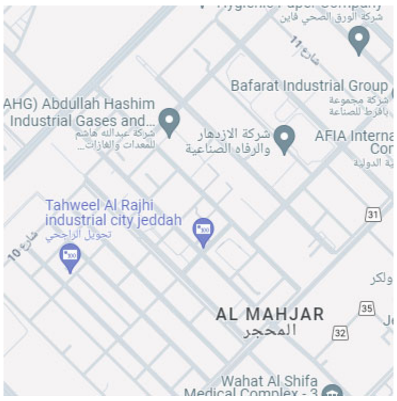
A Parallel Algorithm for Finding Routes in Cities with Diagonal Streets

Hatem M. El-Boghdadi
helboghdadi@iu.edu.sa
Faculty of Computer & Information Systems, Islamic University of Madinah, Saudi Arabia

Summary
The subject of navigation has drawn a large interest in the last few years. The navigation within a city is to find the path between two points, source location and destination location. In many cities, solving the routing problem is very essential as to find the route between different locations (starting location (source) and an ending location (destination)) in a fast and efficient way. This paper considers streets with diagonal streets. Such streets pose a problem in determining the directions of the route to be followed. The paper presents a solution for the path planning using the reconfigurable mesh (R-Mesh). R-Mesh is a parallel platform that has very fast solutions to many problems and can be deployed in moving vehicles and moving robots. This paper presents a solution that is very fast in computing the routes.
Keywords:
Path planning, parallel computation, branching streets.

1. Introduction

The navigation problem in smart cities is an important problem because of its applications. The path planning problem (navigation) is to find the path between two locations, a source and a destination.


Bafarat Industrial Group
AHG) Abdullah Hashim Industrial Gases and...
AFIA Interna Cor
Tahweel Al Rajhi industrial city jeddah
AL MAHJAR
المحجر
31
35
32
Wahat Al Shifa Medical Complex - 3
Figure 1. A map of a certain geographical area containing diagonal streets

We use the Reconfigurable Mesh (R-Mesh), to compute the path from source location to destination

Navigation systems guide people to move easily among different locations. In such case, solutions to the navigation problem are essential. The navigation algorithms are the core of navigation systems. In this paper we consider such algorithms to find a path between two locations within a map in parallel.

The navigation algorithms that we consider in this paper depends on maps as inputs to the algorithms. The idea is to build explicit maps or uses pre-prepared maps and then planning the route through these maps [12]. The map contains all available routes and its directions and could be for the whole city or certain geographical area of the city (see Figure 1 for an example of a map).
In [6], we presented algorithms to deal with perpendicular streets. In this paper we extend this work and consider maps that contains diagonal streets and follow the same lines as in [6].

Location. Platforms such as the R-Mesh [2,3] (shown in Figure 2) have the ability to change the interconnection between processors dynamically at every step of the computation to allow efficient communication as well as to perform computation faster than conventional non-reconfigurable platforms. A 2D R-Mesh is an array of processors that are connected with fixed connections between each two neighboring processors. Also, each processor has internal connections that can be dynamically reconfigured. This allows altering the interconnection among processors very fast, possibly at each step of the computation.

Manuscript received January 5, 2024
Manuscript revised January 20, 2024
https://doi.org/10.22937/IJCSNS.2024.24.1.6

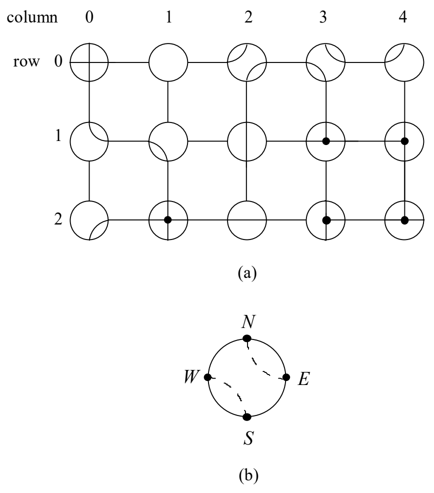


Figure 2. (a) Example of buses in a 3 × 5 R-Mesh (b) Processor ports

Each processor can independently set (partition) its ports to connect certain ports together at each step of the computation. For example, in Figure 2(a) the top left processor connects its N port to its S port, and its E port to its W port. The corresponding partition is denoted by $\{\overline{NS}, \overline{EW}\}$. Figure 2(a) shows the fifteen possible port partitions of the R-Mesh.

A certain setting for a port partition to each R-Mesh processor is called a configuration (see Figure 2(b) for a possible configuration). The port partitions along with the underlying mesh connections between neighboring processors form buses connecting processors (see Figure 2(a) for buses formed in a certain step).

At each step of an R-Mesh algorithm [2,3], a processor could perform the following actions: (1) configure (partition) its ports, (2) read from or write to its ports, and (3) perform a local computation. An R-Mesh could permit concurrent reads and writes.
Researchers have used R-Mesh to design fast and efficient algorithms in image processing, computer vision, arithmetic problems, graph problems, etc. [2,3]. In this paper, we consider the design of parallel algorithms for the navigation problem on R-Mesh. We consider maps with diagonal as well as perpendicular streets. The we analyze the proposed algorithm with respect to running time and the quality of the path.

The next section presents the related work. In section 3, we show the overall system architecture assumed in this paper. Section 4 presents the path planning algorithm for maps with diagonal streets. Finally, in section 5, we make some concluding remarks.

2. Related Work

The path planning problem has a wide range of applications. This operation is an essential operation in many navigation systems [21,22] needed to navigate through cities such as indoor navigation and maze solving [17,18].
The navigation problem finds a path from a source location to a destination location within an environment. In this paper we solve the navigation problem using maps for the environment.

The maps are constructed in the exploration stage [11,12] and then used in the path planning phase to navigate. The map contains all available routes and its directions. This proposal follows this approach and assumes that map is already built in the exploration stage. Maps contain the available paths to follow as well as direction of each path.

Reconfigurable platforms such as R-Mesh [2,3] were used in literature to propose robot path planning algorithms that aim to plan a path for a moving robot from a source location to a target location with the existence of static or dynamic obstacles. The main input to the algorithm is an image (map) with obstacles. In this paper we consider maps with available paths and direction of each path to accommodate the navigation problem within cities. Adding these restrictions to the map would add more complexity to the algorithm. The R-Mesh was also to handle restrictions on robot movement [2,3].
To simplify the design and analysis of the problem, the map is used first to generate the configuration space [8].

Planning for collision-free path was presented by Tzionas et al. [9] for diamond-shaped robot. The configuration space was computed optimally on hypercube in [10]. The algorithm was shown to be optimal where it requires $O(\log n)$ time for an n x n image by using n x n Mesh of processors. The reachability problem in one dimensional space was solved efficiently D. Wang [7].
The maze-routing problem was considered by H.C. Lee [5]. D. The authors in [4] used disjoint convex or concave polygons as obstacles. The algorithm uses

$O(k)$ time to compute a path from a source to a destination while avoiding all obstacles in the environment, where N is the total number of processors (pixels) and k is the number of obstacles. However, the algorithm requires $O(\log 2N)$ preprocessing time for the given obstacle image.
The authors in [5] proposed $O(1)$ time algorithm to find a collision free path in the existence of obstacles. Also, the authors proposed a measure for the quality of the path that depends on the number of bends in the path.

All the above algorithms assume an image with obstacles. Thus, it is available to navigate and avoid obstacles.

In [6], the authors presented two algorithms to find the route between a source location and a destination location using the R-Mesh. They considered streets with linear streets. In this work we follow the same lines as in [6] (see Figure 1).

3. System Architecture

We use the same system architecture as in [6]. The proposed system is shown Figure 4. The main platform for computation is the R-Mesh that takes map areas as input and generates in parallel the navigation path.

In this work, we used the R-Mesh platform to solve the navigation problem. As mentioned before, one very important advantage is the speed by which we achieve the path from source location to destination location. The input to the algorithm is a map with all routs and its directions. We plan to look at the speed of the solution as well as the quality of the generated solution. We consider two types of maps; maps with linear streets and maps with branching streets.

4. Algorithm for Diagonal streets

In this section, we present the main algorithm proposed in this paper to find the route between two locations using maps with diagonal streets with the existence of perpendicular streets. Diagonal streets pose a problem in the sense that they may cause circular routes. This raises the question of the quality of the found path. The found path between two locations might be longer than it is required. We first start by presenting some known operations on R-Mesh that will be used later.

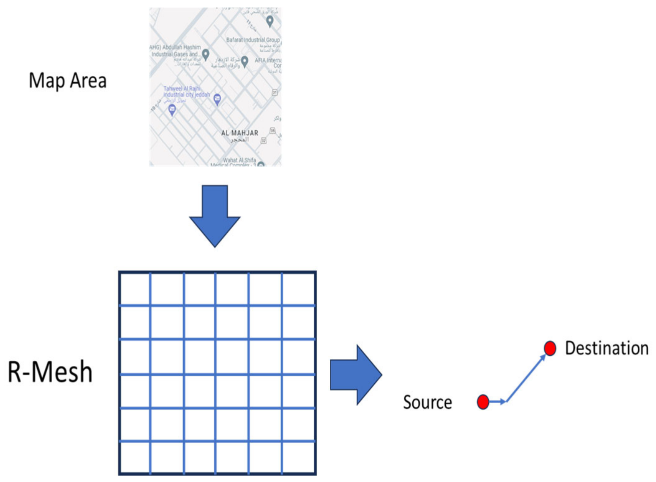


Figure 4. Overall system Architecture [6]

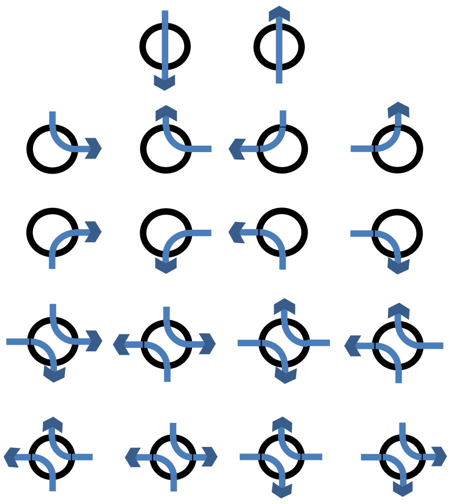


Figure 5. Directions of linear streets through a node

4.1 Basic Operations

In this section we show the two main operations that are performed on the RMesh that will be used in the proposed algorithms. Namely, Broadcasting Data and Counting N bits.

The first operation, Broadcasting data, is well known to be performed in $O(1)$ time [3] on R-Mesh. This can be done by partitioning all ports as NSEW. If any processor writes data to one of its ports, data will reach all other processors. The second basic operation is Counting N bits, where the operation can be performed on an N x N R-Mesh in $O(1)$ [3].

Now we describe the proposed algorithm to solve the navigation problem using maps with diagonal streets. As in [6], the image of map and streets are digitized and stored in the R-Mesh, with one processor holding one pixel of the image. Also, the direction of each street is stored in each processor. Then each processor has all the information about all streets passing through them. Since we consider linear streets, any processor could have only at most one street with directions. These directions are shown in Figure 5. As shown, there are 18 different directions that could pass through a processor.

4.2 Path Planning Algorithm

Assume S (resp. D) be the source node resp. destination node.
Input. S, D, and all map information
Output. Route from S to D

Step 1 Nodes S and D broadcast their IDs to all processors.

Step 2 Nodes S and D exchange their IDs. If the IDs received by the two nodes, then there is a route between the two nodes.

Step 3 Let all processors read their own four ports in sequence.

Step 4 Node D writes its ID to all of its ports and Node S reads its ports. We have the following cases:

Case 1 If S reads the ID of node D from one of its ports, p, then there is a route from the S node to D node starting from port p in S node to port y in D node.

Case 2 If S reads ID of node D from more than one of its ports, then S concludes that there is more than one route from S to D.

Case 3 If S does not read ID of node D from any of its ports, then S concludes that there is no route from S to D.

Step 5 If a processor reads ID of S and ID of D in Step 2, then this processor is part of this route from S to D. Let all such processor record the route number, R, it belongs to.

Step 6 For each found route, let each processor belong to the route cut the bus and exchange ID with neighboring nodes along the route.

Now we describe the Algorithm in details. Step 1 broadcasts the IDs for S node and D nodes to all nodes in RMesh. Thus, all processors have information about the S and D nodes. In Step 2, S and D nodes exchange their IDs using the configuration of processors according to streets passing through it. This allows to send information along the streets according to the actual map. In Step 3, processors read their ports in sequence and each processor decide whether they belong to the route from S to D or not.

Since all streets are linear, then at most there could be four different routes from S node to D node. Step 4 checks how many routes are there from S to D (if any). We have three cases as shown in the algorithm.

All processors read their ports in sequence in Step 5.

Finally, in Step 6, to reach from *S* node to *D* node, for each found route, let each processor belong to the route cut the bus and exchange ID with neighboring nodes along the route. Now each processor knows its successor and predecessor.

Time Analysis

Here, we show the time analysis of the algorithm.

Step 1 uses one of the basic operations shown in section 4.1 which runs in $O(1)$ time.

Step 2 performs local port partitioning which requires constant time.

In **Step 3**, processors read their ports in sequence. Since we have four ports, then this step requires at most $O(1)$.

In **Step 4**, various processors either write to a port or read from a port. Both operations runs in $O(1)$ time.

In **Steps 5** and **6**, various processors either write to a port or read from a port. Both operations run in $O(1)$ time.

Thus, the whole algorithm runs in $O(1)$ time.

Theorem 1 For a map with **diagonal** streets, an NxN R-Mesh can compute the route from a source node to a destination node in $O(1)$ time. ■

As in reference [6], since all the steps of the algorithm requires linear partitions, then the algorithm can run on the linear RMesh (LR Mesh). Only Step 1 requires {NSEW} to broadcast data in constant time. This step can be done also on the LR-Mesh in constant time and we have the following result.

Theorem 2 For a map with **diagonal** streets, an NxN LR-Mesh can compute the route from a Source node to a destination node in $O(1)$ time. ■

Length of the route

Here we measure the length of the route from source node to destination node in terms of the number of processors the route snakes. As in [6], we first, form a list of all processors belonging to the route in the same row. Then, count the number of processors in each row in sequence in $O(1)$ time. For N rows, the total time required to count all processors in all rows belonging to the route is $O(N)$ and we have the following result.

Lemma 3 The number of processors belonging to the route from source node to destination node can be computed using N x N R-Mesh in $O(N)$ time. ■

If we use an N x N^2 R-Mesh, then we can count the total number of processors belonging to the same route in $O(1)$ time.

Lemma 4 The number of processors belonging to the route from source node to destination node can be computed using N x N^2 R-Mesh (or LR-Mesh) in $O(1)$ time. ■

Also the length of the route could be measured in the number of turns in the route.

Counting the Number of Turns

We can follow similar procedure as the one followed in measuring the length of the route as follows. First we flag each processor with 1 if it has a turn in the route. Then, in each row we form a list with all flagged processors. Finally, we count all flagged processors in the R-Mesh. Thus, we have the following results:

Lemma 5 The number of turns from S node to D node can be counted using N x N^2 R-Mesh (or LR-Mesh) in $O(1)$ time. ■

5. Conclusion

In this paper, we have presented the navigation problem and proposed a solution for the maps with diagonal streets, and the use of R-Mesh to solve the problem. Our analysis shows that the algorithm runs in $O(1)$ time. We also showed the computation of the length of the route and the number of turns in the path.

Future directions include using other parallel platforms. Also, more restrictions on the maps and streets could be considered.

References

[1] Jaja J, An introduction to parallel algorithms. Addison Wesley, Redwood City, CA, USA, 1992.

[2] Bondalapati K, Prasanna VK. Reconfigurable computing: architectures, models and algorithms. Curr Sci 78:828–837, 2009.

[3] R. Vaidyanathan and J. L. Trahan, Dynamic Reconfiguration: Architectures and Algorithms (Kluwer Academic/Plenum Publishers), 2004.

[4] D. Wang, A linear-time algorithm for computing collision-free path on reconfigurable mesh, J. Parallel Comput. 34, 487–496, 2008.

[5] Hatem M. El-Boghdadi. Constant Time Algorithm for Computing a Collision-Free Path on R-Mesh with Path Quality Analysis. Journal of Circuits, Systems, and Computers 24(8): 1550112:1-1550112:20. 2015.

[6] Hatem M. El-Boghdadi and Fazal Noor. A Parallel Approach to Navigation in Cities using Reconfigurable Mesh. IJCSNS International Journal of Computer Science and Network Security, VOL.21 No.4, April 2021

[7]H.-C. Lee, Effecient parallel algorithms on recon¯gurable mesh architectures, Ph.D. Dissertation, University of Missouri-Rolla, 1996. <http://www.mis.yzu.edu.tw/faculty/ hlee/csdiss/>.

[8] D. Wang, Two algorithms for a reachability problem in one-dimensional space, IEEE Trans. Syst., Man, Cybern. 28, 1998.

[9] F. Dehne, A. Hassenklover and J. Sack, Computing the con¯guration space for a robot on a mesh-of-processors, Proc. 1989 ICPP 3, 40–47 1989.

[10] P. Tzionas, A. Thanailakis and P. Tsalides, Collision-free path planning for a diamondshaped robot using two dimensional cellular automata, IEEE Trans. Robot. Automat. 13, 237–250, 1997.

[11] J. Jenq and W. Li, Computing the configuration space for a convex Robot on hypercube multiprocessors, Proc. 7th IEEE Symp. Parallel and Distributed Processing, pp. 160–167, 1995. ss

[12] Michael Hoy, Alexey S. Matveev and Andrey V. Savkin. Algorithms for collision-free navigation of mobile robots in complex cluttered environments: a survey. Robotica, volume 33, pp. 463–497, 2015.

[13] Otte, M.W. A Survey of Machine Learning Approaches to Robotic Path-Planning. 2009.

[14] Jeffrey Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, KateSaenko, andTrevorDarrell. Long-term recurrent convolutional networks for visual recognition and description. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 2625–2634, 2015.

[15] Mateusz Malinowski, Marcus Rohrbach, and Mario Fritz. Ask your neurons: A deep learning approach to visual question answering. International Journal of Computer Vision,125(1-3):110– 135, 2017.

[16] Rodrigo F Berriel, Lucas Tabelini Torres, Vinicius B Cardoso, Rânik Guidolini, Claudine Badue, Alberto F De Souza, and Thiago Oliveira-Santos. Heading direction estimation using deep learning with automatic large-scale data acquisition. 2018.

[17] Aditya Khosla, Byoungkwon An An, Joseph J Lim, and Antonio Torralba. Looking beyond the visible scene. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 3710–3717, 2014.

[18] Guillaume Lample and Devendra Singh Chaplot. Playing FPS games with deep reinforcement learning. In Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, 2017.

[19] Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andrew Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, Dharshan Kumaran, and Raia Hadsell. Learning to navigate in complex environments. arXiv preprint arXiv:1611.03673, 2016.

[20] Yi Wu, Yuxin Wu, Georgia Gkioxari, and Yuandong Tian. Building generalizable agents with a realistic and rich 3d environment. arXiv preprint arXiv:1801.02209, 2019.

[21] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J. Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In 2017 IEEE International Conference on Robotics and Automation, ICRA, pages 3357–3364, 2017.

[22] Michael J Milford, Gordon F Wyeth, and David Prasser. Ratslam: a hippocampal model for simultaneous localization and mapping. In Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on, volume 1, pages 403–408. IEEE, 2004.