: Propagate wavefront

iii. clear_map

Function used to clean the map for the differents experiments

```
sleep();
for ( $x = 0$; $x < SIZE\_MATRIX_X$; $x = x + 1$ ) {
    for ( $y = 0$; $y < SIZE\_MATRIX\_Y$; $y = y + 1$ ) {
        if map[x][y]!=WALL then
            map[x][y] = EMPTY;
        end
    }
}
show_map();
```

iv. explore_neighbors

Explore the neighbors, it starts once the robots is found. Consist to visit diferents nodes (UP,DOWN,LEFT,RIGHT)

```
Data: $x, y$
Result: min_node
min_node ← SPECIAL_ITEM;
if (x<SIZE_MATRIX_X-1) then
    if ((map[x+1][y]<min_node)&&(map[x+1][y]!=EMPTY)) then
        min_node = map[x+1][y];
    end
end
if x>0 then
    if ((map[x-1][y]<min_node)&&(map[x-1][y]!=EMPTY)) then
        min_node = map[x-1][y] ;
    end
end
if (y>0) then
    if ((map[x][y-1]<min_node)&&(map[x][y-1]!=EMPTY)) then
        min_node = map[x][y-1];
    end
end
if y<SIZE_MATRIX_Y-1 then
    if ((map[x][y+1]<min_node)&&(map[x][y+1]!=EMPTY)) then
        min_node = map[x][y+1] ;
    end
end
```

Algorithm 3: Explore neighbors

v. show_map

```
for ( $i = 0$; $i < SIZE\_MATRIX_X$; $i = i + 1$ ) {
    for ( $j = 0$; $j < SIZE\_MATRIX\_Y$; $j = j + 1$ ) {
        if map[i][j]==WALL then
            print(###);
        else if map[i][j]==ROBOT then
            print(−R−);
        else if map[i][j]==GOAL then
            print(−X−);
        else
            print(map[i][j])
    }
}
show_map();
```

The time complexity of the wavefront expansion algorithm without a queue is $O(n^3)$, where N is the size of the environment. This is because the algorithm iterates over all cells in the environment for each iteration of the algorithm, and each cell is processed once per iteration. The calculation of the cost of each cell is typically a constant time operation, so the total time complexity is proportional to the number of cells in the environment multiplied by the number of iterations.

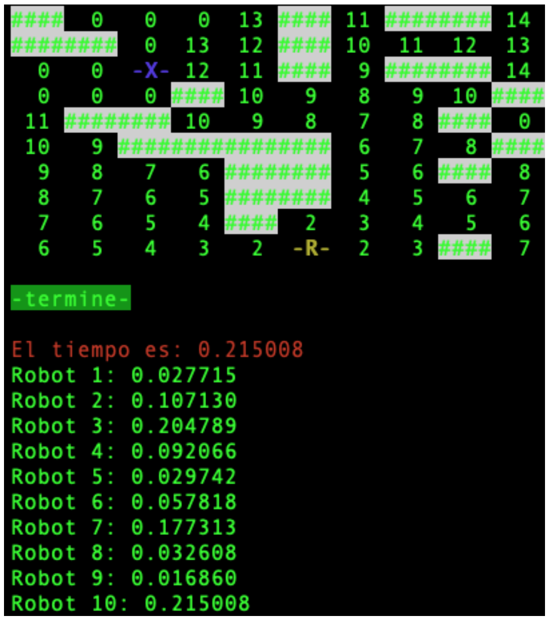
The space complexity of the wavefront expansion algorithm without a queue is $O(n^2)$, as it requires a 2D array to represent the environment and store the costs of each cell. Additionally, a variable may be used to keep track of whether each cell has been processed, which would also contribute to the space complexity.

III. Results

i. Examples

The following image shows a table with different times after an iteration of ten differents robots changing the robot location so as the goal location.

At this time the results are not confident because the locations are computed randomly and there is no equatity at the location, by that I mean maybe one problem could be harder than the other.



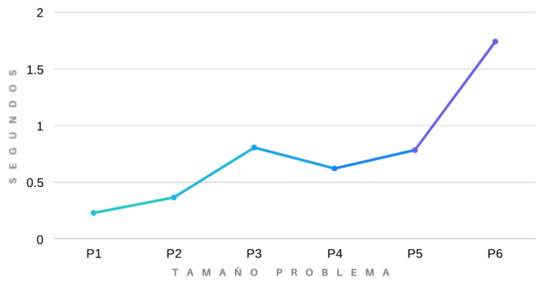
```
####   0   0   0  13####  11########  14
########   0  13  12####  10  11  12  13
   0   0 -X-  12  11####   9########  14
   0   0   0####  10   9   8   9  10####
  11########  10   9   8   7   8####   0
  10   9################   6   7   8####
   9   8   7   6########   5   6####   8
   8   7   6   5########   4   5   6   7
   7   6   5   4####   2   3   4   5   6
   6   5   4   3   2 -R-   2   3####   7

-termine-

El tiempo es: 0.215008
Robot 1: 0.027715
Robot 2: 0.107130
Robot 3: 0.204789
Robot 4: 0.092066
Robot 5: 0.029742
Robot 6: 0.057818
Robot 7: 0.177313
Robot 8: 0.032608
Robot 9: 0.016860
Robot 10: 0.215008
```

ii. Plot & table

The larger the matrix, the longer it will take to solve the problem.

Table 1: *Problem Size vs Time*

P1	0.232747
P2	0.369061
P3	0.809046
P4	0.624046
P5	0.787176
P6	1.743657



IV. Conclusions

The wavefront expansion algorithm is a technique used to parallelize the processing of graph algorithms. This algorithm works by processing vertices in a wave-like manner, where vertices in the same wavefront are processed in parallel. The wavefront expansion algorithm has been shown to be highly efficient for certain types of graphs and is commonly used in parallel programming.

It's worth noting that the actual complexity of the wavefront expansion algorithm without a queue may be influenced by the specifics of the environment and the requirements of the specific use case. For example, if the environment is sparse, with large areas of empty space, the number of iterations may be significantly reduced, which would decrease the time complexity.

In general, the wavefront expansion algorithm without a queue is not as efficient as the version that uses a queue, as the time complexity is higher. However, this alternative version may still be useful in certain situations, such as environments where memory is limited and a queue data structure cannot be used.

Parallel programming refers to the design and development of applications that run simultaneously on multiple processors or cores. It enables faster and more efficient processing by dividing a problem into smaller, parallelizable tasks that can be executed in parallel. The wavefront expansion algorithm is an example of a parallel programming technique that can be used to speed up the processing of graph algorithms.

References

[George A. Bekey, 2005] George A. Bekey - Autonomus Robots From Biological Inspiration to Implementation and Control - MIT Press (2005). *ISBN*, 0-262-02578-7

[Ronald C. Arkin, 1998] Ronald C. Arkin - Behavior Based Robotics - MIT Press (1998). *ISBN*, 978-0-262-01165-5

[Zidane,Issa and Ibrahim,Khalil, 2018] Wavefront and A-Star Algorithms for Mobile Robot Path *ISBN*, 978-3-319-64860-6

[Zhang,Han-ye and Lin,Wei-ming and Chen,Ai-xia, 2018] Path Planning for the Mobile Robot: A Review *ISSN*, 2073-8994

[Wu,Sifan and Du,YU and Younghua Zhang, 2020] Mobile Robot Path Planning Based on a Generalized Wavefront Algorithm. Mathematical Problems in Engineering Volume 2020 *Article ID*, 6798798 Volume 2020

[Bhavya Ghai and Anupam Shukla BV- Indian Institute of Information Technology and Management] Wave front Method Based Path Planning Algorithm for Mobile Robots *ISSN*, 2073-8994

[Ansuategui A, Arruti A, Susperregi L, Yurramendi Y, Jauregi E, Lazkano E, Sierra B., 2014] Robot trajectories comparison: a statistical approach. *ScientificWorldJournal*, 2014;2014:298462

[Michael Soulignac, Patrick Taillibert, Michel Rueher] Adapting the Wavefront Expansion in Presence of Strong Currents. *2008 IEEE*, International Conference on Robotics and Automation