Parallel implementation of A* search algorithm for road network

Safa BELHAOUS
Mathematics and computer science department
ENSET of Mohammedia
Mohammedia, Morocco
safaabelhaous@gmail.com

Sohaib BAROUD
Mathematics and computer science department
University Hassan II
Mohammedia, Morocco
sohaib.baroud89@gmail.com

Soumia CHOKRI
Mathematics and computer science department
University Hassan II
Mohammedia, Morocco
chokri.soumaya90@gmail.com

Zineb HIDILA
Mathematics and computer science department
University Hassan II
Mohammedia, Morocco
zineb.hidila@gmail.com

Abdelwahab NAJI
Mathematics and computer science department
University Hassan II
Mohammedia, Morocco
abdelwahab.naji@gmail.com

Mohammed MESTARI
Mathematics and computer science department
University Hassan II
Mohammedia, Morocco
mestari@enset-media.ac.ma

***Abstract*—Path finding is a fundamental topic in artificial intelligence, which can be used in many problems and applications. There are several algorithms developed in order to find shortest path from source to destination. The most popular of them are A*, Dijkstra and ant colony optimization. This work is based on A* search algorithm to find the optimal path among cities with reduced time. Therefore, our parallel algorithm is implemented on a shared-memory multiprocessor system which many threads running simultaneously to find the path from each neighbor of start city to goal city. The number of threads generated depends on the number of starts neighbor. Experimental results are compared with the sequential A* to evaluate a path search performance between two nodes in a graph.**

***Index Terms*—parallel A star Algorithm, road network,shared-memory processor (SMP), heuristic, path finding.**

I. Introduction

In the literature, a path finding algorithm is characterized by two inputs and one output which are respectively a source state, destination state and a short path from the source state to reach the input goal. An error is produced when the algorithm is unable to find the destination. [2].

Path finding algorithms are especially practiced in the artificial intelligence area, such as navigation of visually impaired people, strategy games, the traveling salesman problem, network management, transportation networks, web search engines, information retrieval systems and robotics [5]–[11]. The whole of those fields are differently implemented due to the searching algorithm chosen.

In this paper, A* path finding algorithm is used to find the optimal path among two nodes in a graph. In general, the sequential implementation of A* requires a lot of computation time, which makes it impractical for domains that have huge search space. Parallelizing the A* algorithm will overcome timing constraints problem, which is our main focus [14].

Writing parallel programs depend on the basic idea of partitioning the work to be done among the cores. There are two widely used approaches: task-parallelism and data-parallelism. Task-parallelism is characterized by dividing tasks among the threads or processes, on the other hand, data-parallelism is obtained by dividing data between the processors [1]. Our parallel algorithm is focused on data-parallelism approach. As we noted earlier, a source node and a destination node are provided as inputs, and the program create automatically many threads in depends on the number of starts neighbor. Instead of expanding the entire graph, every thread occupy of its part of data ensuring that every node will only be expanded at most once (the heuristic should be consistent).

The organization of this paper is as follows : Section II is focused to a background of the search algorithm and parallel programming. Section III is dedicated to present a review of the most pertinent state of the art of A* search algorithms. Section IV is devoted on the proposed solution. The experimental Results of the proposed algorithm is presented in Section V. Finally, the last section is provided to the conclusion.

978-1-7281-0003-6/19/$31.00 © 2019 IEEE

II. Background

This section presents the most interesting concepts in this work which are A* search algorithm and parallel programming by giving more detail about them.

A. A* search algorithm

The A star algorithm was described in 1968 by Hart, Nilsson and Raphael. It is a famous path finding algorithm based on a heuristic function and an evaluation function f(n) to select the next node that will be expanded. The search process (see Fig. 1) needs two lists: open and closed [15]; For the open list, all the nodes were sored due to their f(n) values and the first one will be expanded. While the closed list contains the set of nodes already expanded [16].
For a node n, its f(n) is computed as follows :

$$f(n) = g(n) + h(n) \tag{1}$$

where g(n) is the cost to achieve the node n from the source state, and h(n) is the estimated cost of reaching the destination node from the node n. The cost between nodes is not necessary to be distance. The cost could be time, if the programmer wanted to find the faster path in the graph [12] [13].
A graph is a collection of nodes and edges or line segments joining pairs of nodes. It is labeled if the nodes and/or edges have labels. In our case, the nodes of our graph correspond to the cities while the edges correspond to routes between them, and the labels on the edges correspond to the duration to traverse the routes [1]. In general, the efficient search process guarantees that its heuristic function should be admissible, i.e., h(n) is never greater than the actual cost to the destination node. If the search graph is not a tree, a stronger condition called consistency:

$$h(n) \leq d(n, m) + h(m) \tag{2}$$

where d(n, m) represents the cost or distance from n to m, guarantees that once a node is extracted from the open list, the path that it follows is optimal [18].
The heuristic h(n) [6] can be estimated in different ways depending on the allowed movement; some of these ways are :

a) Manhattan distance: It allows a move in four directions (North, South, East and West). The following equation is used to calculate the Manhattan distance:

$$h(n) = C * (|node.x - goal.x| + |node.y - goal.y|) \tag{3}$$

Where C represents the cost of moving one node to one of its neighbors. In the simple case, we can set C to be 1. The advantage of the Manhattan heuristic is that it runs faster than the other distance measures, and the main disadvantage is that it is used to find the shortest path to the goal; however, an optimal solution is not a certainty with this approach.

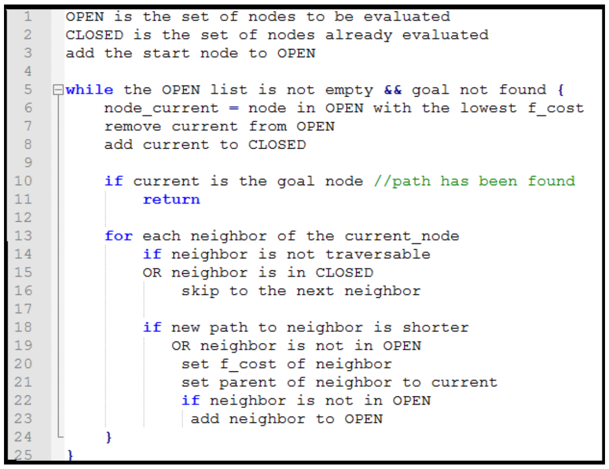
b) Diagonal distance: In this method, computation speed is slower than that in the Manhattan method. The equation used to calculate the Diagonal distance is as follows:

$$h(n) = C * max(|node.x - goal.x|, \\ |node.y - goal.y|) \tag{4}$$

This method was implemented as a function in our program to calculate the heuristic cost of each node. The signature's function is : "public double heuristic(double cost,double x_source,double y_source,double x_target,double y_target)"

c) Euclidean distance: The Euclidean heuristic is admissible, however cant estimate the real cost by a significant sum. It is also costly to apply contrasted with the Manhattan heuristic, as it additionally involves two multiplication operations and calculating the square root [14]. The equation used to calculate the Euclidean distance is as follows:

$$h(n) = \\ C * \sqrt{(node.x - goal.x)^2 + (node.y - goal.y)^2} \tag{5}$$



```
OPEN is the set of nodes to be evaluated
CLOSED is the set of nodes already evaluated
add the start node to OPEN

while the OPEN list is not empty && goal not found {
    node_current = node in OPEN with the lowest f_cost
    remove current from OPEN
    add current to CLOSED

    if current is the goal node //path has been found
        return

    for each neighbor of the current_node
        if neighbor is not traversable
        OR neighbor is in CLOSED
            skip to the next neighbor

        if new path to neighbor is shorter
           OR neighbor is not in OPEN
            set f_cost of neighbor
            set parent of neighbor to current
            if neighbor is not in OPEN
             add neighbor to OPEN
    }
}
```

Fig. 1. Pseudo code of A* search algorithm.

B. Parallel programming

Parallel hardware has been ubiquitous for some time . Its difficult to find a desktop or server that doesnt use a multicore processor. There are two main types of parallel systems namely shared-memory systems and distributed-memory systems. Well be focusing on the first one, therefore the cores can share access to the computers memory; in principle, each core can read and write each memory location.
Thread or Pthreads (for POSIX thread) and OpenMP are both APIs for shared-memory programming, they have numerous differences. Pthreads necessitates that the software engineer explicitly indicate the main job of each thread. Contrariwise, OpenMP in some cases permits the software engineer to identify which block of program must be executed in parallel, and the exact determination of the thread which can be able to execute them. This recommends a further distinction among OpebnMP and Pthreads, that will be, that Pthreads (like MPI

for distributed-memory systems) is a library of functions that can be connected to a C compiler. OpenMP, on the other hand, requires compiler support for certain operations, and consequently its altogether conceivable that you may keep running over a C compiler that cant compile OpenMP programs into parallel programs [1].

III. Related work

Barnouti et al. in [11] Were executed A* path finding algorithm to locate the most brief way between the source and goal. Their work was dedicated for strategy game by using map and maze. The map is changed over into three principle colors, on the other hand, the maze considers the image as black and white. For testing this work, the authors were used 100 different images for each map and maze. The main result was guaranteed that more than 85% images can achieve the goal by showing the best path between two points selected.

S. Zaghloul et al. in [14] proposed a parallel version of A* for finding the shortest path in a grid map. Different parameters are used to assess the performance of the parallel version of the A* algorithm; namely, the execution time, the speedup, the scalability, and the efficiency. As a result, parallelizing the A* decreases the overall execution time and increases the gained speedup of the algorithm. Therefore, a considerable improvement in performance of A* was observed.

In [16], the parallel multithreaded A* heuristic search algorithm has been executed utilizing the POSIX strings (Pthreads) library. Mahafzah gave a systematic evaluation of his proposed algorithm to solve the 15 puzzle problem. In this reason, many metrics were used to compare the parallel version with the sequential version such as speedup, number of generated nodes, number of expanded nodes and search effectiveness. The results confirmed the better performance of the parallel multithreaded A* heuristic algorithm.

Phillips et al. [17] presented PA*SE is an original parallel of A* (and weighted A*) which parallelized state extensions by exploiting this property. The results demonstrated that PA*SE returns a minimal cost path and when relaxing the independence checks the sub-optimality can be limited. One of the important hypothetical discoveries is that their autonomy rule is really a generalization of the A* expansion rule. A* just permits states with a base f-value to be extended, while PA* SE's hypothesis prove that this rule can be looser.

Choi et al. [22] presented a path search engine (PSE) for the quick optimal path search, utilizing streamlined estimation and parallel architectures to apply path search algorithm. Additionally, A* architecture is included in PSE to enhance sub-optimal issue. The PSE obtains the optimal path with the reasonable accuracy based on preference in real-time traffic information.

IV. Proposed algorithm

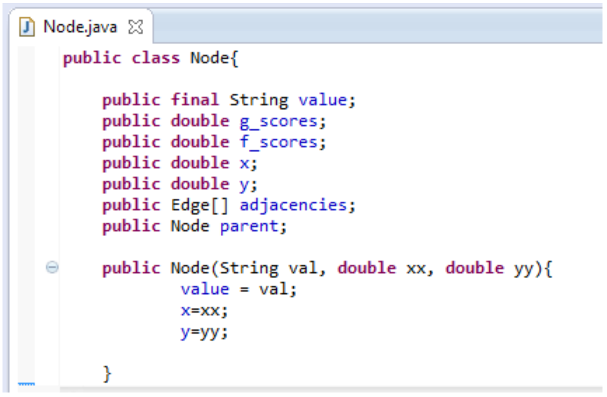
In this work, a parallel A* search algorithm was developed in java, the main objective is to find the optimal path among the start city and goal city in a parallel way. We present our proposed algorithm trough Algorithm 1. Firstly, we initialize

Algorithm 1: Parallel A* search algorithm

```
  // Graph initialization based on the
  Morocco map
  // Edges initialization of every node
1 for Edge e : n1.adjacencies do
2     e.target.parent=n1
3     n1.g_scores=0
4     AstarSearch thread=new
        AstarSearch(e.target,e.cost,goal)
5     thread.run()
6     queue.add(thread)
  // Poll() function return the thread
  with the best score
        // The best path score will be
  displaying
```

our nodes graph. In our algorithm, the node class as it shown in Fig. 2 is defined as follows "public Node(String val, double x_city, double y_city)".

Then, a start city is selected in line 1 in order to recover all



```
public class Node{

    public final String value;
    public double g_scores;
    public double f_scores;
    public double x;
    public double y;
    public Edge[] adjacencies;
    public Node parent;

    public Node(String val, double xx, double yy){
        value = val;
        x=xx;
        y=yy;

    }
```

Fig. 2. Node class structure.

its neighbors, each city is assigned to a thread . Generally, all the neighbors will be the start cities, on the other hand the destination city will remain the same for all threads. The goal of Line 2 is to ensure that the initial node wont be expanded but for its neighbors will only be expanded at most once. We talk therefore about the heuristic consistent principle. Line 4 and 5 show that every thread call AstarSearch() Java function to find the best path from their respective local start to the goal node without forgetting the cost from the first start node. Due to a priority queue java class (as it shown in line 6), we can classify threads according to their f(n) score. As a result, the best path score will be chosen as the final path.

Our algorithm has been compared with several algorithms of the third section as depicted in Table I. Each algorithm has different characteristics such as processing type, programing language, type of data, search domain, number of thread and metrics. The [16] is the most effective because it proved the better performance of the parallel multithreaded A* algorithm

TABLE I
COMPARISON BETWEEN METHODS BASED ON A*

Reference	Processing type	Programing language	Type of data	Search domain	Number of thread	Metrics
Our algorithm	Parallel	Java	Graph	Road network	8	execution time and speedup
[14]	Parallel	Java	Grid map	Path finding	8	execution time,speedup, scalability and the efficiency
[11]	Sequential	Visual Basic	Images map	Strategy game	-	System performance was tested by different map images
[16]	Parallel	POSIX Thread	Grid map	15 puzzle problem	120	speedup, number of generated nodes, number of computed distances, number of expanded nodes, number of duplicate checks, number of duplicates found and search effectiveness
[17]	Parallel	-	3D map	Robot Navigation	32	expansion time, speedup

in terms of different performance metrics. For our case, we have been proposed the first version of our algorithm which we will be to improve it by using an other programing language like Python and increasing the data size.

V. RESULTS

In this section, the proposed algorithm has been implemented in java using Eclipse Juno IDE (Integrated Development Environment). For a reasonable comparison between sequential and parallel versions of the A* algorithm, we have chosen two different graphs. The first one, is presented through Fig. 5 which represents the 24 most populous Morocco's cities(see Table III). Every city was characterized by its x and y value to compute the diagonal heuristic. The x and y real values were defined based on ArcGIS tool. It is a general purpose GIS ,Geographic Information System, software developed by ESRI. It is an extensive and integrated software platform technology for building operational GIS [20]. For example Casablanca's city is defined by (33.578617,-7.592249).
On the other hand, the cost between nodes represents the time in minute through an online tool calculator [16]. The aim of this tool is to calculate the Distance, Driving Directions and time among two positions such as cities, addresses, touristic places or airports in Morocco.

The last one, represent the 14 most popular Romania's cities such as Arad, zerind, Giurgiu and Bucharest.

In order to compare the performance of our parallel implementation, we calculate the speedup metric as follows:

$$speedup = \frac{S}{P} \tag{6}$$

The speedup is defined as the ratio of the execution time achieved when using sequential A * algorithm (S) to the execution time that is required through the parallel A* (P), as shown in (6). Usually, the execution time is the sum of time required by an algorithm to solve a problem. The sequential A* algorithms execution time is the time expended from the beginning of the search process until an answer is found. This metric is defined in the parallel A* algorithm as the total of time required by the search process since the start of the principal phase until the second phase of the algorithm is finished. Therefore, the execution time incorporates the time that is required for creating and finishing concurrent threads in addition to the time of serializing the algorithm. [16].
The speedup presents an evaluation of the performance improvement of the parallel A* algorithm over the sequential A* algorithm [21]. Table IV and Fig. 3 present our speedup results to solve the same problem. It is clear that for the two cases the parallel algorithm achieves better performance than

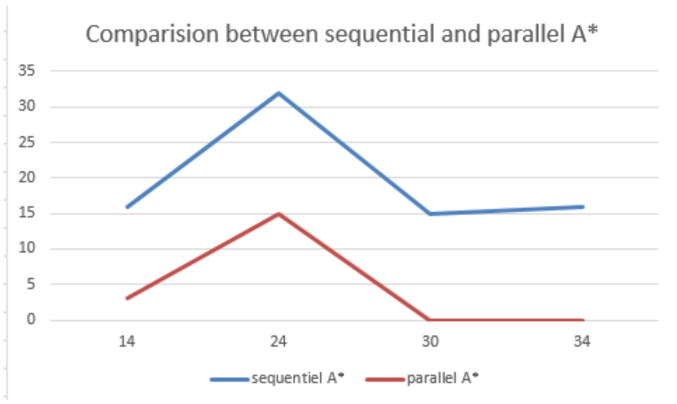


Fig. 3. The running time in both cases when we increase the graph's nodes.

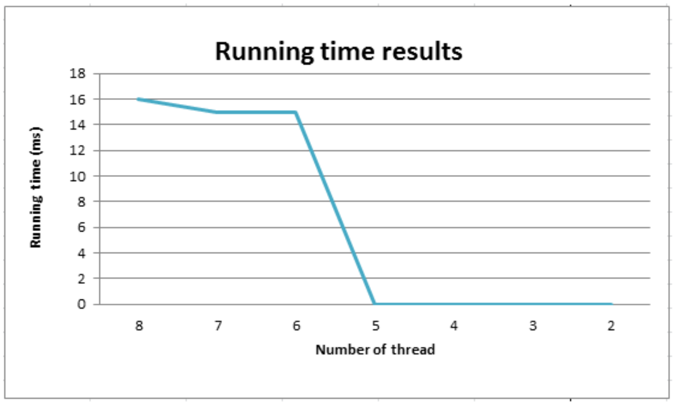


Fig. 4. Parallel A* running time according to the number of threads.

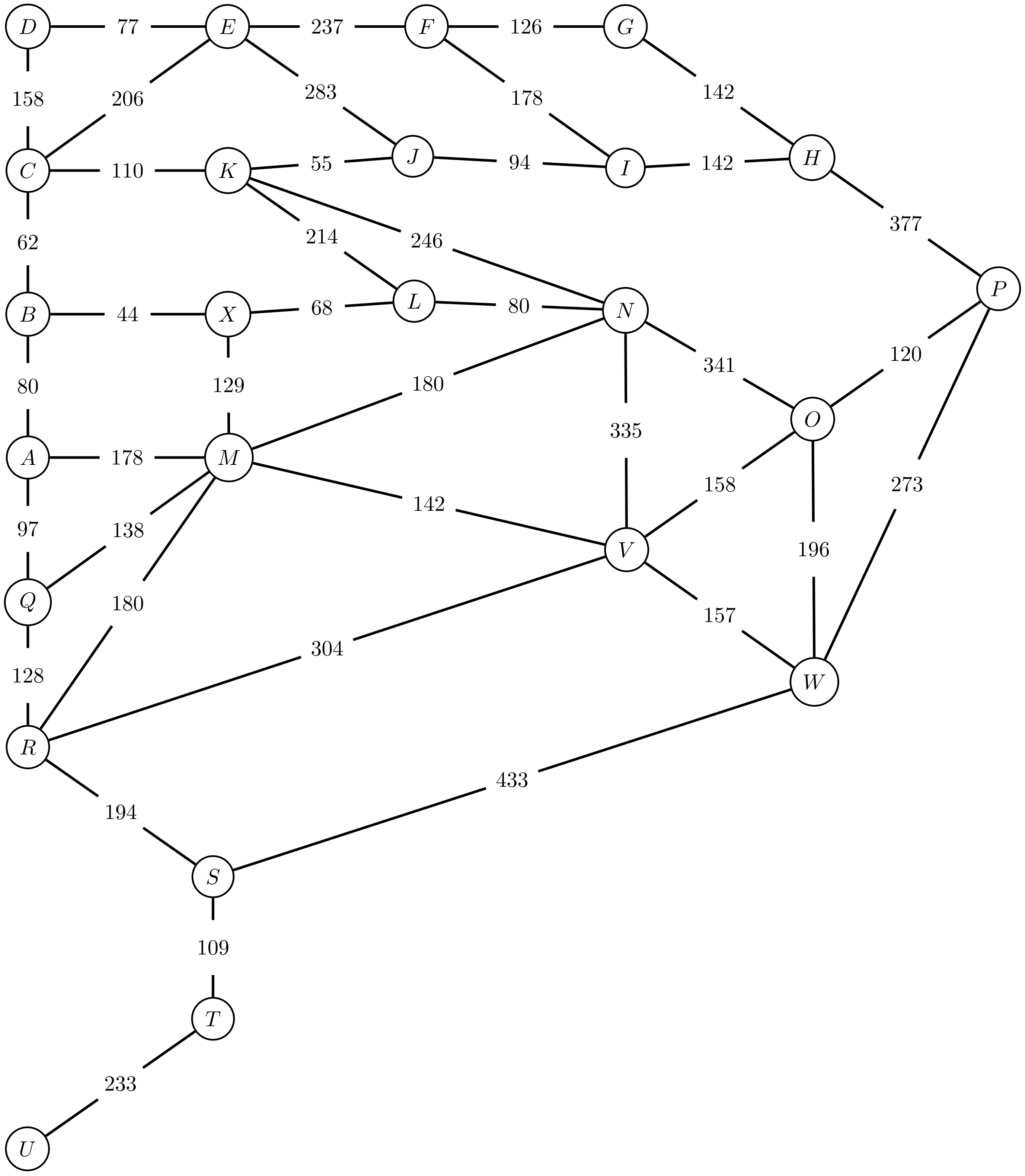


Fig. 5. Graph based on the Morocco map.

TABLE II
RUNNING TIME RESULTS

Start Node	goal Node	Number of thread	Path	Running time
Meknes	Sidi ifni	8	Meknes, Rabat, Casablanca, El Jadida, Safi, Agadir, Taroudant, Tiznit, Sidi ifni	16
Rabat	Al Ayoun	7	Rabat, Casablanca, El Jadida, Safi, Agadir, Taroudant, Tiznit, Guelmim, Tan-Tan, Al Ayoun	15
Marrakesh	Nador	6	Marrakesh, Berrechid, Casablanca, Rabat, Meknes, FeZ, Taza, Al houceima, Nador	15
Ouarzazat	Casablanca	5	Ouarzazate, Marrakech, Berrechid, Casablanca	0
Agadir	Tetouan	4	Agadir, Safi, El Jadida, Casablanca, Rabat, Tanger, Tetouan	0
Fez	Al Ayoun	3	Fez, Meknes, Rabat, Casablanca, El Jadid, Safi, Agadir, Guelmim, Tan-Tan, Al Ayoun	0
Tangier	Tinghir	2	Tangier, Rabat, Casablanca, Berrechid, Khouribga, Beni Mellal, Tinghir	0

TABLE III
GRAPH'S NODES DESIGNATION

Node	City	Node	City
A	El jadid	M	Marrakesh
B	Casablanca	N	Beni Mellal
C	Rabat	O	Tinghir
D	Tangier	P	Errachidia
E	Tetouan	Q	Safi
F	Al houceima	R	Agadir
G	Nador	S	Guelmim
H	Oujda	T	Tan-Tan
I	Taza	U	Al Ayoun
J	Fez	V	Ouarzazat
K	Meknes	W	Zagora
L	Khouribga	X	Berrechid

TABLE IV
SPEEDUP RESULTS

Number of nodes	**Sequential A* running time**	**Parallel A* running time**	**Speedup**
14	16 ms	3 ms	5.33
24	32 ms	15 ms	2.13
30	15 ms	1 ms	15
34	16 ms	1 ms	16

the sequential algorithm.
Moreover, we tested our parallel program on different cases based on the number of neighbors of the start city, and the results were promising. Table II and Fig. 4 show the effectiveness of our parallel A* according to different threads' number. They prove that the parallel A* running time is always the smallest.

From the results described above, we concluded that the parallel A* algorithm decreases the execution time of the algorithm and guarantees that the path founded is the optimal one.

VI. CONCLUSION AND PERSPECTIVES

In this paper, a Parallel implementation of A* search algorithm is proposed to solve the problems existing in the road network. Our solution is developed in Java environment using threads. The experimental results show that the parallel algorithm is effective. Therefore, its execution time is faster than the sequential version.
In our future work, we will be propose an other parallel implementation of A* algorithm using the multithreading library POSIX threads (Pthreads).

ACKNOWLEDGMENT

This project has received funding from :

- The European Unions Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement No 777720.
- The National Center for Scientific and Technical Research (NCSTR - Morocco) grant agreement No 10UH2C2017.

REFERENCES

[1] P. Pacheco, An Introduction to Parallel Programming, 1st Edition, Massachusetts, USA,Elsevier, Morgan Kaufmann, 2011.
[2] S. Russell and S. Norvig , Artificial intelligence: A modern approach, 2nd edition Englewood Cliffs, NJ: Prentice Hall, 2003.
[3] K. Chen, Heuristic search and computer game playing IV, Information Sciences 2005; 175(4): 245-246.
[4] W. Pedrycz and A. Vasilakos, "Computational intelligence in telecommunications networks," 1st edition. Boca Raton, FL: CRC Press,2000.
[5] C-F. Tsai,C-W. Tsai and C-C. Tseng, "A new hybrid heuristic approach for solving large traveling salesman problem," Information Sciences 2004; 166(1-4): 67-81.
[6] M. Kilinarslan, "Implementation of a path finding algorithm for the navigation of visually impaired people," in MS Thesis in Computer Engineering, Atilim University, Ankara,Turkey,pp.68-73, 2007.
[7] S. Perry and P. Willett, "A review of the use of inverted files for best match searching in information retrieval systems," Journal of Information Science 1983; 6(23): 5966.
[8] V. Frants, J. Shapiro and V. Voiskunskii, "Optimal search available to an individual user," Journal of Information Science 1996; 22(3): 181191.
[9] G. Singer, U. Norbisrath and D. Lewandowski, "Ordinary search engine users carrying out complex search tasks," Journal of Information Science 2013; 39(3): 346-358
[10] S-W. Chioua, "An efficient search algorithm for road network optimization," Applied Mathematics and Computation 2008; 201(12): 128137.
[11] N.H. Barnouti, S.S.M. Al-Dabbagh, and M. A. S. Naser, "Pathfinding in Strategy Games and Maze Solving Using A Search Algorithm", Journal of Computer and Communications, p. 15, 2016.
[12] X.Cui, and H.Shi,"A*-Based Pathfinding in Modern Computer Games," International Journal of Computer Science and NetworkSecurity, 11, 125-130.
[13] A.Ansari, M.A.Sayyed, K.Ratlamwala, and P.Shaikh, (2015) "An Optimized Hybrid Approach for Path Finding" [Online] available : https://arxiv.org/ftp/arxiv/papers/1504/1504.02281.pdf
[14] S. S. Zaghloul, H. Al-Jami, M. Bakalla, L. Al-Jebreen, M. Arshad,A. Al-Issa,"Parallelizing A* Path Finding Algorithm" International Journal Of Engineering And Computer Science ISSN:2319-7242. Volume 6 Issue 9 September 2017, Page No. 22469-22476.
[15] P. Hart, N.Nelson and B.Raphael "A formal basis for the heuristic determination of the minimum cost paths" IEEE Transactions on Systems Science and Cybernetics; 4(2): 100107, 1968.
[16] B.A. Mahafzah,"Performance evaluation of parallel multithreaded A* heuristic search algorithm" journal of Information Science, Vol. 40(3) ,pp. 363375, 2014.
[17] M. Phillips, M. Likhachev, and S. Koenig, "PA* SE: Parallel A* for Slow Expansion," in the Twenty-Fourth International Conference on Automated Planning and Scheduling, New Hampshire, USA, pp. 208-216,2014.

[18] J. Pearl,“Heuristics” Addison-Wesley Publishing Company Reading, Massachusetts, 1984.

[19] Distance Calculator and Driving Directions Morocco,[Online]. Available: https://distancecalculator.globefeed.com/Morocco_Distance _Calculator.asp.

[20] D.Maguire (2008), “ArcGIS: General Purpose GIS Software System” In: Shekhar S., Xiong H. (eds) Encyclopedia of GIS. Springer, Boston, MA.

[21] J.Hennesy and D.Patterson, “Computer architecture: A quantitative approach,” 3rd edition. San Francisco, CA: Morgan Kaufmann,2003.

[22] I. Choi, T. Han ,I. Kim and S. Kang , “Path search engine for fast optimal path search using efficient hardware architecture,” in International SoC Design Conference (ISOCC),pp. 9699, 2011.