Artificial Intelligence and Machine Learning: Search and ML Clustering

1st Given Edward Patch
Software Engineer (of BSc Year 3)
Artificial Intelligence and Machine Learning
University of Wales Trinity St. Davids (of Gordon Dickers)
Swansea, Wales
Student ID: 1801492

***Abstract*—Artificial Intelligence and Machine Learning play a considerable part in Computer Science. Visualisation of Depth First Search and KMeans algorithms with mathematical representations and explanations. Performance testing of each algorithm gives an idea of what algorithm provides an accurate and quick method to process any given data in different scenarios.**

***Index Terms*—Deep First Search, KMeans, Algorithms.**

I. Introduction

Artificial Intelligence (AI), commonly known as Machine Learning (ML), are built-up with many algorithms to provide tools to create AI and ML solutions.Research and implementation of two types of algorithms, *Search Algorithms* and *ML Clustering Algorithms* to solve two given problems found within AI development.

AI algorithms enable the developer to reuse algorithms to identify different items, generally in a recursive state. For instance, if a text file with the text stored as 'Find me' was mixed with many other duplicate text files, then different algorithms could find precise information, whether it is a date and time or author's name from the file's metadata, creating a new identity to the found file.

II. Search Algorithms

Depth First Search (DFS) algorithm provides a method of finding a node through a tree-based search, refer to Fig. 1 using a Stack data structure. The data structure stores new memory on top of recent memory within the Stack. A Stack only allows access to the last data added to the Stack. The algorithm selects the first node that is pre-identified before the algorithm starts. After this step, the algorithm will loop through each node, adding the latest node to the Stack, and if the node has no other pathways, the algorithm pops each node until the following path becomes available. This recursion continues until the finding of the search node. Although, if a particular node does not exist, the algorithm will not break, causing an infinite loop unless there is a 'depth' variable. The process of the DFS algorithm summarised by Vigneshwaran Palanisamy's [1], Journal Article, 'A Novel Agent Based Depth First Search Algorithm' as quoted 'DFS is about to start a source vertex s and travel the graph level by level, and it first visits all vertices which are adjacent to the s [5].' The time complexity of the DFS algorithm according to Vigneshwaran Palanisamy [1] '...DFS will be $\mathcal{O}(\mathcal{V} + \mathcal{E})$, where $\mathcal{V}$ is the number of vertices and $\mathcal{E}$ is the number of Edges.' It also states that if a matrix data structure is present, then it is noteworthy to measure the time complexity as $\mathcal{O}(\mathcal{V}^2)$.

Dijkstra algorithm applies the Breadth-First Search (BFS) algorithm logic. The BFS algorithm functions with Queues rather than a Stack data structure. When the next available pathway becomes available, an addition of the following pathway to the queue's end, so when the objective node is not present, the algorithm will continue to the subsequent nodes on the next queue. Vigneshwaran Palanisamy [2], 'Cluster Based Multi Agent System for Breadth First Search' mentions the time complexity of the Breadth-First Search is $\mathcal{O}(\mathcal{V} + \mathcal{E})$. Again, if the data structure uses a matrix data structure, then the time complexity is equal to $\mathcal{O}(\mathcal{V}^2)$.

Dijkstra algorithm appears in different scenarios like AI pathfinding found in-game scenarios, where the enemy character tries finding the shortest pathway to find the main character. These algorithms weigh each pathway and select the best pathway that costs less time to complete. Instead of using queues, the Dijkstra algorithm uses weighted graphs to calculate the shortest path. According to 'A Note on the Complexity of Dijkstra's Algorithm for Graphs with Weighted Vertices' written by Michael Barbehenn [3] 'Therefore, the complexity of Dijkstra's algorithm for vertex-based cost functions is $\mathcal{O}(|\mathcal{E}| + |\mathcal{V}| \log |\mathcal{V}|)$ using a binary heap implementation for the priority queue.'

A CA-ICSE algorithm tree-based search to represent the house-style of how these search algorithms work, Fig. 1. The design of the CA-ICSE algorithm is to find a specific webpage in a similar manor of BFS and Dijikstra Algorithm by trying to find the shortest path but instead of calculating the path cost, the algorithm works by finding a similar node identical to it's current one. The following quote explains how CA-ICSE algorithm works.

> 'In addition, the CA-ICSE algorithm can be used to reduce the clustering time of cluster search engine system. For example, as Fig. 4 shows, if there are 4 nodes at each level in the meta-directory tree, then the CA-ICSE algorithm can eventually cut the search space down to 25% of the whole

search space because we only look at the most similar node and then iterate. In other words, we only look at 4 nodes at each level instead of all the nodes at each level. For this reason, this clustering algorithm can find the suitable categories (nodes) rapidly' - Chun-Wei Tsai [4].

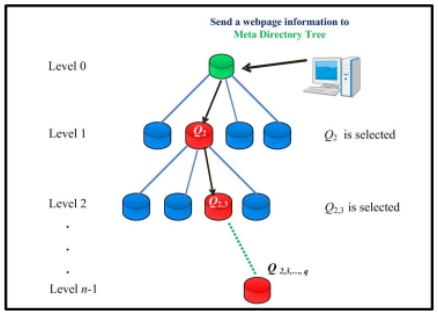

Fig. 1. Tree-Based Search Representation

Comparing the previously noted time complexities of DFS, BFS and Dijkstra algorithms, the following formulae offers an idea of how to calculate the time of completion for each one.

Time Complexities Formulae

$\mathcal{V}$: Vertices — $\mathcal{E}$: Edges

DFS: $\mathcal{O}(\mathcal{V}+\mathcal{E})$

BFS: $\mathcal{O}(\mathcal{V}+\mathcal{E})$

Dijkstra: $\mathcal{O}(\mathcal{V}+\mathcal{E}(\log \mathcal{V}))$

With $\mathcal{E}=8$ and $\mathcal{V}=6$, DFS and BFS algorithms calculates to the time complexity of $\mathcal{O}(14)$ and Dijkstra algorithm is equal to the time complexity of $\mathcal{O}(13.42)$ rounded-up by two decimal places. DFS and BFS algorithms have similar execution times based solely on the time complexity. Dijkstra algorithm has a time complexity difference of $\mathcal{O}(0.58)$ compared to either DFS or BFS algorithms. However, due to the BFS algorithm being more memory intensive, the DFS algorithm will empty memory to find the next available pathway. This reasoning proves that even though both have the same time complexity, the BFS algorithm may prove slower than the DFS algorithm if the BFS algorithm causes the system memory to clog up. However, the BFS algorithm could prove quicker than the DFS algorithm if an infinite loop occurs.

For this experiment, the DFS algorithm must run in an environment capable of reading a text file with the ability of the user to enter a search node for the algorithm to find. The text file represents a tree-based search, similar to the Fig. 1. The environment consists of several rules for the DFS algorithm to follow. This environment creates a test environment to provide different scenarios for the DFS algorithm implementation. These rules consist of:-

- An ability to load different files with dynamic height and width *(Constrained to a 2-by-2 grid).*
- Ascii '42' characters are out-of-bounds.
- Ascii '32' characters are pathways.
- Ascii '83' character is a start position.
- Ascii '70' character is a finish position.
- Ascii '48 - 57 and 65 to 90' characters are nodes.
- Stack data structure to repsresent a DFS algorithm setup.

Note: The input file to test the DFS algorithm implementation is 'm1.txt', and the assignment of the search node is 'E'. The content of the input file, Fig. 2. Results of the DFS algorithm console logs found within Fig. 3. Graph Results using MATLAB to plot the data from the CVS file generated the DFS implementation. MATLAB DFS application enables the user to control the depth to see the program's visualisation. Check out Fig. 4.

```
************
*S     A    *
***** *****
*C*         *
* ******* *
* *         *
* ** ******
* *B      D*
* **** ** *
*E         F*
************
```

Fig. 2. DFS Algorithm: Input File

```
Enter filename: m1.txt
Enter Node to Find (Char): E
************
*S     A    *
***** *****
*C*         *
* ******* *
* *         *
* ** ******
* *B      D*
* **** ** *
*E         F*
**********
N: S X: 1 | Y: 1 | Last Direction
    Enum: 0 | Current Direction
    Enum: 0
N: A X: 1 | Y: 6 | Last Direction
    Enum: 1 | Current Direction
    Enum: 1
N: S X: 1 | Y: 1 | Last Direction
    Enum: 0 | Current Direction
    Enum: 5
N: B X: 7 | Y: 3 | Last Direction
    Enum: 3 | Current Direction
    Enum: 5
N: E X: 9 | Y: 1 | Last Direction
    Enum: 3 | Current Direction
    Enum: 5
Generated Necessary Data Files for
    Matlab:
Go to the DFSMatlab folder and open
    DFS Matlab application
```

Fig. 3. DFS Algorithm: Application Results

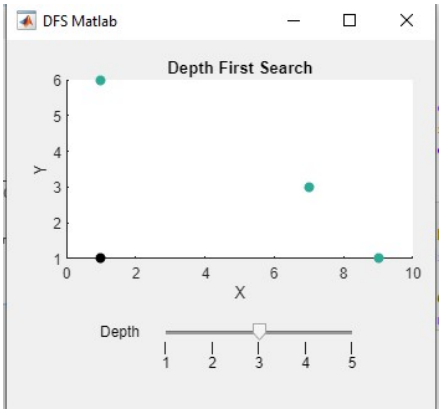

Fig. 4. DFS Matlab Graph

III. ML Clustering Algorithms

The resprentation of the KMeans algorithm

$$\mathcal{C}(n) = \left| \begin{array}{ccc} \mathcal{K} & \mathcal{X} & \mathcal{Y} \end{array} \right|$$

$$\mathcal{K}(n{<}\overline{\mathcal{C}_i}\frac{1}{2}) = \left| \begin{array}{cc} \mathcal{X} & \mathcal{Y} \end{array} \right|$$

$$\sum_{\mathcal{C}_{n-1}\neq\mathcal{C}}^{\infty}$$

$$\int_{\overline{\mathcal{C}}}^{i} \mathcal{C}_{ik} = \int_{\overline{\mathcal{K}}}^{j} \left(\sqrt{(\mathcal{K}_{jx}-\mathcal{C}_{ix})^2+\mathcal{K}_{jy}-\mathcal{C}_{iy})^2}\right) \leq \mathcal{K}_j$$

$$\int_{\overline{\mathcal{K}}}^{i} \left(\int_{\overline{\mathcal{C}}}^{j} \left(\mathcal{K}_{ij} = \left(\int_2 {}^{j} \overline{\mathcal{M}_k} = \frac{\sum \mathcal{C}_{j\equiv i}^{k}}{n} \right) \right) \mathcal{K}_{ik} = \left(\int_{\overline{\mathcal{M}}}^{j} \left(\int_2^k \mathcal{M}_{jk} \right) \right) \right)$$

KMeans, an unsupervised learning algorithm, works by sorting a given set of data points ($\mathcal{C}$) with a given size of K cluster points ($\mathcal{K}$) with a constraint of $||\mathcal{K}_i||$ is less than half of $||\mathcal{C}_i||$. A random Center of Gravity (CoG) to each K cluster is assigned. This technique enables the software to start the grouping procedure. Based on the Journal Article, 'Web User Clustering Analysis based on KMeans Algorithm', authored by JinHuaXu [5] states that 'These centroids should be placed in a cunning way because of different location causes different result. So, the better choice is to place them as much as possible far away from each other.' A way of fabricating this method into our implementation is to run a K Cluster through the randomise function; if a K Cluster $(\mathcal{X} \vee \mathcal{Y})$ CoG is the same, then randomise both $(\mathcal{X} \wedge \mathcal{Y})$ on the K Cluster points again.

After generating the $\mathcal{C}$ data set and $\mathcal{K}$ clusters, the next stage determines which data points belong to which cluster. To do this, it is a matter of finding the distance and assigning the $\mathcal{C}$ data set to the closest $\mathcal{K}$ cluster. The formula used to solve this problem, $||\mathcal{C}_i - \mathcal{K}_i||$. After this stage, gather each of the $\mathcal{C}$ that corresponds with $\mathcal{K}_i$ and calculate the mean. Set the mean from the $\mathcal{K}_i$ and move on to the next one. Repeat these said steps until there is no longer a change.

The aim of this experiment is to generate a \ number of data points and calculate \ number of clusters. To visualise each step of the KMeans experiment, a MATLAB application that enables users to control the depth provides this opportunity. Fig. 5 and Fig. 6 displays the two steps of $\mathcal{C}_5$ and $\mathcal{K}_3$ needed to group three clusters.

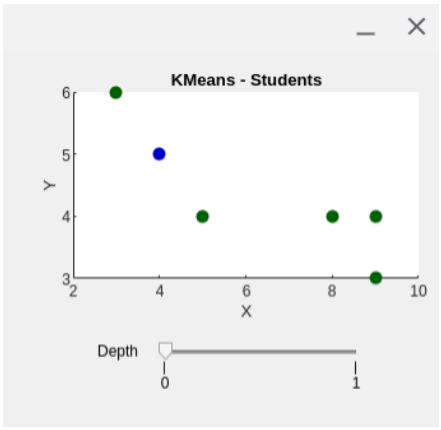

Fig. 5. KMeans Matlab Graph - 1

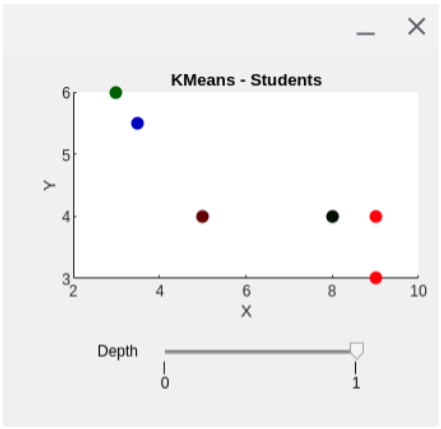

Fig. 6. KMeans Matlab Graph - 2

IV. Visualisation

After experimenting with Jupyter Notebook, RStudio and MATLAB, MATLAB offers tools to create shareable applications and visualisation tools. One of the ways to plot the data in steps within the MATLAB environment within the K-Means algorithm process was creating a C-library that communicates with existing code. This process enables users to interact with the MATLAB Application, changing data and watching the application plot.

Using the MathWorks - MATLAB Documentation, the library CLib successfully compiles communication files and load EntryPoint functions. However, this method had another problem that proved time-consuming that consists of moving a *string* over from the Clang to MATLAB. This was tried with, but not limited to, *char** and *char[]* datatypes, referring

to this manual https://uk.mathworks.com/help/matlab/matlab_external/passing-arguments-to-shared-library-functions.html.

Below is some evidencing of the test library, communicating to the *MTools* namespace, *Randomise* function that generates a number within a range, using an argument of a C Struct with the name of 'A'.

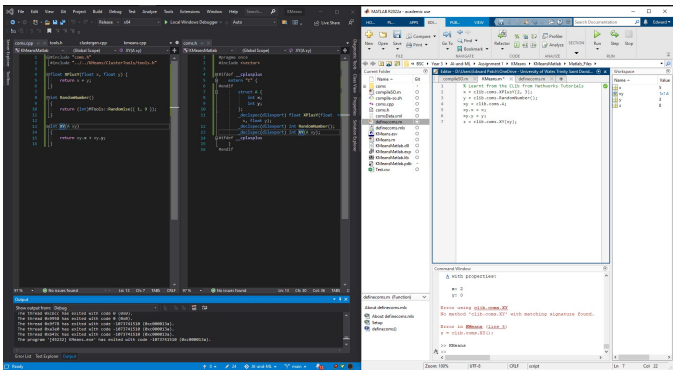

Fig. 7. C and MATLAB Working

A updated version of the code and with an attempt to send strings over from MATLAB to the C lang.

Header:

```
#pragma once

#ifdef __WIN32
  #define EXPORT _declspec(dllexport)
#else
  #define EXPORT
#endif

#ifdef __cplusplus
  extern "C" {
#endif
  struct CMemory {
    char name*;
    int value;
  };

  namespace CreateData {
    EXPORT CMemory CreateCMemory(char
        *name, int size);
  }

#ifdef __cplusplus
  }
#endif
```

Source:

```
#include "coms.h"

CMemory CreateData::CreateCMemory(char
    *name, int size)
{
   CMemory test({name, size});

   return test;
}
```

Due to this problem, another way of visualisation was to generate a Comma-Separated Values file to load data into MATLAB. The only difference this made is an added step of executing the DFS or KMeans software before executing the correspondent MATLAB application.

V. Conclusion

After conducting the Search and ML Clustering experiments with DFS and KMeans algorithms, it backs up that the two applications are straightforward to set up and are reusable for different objectives. The MATLAB applications demonstrate an interactive visual representation of the steps required to complete both tasks.

VI. Terminology

List of terminologies used in this document:-

- AI - Artificial Intelligence.
- BFS - Breadth First Search.
- DFS - Depth First Search.
- ML - Machine Learning.

VII. Reference List

[1] V. Palanisamy and S. Vijayanathan, "A novel agent based depth first search algorithm," in *2020 IEEE 5th International Conference on Computing Communication and Automation (ICCCA)*, pp. 443–448, ISSN: 2642-7354.

[2] ——, "Cluster based multi agent system for breadth first search," in *2020 20th International Conference on Advances in ICT for Emerging Regions (ICTer)*, pp. 54–58, ISSN: 2472-7598.

[3] M. Barbehenn, "A note on the complexity of dijkstra's algorithm for graphs with weighted vertices," vol. 47, no. 2, pp. 263–.

[4] C.-W. Tsai, K.-W. Huango, M.-C. Chiang, and C.-S. Yang, "A fast tree-based search algorithm for cluster search engine," in *2009 IEEE International Conference on Systems, Man and Cybernetics*, pp. 1603–1608, ISSN: 1062-922X.

[5] JinHuaXu and HongLiu, "Web user clustering analysis based on KMeans algorithm," in *2010 International Conference on Information, Networking and Automation (ICINA)*, vol. 2, pp. V2–6–V2–9, ISSN: 2162-5484.

[6] A. Boukhdhir, O. Lachiheb, and M. S. Gouider, "An improved mapReduce design of kmeans for clustering very large datasets," in *2015 IEEE/ACS 12th International Conference of Computer Systems and Applications (AICCSA)*, pp. 1–6, ISSN: 2161-5330.

[7] A. K. Mishra and P. C. Siddalingaswamy, "Analysis of tree based search techniques for solving 8-puzzle problem," in *2017 Innovations in Power and Advanced Computing Technologies (i-PACT)*, pp. 1–5.

[8] P. Jurkiewicz, E. Biernacka, J. Domżał, and R. Wójcik, "Empirical time complexity of generic dijkstra algorithm," in *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pp. 594–598, ISSN: 1573-0077.

[9] S. Jeyalatha and B. Vijayakumar, "Design and implementation of a web structure mining algorithm using breadth first search strategy for academic search application," in *2011 International Conference for Internet Technology and Secured Transactions*, pp. 648–654.

[10] X. Deng, Y. Yao, J. Chen, and Y. Lin, "Combining breadth-first with depth-first search algorithms for VLSI wire routing," in *2010 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE)*, vol. 6, pp. V6–482–V6–486, ISSN: 2154-7505.