Performance Improvement of Path Planning algorithms with Deep Learning Encoder Model

1st Janderson Ferreira  
department of Computer Engineering.  
Polytechnic School (UPE)  
Recife, Brazil  
jrb@ecomp.poli.br

2nd Agostinho A. F. Júnior  
department of Computer Engineering.  
Polytechnic School (UPE)  
Recife, Brazil  
aafj@ecomp.poli.br

3rd Yves M. Galvão  
department of Computer Engineering.  
Polytechnic School (UPE)  
Recife, Brazil  
ymg@ecomp.poli.br

4th Pablo Barros  
Cognitive Architecture for Collaborative Technologies  
Istituto Italiano di Tecnologia (IIT)  
Genova, Italy  
pablo.alvesdebarros@iit.it

5th Sergio Murilo Maciel Fernandes  
department of Computer Engineering.  
Polytechnic School (UPE)  
Recife, Brazil  
smmf@ecomp.poli.br

6th Bruno J. T. Fernandes  
department of Computer Engineering.  
Polytechnic School (UPE)  
Recife, Brazil  
bfjtf@ecomp.poli.br

Abstract—Currently, path planning algorithms are used in many daily tasks. They are relevant to find the best route in traffic and make autonomous robots able to navigate. The use of path planning presents some issues in large and dynamic environments. Large environments make these algorithms spend much time finding the shortest path. On the other hand, dynamic environments request a new execution of the algorithm each time a change occurs in the environment, and it increases the execution time. The dimensionality reduction appears as a solution to this problem, which in this context means removing useless paths present in those environments. Most of the algorithms that reduce dimensionality are limited to the linear correlation of the input data. Recently, a Convolutional Neural Network (CNN) Encoder was used to overcome this situation since it can use both linear and non-linear information to data reduction. This paper analyzes in-depth the performance to eliminate the useless paths using this CNN Encoder model. To measure the mentioned model efficiency, we combined it with different path planning algorithms. Next, the final algorithms (combined and not combined) are checked in a database that is composed of five scenarios. Each scenario contains fixed and dynamic obstacles. Their proposed model, the CNN Encoder, associated to other existent path planning algorithms in the literature, was able to obtain a time decrease to find the shortest path in comparison to all path planning algorithms analyzed. The average decreased time was 54.43%.

Index Terms—component, formatting, style, styling, insert

I. INTRODUCTION

Path planning algorithms are essential for the accomplishment of many activities in different areas, for example, robot navigation [1], path apps for locomotion in cities (for pedestrian and driver) [2], autonomous-driver cars [3]. These algorithms have different approaches to treat spatial information, the most used in the literature are, Grid-based search (which transforms the environment in a grid-mesh) [4], Interval-based search (similar to grid-based search it but uses space data instead of a grid) [5] and Reward-based (similar to a reinforcement learning in deep learning) [6].

Based on Grid-based search, the first path planning algorithm was proposed by Dijkstra in 1956. Although this solution is always able to find the shortest path between two points, Dijkstra’s algorithm has become obsolete because it has very high computational complexity. Considering the response time of the Dijkstra algorithm, it would be infeasible to be applied in many scenarios.

Given this problem, new algorithms with different approaches were created to improve performance in finding the shortest path between two points.

Since Dijkstra’s proposal, many algorithms have been created being able to find the shortest path with the lowest computational cost. A* [4], Bi A* [7], Breadth-first [8], Best-First [9] are a few of the search algorithms that exist for path planning. They have peculiarities that tackle different problems, and, therefore, are useful and important in many areas. These algorithms, however, become extremely costly when applied to large environments or environments with dynamic objects [10]. All these algorithms are detailed in the theoretical foundation section [II-A].

The problem of the increased computational cost when increasing the amount of information is a problem that affects several fields of research, for example, pattern recognition [11], computer vision [12], text mining [13]. A very well-known approach to avoid this problem is to reduce dimensionality by discarding irrelevant information to the task.

Principal component analysis (PCA) is a mathematical procedure based on orthogonal transformation to convert data into a set of values of linearly unrelated variables called principal components. The number of principal components is always less than or equal to the number of original variables [14].

Truncated Singular value decomposition (TSVD) This algorithm use means of TSVD to performs linear dimensionality reduction. Differently of PCA, this solution does not center the
data before computing the singular value decomposition [15].

Non-negative matrix factorization (NMF) Creates two non-negative matrices (W, H). The product of these matrices is an approximation of the non-negative input data. This method is used for dimensionality reduction [16].

These solutions significantly reduce dimensionality, keeping enough information to accomplish some tasks. However, a limitation of these approaches is since they reduce dimensionality using only linear correlation [17]. Therefore, in [18], the authors built a deep learning model able to reduce dimensionality using non-linearity correlation, named Convolutional Neuronal Network (CNN) Encoder. Their method removes mostly the useless information of the input data, including in dynamic environments. Eliminate useless information for path planning problem means to remove the paths which do not connect the start point and the goal point.

In this work, we perform an in-depth evaluation of the application of their proposed CNN Encoder to decrease the time spent by different path planning algorithms, showing the efficiency of the proposal to improve various path planning solutions.

II. MATERIAL AND METHODS

A. Theoretical Foundation

1) Dijkstra’s algorithm: The Dijkstra’s algorithm proposed in 1956 by Edsger W. Dijkstra [19] is a path planning algorithm based on graph search. It solves the single-source shortest path problem for a graph with non-negative edge path costs, producing a shortest-path tree, it can be defined as follows:

\[
\text{Algorithm 1: Dijkstra’s algorithm}
\]

\[
\begin{align*}
\text{Result: } & \text{H, dist} \\
\text{Initialization: } & \text{} \\
\text{for each vertex } v \text{ in } V_G & \text{ do} \\
& \text{dist}_v \leftarrow \infty; \\
& \text{parent}_v \leftarrow \text{NIL} \\
\text{end} \\
& \text{dist}_s \leftarrow 0; \\
\text{while } Q \neq \emptyset & \text{ do} \\
& \text{u} \leftarrow \text{ExtractMin}(Q); \\
& \text{for each edge } e = (u, v) \text{ do} \\
& \text{dist}_v \leftarrow \infty; \\
& \text{if dist}_u > \text{dist}_u + \text{weight}_e \text{ then} \\
& \text{dist}_v \leftarrow \text{dist}_u + \text{weight}_e; \\
& \text{parent}_v \leftarrow u \\
\text{end} \\
\text{end} \\
& H \leftarrow (V_G, \emptyset); \\
& \text{for each vertex } v \in V_G, \text{ v} \neq s \text{ do} \\
& E_H \leftarrow E_H \cup \{(\text{parent}_v, v)\}; \\
& \text{return } H, \text{dist}
\end{align*}
\]

2) A*: The A* algorithm [20] was proposed to solve the limitations of Dijkstra’s algorithm, and to overcome it in the time spent to find the shortest path. This solution uses heuristics to be faster. It is defined as:

\[
\text{Algorithm 2: A* algorithm}
\]

\[
\begin{align*}
\text{Result: } & \text{Shortest path} \\
\text{openset} & \leftarrow \emptyset; \\
& \text{closedset} \leftarrow \emptyset; \\
& \text{came from} \leftarrow \emptyset; \\
& \text{heuristic cost estimate} \leftarrow 0; \\
& \text{g score} \leftarrow 0; \\
& \text{fscore} \leftarrow 0; \\
& \text{g score}[\text{start}] \leftarrow 0; \\
& \text{fscore}[\text{start}] \leftarrow \text{heuristic cost estimate}(\text{start}, \text{goal}); \\
\text{while openset} \neq \emptyset & \text{ do} \\
& \text{current} \leftarrow \text{the node in openset having the lowest f score value}; \\
& \text{if current} = \text{goal} \text{ then} \\
& \text{return } \text{Reconstruct Path}(\text{came from}, \text{goal}); \\
& \text{end} \\
& \text{remove current from openset}; \\
& \text{add current to closedset}; \\
& \text{for each neighbor in neighbor nodes(current) do} \\
& \text{if neighbor in closedset then} \\
& \text{continue}; \\
& \text{end} \\
& \text{tentative g score} \leftarrow \text{g score}[\text{current}] + \text{dist between}(\text{current, neighbor}); \\
& \text{end} \\
& \text{if neighbor not in openset or tentative g score} < \text{g score}[\text{neighbor}] \text{ then} \\
& \text{came from}[\text{neighbor}] \leftarrow \text{current}; \\
& \text{g score}[\text{neighbor}] \leftarrow \text{tentative g score}; \\
& \text{fscore}[\text{neighbor}] \leftarrow \text{g score}[\text{neighbor}] + \text{heuristic cost estimate}(\text{neighbor, goal}); \\
& \text{if neighbor not in openset then} \\
& \text{add neighbor to openset}; \\
& \text{end} \\
& \text{return failure}
\end{align*}
\]

3) Bi A*: Bidirectional A* search is a graph search algorithm that finds the shortest path from an initial vertex to a goal vertex in a directed graph running two simultaneous searches [7]. It can be described as:

4) Breadth-first: The Breadth-first Search [8] is a classic graph search algorithm, and it works by expanding and systematically exploring a given node and progressively redoing the same procedure for all its neighbors. At each iteration, the last one explored, but not expanded, or visited, is selected. Also, this algorithm discovers all nodes that are a certain distance from the start.

5) Best-First: Based on the different strategies for solving search problems, the Best-First [9] algorithm is one of the most
Algorithm 3: Bi A*

**Result:** $H, \text{dist}$

$\text{Astart}_1 \leftarrow \text{Astart}(\text{start}, \text{goal})$;

$\text{Astart}_2 \leftarrow \text{Astart}(\text{goal}, \text{start})$;

while $\text{Astart}_1.\text{current} \neq \text{Astart}_2.\text{current}$ do

path one $\leftarrow \text{Astart}_1.\text{run}()$;

path two $\leftarrow \text{Astart}_2.\text{run}()$;

end

shortest path $\leftarrow$ path one $+$ path two;

$H \leftarrow$ (shortest path);

return $H$

Algorithm 4: Breadth-first algorithm

**Input:** nodes, source

**Result:** tree

$\text{start} \leftarrow \{\text{source}\}$;

$\text{next} \leftarrow \{}$;

$\text{parents} \leftarrow [-1, -1, ..., -1]$;

while $\text{start} \neq \{\}$ do

for $v \in \text{start}$ do

for $n \in \text{neighbors}[v]$ do

if $\text{parents}[n] = -1$ then

$\text{parents}[v] \leftarrow v$;

$\text{next} \leftarrow \text{next} \cup \{n\}$;

end

end

$\text{start} \leftarrow \text{next}$;

$\text{next} \leftarrow \{}$;

end

return tree

I. Related Work

Algorithm 5: Best-First algorithm

**Input:** Graph, start, goal

$\text{PriorityQueue} \leftarrow \{}$;

$\text{PriorityQueue} \leftarrow \text{start}$;

while $\text{PriorityQueue} \neq \{\}$ do

$u \leftarrow \text{removeMin}(\text{PriorityQueue})$;

if $u = \text{goal}$ then

exit;

end

for each neighbor $v$ of $u$ do

if $v$ is unvisited then

mark $v$ as visited;

$\text{PriorityQueue} \leftarrow v$;

end

end

mark $u$ as exploited;

end

Popular in the literature. Given the heuristic function $F(n) = h(n)$, which is applied equally throughout the search space, the algorithm aims to use this to quantify the value of each candidate exploited during the process and, thus, continue the exploration until reach the point of interest.

B. Path Planning Database

Following the research of Janderson et al. previous work and thus demonstrating the efficiency of their proposal concerning the conventional approaches, we use the image database that was proposed by them, to expand previous results through the comparison with other solutions present in the literature. It contains various formats, distributed in five different scenarios. They have variations of start, goal points for each instance in the database; consequently, the possibilities of paths to be taken. In Figure 2, it is possible to see an instance of each scenario.

Using their database is possible to perform a more detailed analysis of the model’s ability to generalize its responses, as well as its level of efficiency when compared to other solutions in different scenario configurations. As mentioned before, their database contains five scenarios, where each one has a total of 10000 scenes, which are RGB images with a resolution of 60 x 60 pixels. The variation between them is due to the random positioning of obstacles; which simulates physical obstacles. Also, for each one of the images, there is a label, being a GrayScale 60 x 60 image, which contains the shortest path of the scene.

![Example Scenes](image1.png)

Fig. 1. The instances of the used database. The yellow pixels represents the start and, the gray ones, the end of the path.
Autoencoders are being used to code data information in unsupervised learning [7]. They are trained to reconstruct the input data using fewer data than the original input; this way, many times, they can eliminate useless information. On the other hand, Convolutional Neural Networks has an excellent capability to extract high-level features in tasks of deep learning and computational vision problems [21]. Trying to get the best of each model, Janderson et al. built a CNN Encoder to reduce the dimensionality of the data. This solution can eliminate useless routes from 2D maps [18].

1) The model architecture: The architecture used was obtained through a testing process, where its construction took place through adjustments based on the results generated. Finally, the architecture reached can be analyzed in Table I.

TABLE I
ARCHITECTURE OF THE CNN ENCODER

| Layer | Filters | Kernel Size | Activation | Batch Norm | Dropout |
|-------|---------|-------------|------------|------------|---------|
| 1     | -       | -           | -          | -          | -       |
| 2     | Conv    | 64          | 3x3        | ReLu       | True    |
| 3     | Conv    | 128         | 3x3        | ReLu       | False   | 30%     |
| 4     | Max-Pool| -           | 3x3        | -          | -       |
| 5     | Conv    | 256         | 3x3        | ReLu       | False   | 30%     |
| 6     | Conv    | 512         | 3x3        | ReLu       | False   | 30%     |
| 7     | Dense   | 256         | -          | LeakyReLU  | True    | 30%     |
| 8     | Dense   | 512         | -          | LeakyReLU  | True    | 30%     |
| 9     | Dense   | 1024        | -          | LeakyReLU  | True    | 30%     |
| 10    | Dense   | 3600        | -          | Tanh       | False   | 30%     |

D. Experimental Setup

In this section, we describe the metrics used to evaluate the obtained results, how the database was manipulated, and the output processing. We also specify the hardware characteristics of the desktop computer used to perform the experiments; this is important because the hardware influences some experiments.

1) Hardware Specification: The algorithms were written in python 3.7 and implemented in an Intel i5-8400 six-core, with a base frequency of 2.80GHz and 8GB of RAM. More details are depicted in Table II.

TABLE II
SPECIFICATIONS OF THE DESKTOP COMPUTER USED TO RUN THE EXPERIMENTS

| Model | Intel(R) Core(TM) i5-8400GHz |
|-------|-------------------------------|
| Number of Cores | 6 |
| Number of Threads | 6 |
| Base Frequency | 2.80GHz |
| Cache Size | 9MB |
| RAM | 8GB |

2) Metrics:
- Number of interactions. Represents the number of attempts until the algorithm found the shortest path.
- The path planning algorithm time. Represents the time in seconds that the algorithm spent to find the shortest path.
- CNN Encoder and preprocess output time. Represents the time in seconds that the CNN Encoder spent to predict the data input plus the time to preprocess it.
- Total time.

3) Database Division: Percentage Split
- Train 80%
- Validation 10%
- Test 10%

III. EXPERIMENTAL RESULTS

To evaluate this work, we applied the path planning algorithms mentioned in the theoretical foundation to the path planning database, and compare the average number of iterations when the CNN Encoder is combined or do not with these algorithms. Also, we compare the average of the time spent with and without using the proposal. The results were obtained using the test set (1000 images for each scene).

To facilitate the visualization of the results, they were compiled and divided into three tables. Table III shows the number of interactions of each algorithm for each scenario separately; this is important to analyze the model behaviour in different scenarios. Also, Table III shows the percentage of improvement between the algorithms when using or not using the model. On the other hand, Table III shows the spent time of the algorithms to find the shortest path for each scenario. Also, the improvement time with and without the model was calculated.

In the conventional A* algorithm, we obtained an average improvement of 59.87% in terms of number of iterations.
Looking at the execution time, with the application of the solution, we achieved an average improvement of 49.87%.

Going to the Best-First search algorithm, we achieved an average improvement of 53.87% in terms of number of iterations, also, in execution time, an average improvement of 16.06%.

Using a variation of the first tested algorithm, Bi A*, an average improvement of 56.12% was presented in terms of number of iterations, in the execution time, an average improvement of 40.88%.

We also used the Breadth-first algorithm, which is very widespread in the literature, we managed to achieve an average improvement of 83.77% in terms of number of iterations, in the execution time, an average improvement of 65.46%.

Finally, we applied the proposal to the Dijkstra algorithm, we obtained an average improvement of 84.12% in the number of iterations and looking at the execution time, there was an average improvement of 81.24%.

In Table IV, it is possible to see a compilation of our results, containing a summary comparison between the standard execution of the search algorithms and their new results after the addition of our proposal. Also, it is possible to see some qualitative results in Figure 3.

### TABLE III

| Scene 1 | Scene 2 | Scene 3 | Scene 4 | Scene 5 |
|--------|--------|--------|--------|--------|
| A*     | 910.95 | 623.35 | 449.56 | 766.74 | 384.32 |
| CNN Encoder + A* | 301.42 | 246.30 | 221.70 | 254.06 | 254.07 |
| Improvement | 66.91% | 60.42% | 50.50% | 56.80% | 56.32% |
| Total Improvement | 59.37% | 126.42% | 147.63% | 162.69% | 167.31% |
| Bi A* | 946.41 | 491.44 | 458.24 | 665.34 | 499.88 |
| CNN Encoder + Bi A* | 347.61 | 254.88 | 227.00 | 272.18 | 291.39 |
| Improvement | 61.09% | 68.14% | 50.40% | 50.99% | 50.51% |
| Total Improvement | 55.37% | 128.42% | 152.69% | 155.31% | 155.31% |

### TABLE IV

| Time (s) | Scene 1 | Scene 2 | Scene 3 | Scene 4 | Scene 5 |
|----------|--------|--------|--------|--------|--------|
| A* | 0.027 | 0.021 | 0.013 | 0.018 | 0.017 |
| CNN Encoder + A* | 0.016 | 0.010 | 0.007 | 0.009 | 0.009 |
| Improvement | 51.25% | 31.01% | 28.79% | 49.69% | 46.11% |
| Total Improvement | 49.87% | 40.38% | 28.79% | 49.69% | 46.11% |
| Bi A* | 0.019 | 0.013 | 0.004 | 0.005 | 0.005 |
| CNN Encoder + Bi A* | 0.008 | 0.004 | 0.004 | 0.004 | 0.004 |
| Improvement | 34.94% | 50.54% | 14.42% | 24.42% | 24.96% |
| Total Improvement | 16.96% | 20.37% | 20.37% | 20.37% | 20.37% |
| Dijkstra | 0.024 | 0.013 | 0.011 | 0.017 | 0.017 |
| CNN Encoder + Dijkstra | 0.016 | 0.010 | 0.008 | 0.009 | 0.009 |
| Improvement | 55.08% | 30.37% | 28.37% | 35.76% | 30.87% |
| Total Improvement | 65.46% | 40.38% | 30.37% | 35.76% | 30.87% |
| Best-first | 0.013 | 0.015 | 0.013 | 0.014 | 0.014 |
| CNN Encoder + Best-first | 0.006 | 0.004 | 0.004 | 0.004 | 0.004 |
| Improvement | 66.69% | 85.37% | 84.12% | 83.77% | 83.77% |
| Total Improvement | 81.24% | 75.42% | 74.94% | 74.94% | 74.94% |

### TABLE V

| Scene 1 | Scene 2 | Scene 3 | Scene 4 | Scene 5 |
|--------|--------|--------|--------|--------|
| Imprct | 45.76% | 28.79% | 22.46% | 22.46% |
| CNN Encoder + Dijkstra | 0.043 | 0.056 | 0.045 | 0.049 | 0.049 |
| Improvement | 77.07% | 83.73% | 82.33% | 81.89% | 81.89% |
| Total Improvement | 81.24% | 81.24% | 81.24% | 81.24% | 81.24% |

IV. CONCLUSION

This work aimed to show that it is possible to improve the performance of path planning algorithms using a CNN Encoder to eliminate useless routes.

From the results obtained, we can assume that it is more advantageous to apply the CNN encoder to the existing path planning techniques. That is, the proposal was able to reduce the time to find the shortest path with all analyzed algorithms. In fact that CNN Encoder can eliminate routes in scenarios with fixed and dynamic obstacles, which may help in research with robotic navigation. As such, our contribution is to validate the architecture’s efficiency with different solutions from the path planning literature.

As future work, we hope to check the proposed model for the creation of a socially aware motion planning algorithm [22].

Also, we intend to combine new Deep Learning features with improving the architecture, which may reduce the response time even further.

REFERENCES

[1] Nayan M Kakoty, Mridusmita Mazumdar, and Durlav Sonowal. Mobile robot navigation in unknown dynamic environment inspired by human pedestrian behavior. In Progress in Advanced Computing and Intelligent Engineering, pages 441–451. Springer, 2019.
[2] Baiwei Peng, Lan Zhang, Haichuan Ding, Yuguang Fang, and Shiqiang Chen. Spath: Finding the safest walking path in smart cities. IEEE Transactions on Vehicular Technology, 58(7):7071–7079, 2019.
[3] Meixin Zhu, Xuesong Wang, and Yinhai Wang. Human-like autonomous car-following model with deep reinforcement learning. Transportation research part C: emerging technologies, 97:348–368, 2018.
[4] David Šíslák, Premysl Volf, and Michal Pechoucek. Accelerated a* trajectory planning: Grid-based path planning comparison. In 19th International Conference on Automated Planning and Scheduling (ICAPS), Thessaloniki, Greece, Sept pages 19–23. Citeseer, 2009.
[5] Andre Gaschler, Ingrid Kessler, Ronald P. A. Petrick, and Alois Knoll. Extending the knowledge of volumes approach to robot task planning with efficient geometric predicates. In 2015 IEEE International Conference on Robotics and Automation (ICRA), IEEE, may 2015.
[6] J. Jeong, W. Ko, G. Park, D. Kim, Y. Yoo, and J. Kim. Task intelligence with efficient geometric predicates. In 2019 IEEE International Conference on Automated Planning and Scheduling (ICAPS), Thessaloniki, Greece, Sept, pages 18–27. Citeseer, 2019.
[7] Eric P Lafortune and Yves D Willems. Bidirectional path tracing. 1993.
[8] Maceij Kurant, Athina Markopoulou, and Patrick Thiran. On the bias of the trajectory planning algorithm. In 20th International Teletraffic Congress (ITC 22), pages 1–8. IEEE, 2010.
Fig. 3. Comparison of the search space in an instance of each class, in the left column, the original scenes, in the right, the new search space after the application of the solution.