Wikipedia Game Search Algorithm Solutions

Jadon Lai
Cal Poly
San Luis Obispo, United States
jlai43@calpoly.edu

Abstract—The Wikipedia Game [3] is a fun game where one "races" to get from one Wikipedia page to another. The start and end pages are given, but the catch is that you can only use links within the Wikipedia pages themselves. The goal is to try to reach the goal in the least number of steps and time possible. We use AI search algorithms such as breadth-first search (BFS), depth-first search (DFS), greedy best-first search (GBFS), and A* to try to solve the game, as well as compare their utilities to each other. The Wikipedia game changes the start and end goal daily, so solving the game using AI search algorithms will provide useful insight into how the Wikipedia pages are structured and how each algorithm works.

I. INTRODUCTION

A. Problem

The Wikipedia Game can often take a long time to solve since there are numerous links on every page, which one can easily get lost in. Because of this, we will try to solve the game in the least amount of time and steps possible. For this report, this is done using several AI search algorithms. These algorithms will be compared so that we can further understand their comparative efficiency and how they work. We also want to try to provide an answer to the idea of "Six Degrees of Freedom," [1] to see if most, if not all Wikipedia pages are connected by 6 steps.

If you've ever heard of the phrase "it's a small world," then you'll already have a concept of the Six Degrees of Freedom. The original idea is that everybody in the world is connected by at most six steps, but we've adapted it to the Wikipedia website. Aside from the development of our AI search algorithms, we wish to find if any concept in the world is connected by just 6 steps, or more literally, if we can get from one Wikipedia page to another by 6 steps when only looking at the links on each site.

B. Importance

This research provides useful value due to its insight into how Wikipedia, one of the biggest sites for information, is structured. We'll also be able to expand on the fundamentals of AI search algorithms and apply them to a complex system containing a nearly infinite tree of links, that contains loops. Thus, our solution will effectively demonstrate how the algorithms perform on an extreme edge case (the graph of links created by Wikipedia pages) and display the effectiveness of each one.

C. Impact

Our solution to this problem is not limited to the impact it has on the Wikipedia Game. These solutions can be applied to any other game or problem searching for specific points between large graph systems. We hope our solution will show how basic AI search algorithms can perform in a worst-case scenario for any game when implemented in its simplest, most naïve form.

D. Contribution

Our solution will contribute not only a solution to the Wikipedia Game but also a solid base of algorithms for others to build upon for other games and related problems. At its core, this solution is a useful tool for people to use to compare their solutions to ours. This way they'll not only be able to see how to operate these algorithms, but how certain algorithms reach their solution along with the time they take. For our algorithms, we define distance as the path from the start page to the end page and list the time taken to find our solution when needed.

II. RELATED WORK

A. Six Degrees of Wikipedia

A solution to the Wikipedia Game has already been created, by the software engineer Jacob Wenger [2]. His solution finds the shortest path from the start and end, including every possible variation of that distance. He then lists the number of optimal solutions, the degrees of separation, and the time it takes. He's linked his code [4] if you wish to look more into his solution, however, his implementation doesn't appear to use AI search algorithms.

He gets his data from Wikimedia and stores it inside a database containing 3 tables of pages, page links, and redirects. This allows him to quickly look at a page and its corresponding links, and then get from the start page to the end page from there. Thus, it's a type of informed search that isn't AI, since he already has all the pages and what each page can point to. This implementation of a database is quick, efficient, and optimal, but is different from our purpose of analyzing AI search algorithms.

B. Our Research

The idea of using AI search algorithms isn't a new one, but we wish to explore how such algorithms perform to solve the Wikipedia Game due to the complexity and sheer size of the data. This will allow us to see how the AI works in a real-life scenario. Not only this, but we'll also be able to compare the different search methods by their distance and the time it takes to find the end page, using a naïve solution. This will help the public be able to look at search algorithms in real-time, in a worst-case scenario, and this allows our research to not be limited by the Wikipedia Game, but to be expandable to other games and concepts that might be like our research.

The reason that our search algorithms will most likely be much slower than Wenger's is that ours is uninformed, and we don't know what future pages will point to. This is part of the uniqueness of our solution, which is that we will be able to use our algorithm anytime, with no previous knowledge of the Wikipedia site.

III. METHODS

A. Wikipedia Game

The Wikipedia Game is a "speedrun" where a player tries to go from a starting Wikipedia page to a destination Wikipedia page, as quickly as possible. This can either be in the shortest distance possible, the shortest time, or both. Our goal is to try to optimize both measurements, but ideally, our AI search algorithms will find the optimal path every time.

B. Wikipedia API

We use the Wikipedia API [5] to access the links for each page. There are multiple Wikimedia Wiki endpoints, however, we use the English Wikipedia API for our purposes. This can be done by sending a query to the endpoint that we want: `https://en.wikipedia.org/w/api.php`. Then, the query returns a list of dictionaries, which has the following format:

```
{'pages':
 {'22822937':
  {'pageid':22822937, 'ns':0, 'title':'Miss Meyers',
   'links':[{'ns':0, 'title':'Speed index'},
            {'ns':0, 'title':'Stakes race'},
            {'ns':0, 'title':'Stallion'},
            {'ns':0, 'title':'Thoroughbred'},
            {'ns':0, 'title':'Three Bars'}]}}}
```

We're easily able to get the titles from the query in alphabetical order and effectively search through them in the future.

C. Heuristics

Because our searches are uninformed searches when using algorithms like GBFS or A*, our heuristics, h, and costs, g, need to be estimated. The GBFS uses only h to calculate the total cost, f, in Equation 1, but A* uses both h and g to calculate f in Equation 2.

$$f = h \tag{1}$$

$$f = g + h \tag{2}$$

There are multiple ways of doing this, but for our purposes, we settled on using the cosine similarities between words. One useful library for this is the Sentence Transformers library [6], which transforms a list of given words into embeddings. These embeddings are "dense vector representations" in a vector space so that we can compare the vectors to see which words are close. These comparisons include but are not limited to, semantic search, clustering, and cosine similarities.

The model that we use to transform the words into vectors is the all-MiniLM-L6-v2 model [7]. This is a pre-trained model that was trained on all available training data, with more than 1 billion training pairs [9]. The reason that we use this model instead of different ones is because it's a fast model (14,200 sentences per second), that still has relatively good quality (arbitrary value of 58.8). The quality score is given based on the model's performance on sentences and semantic search. An alternative to this model would be the all-mpnet-base-v2 model which has a better average performance (63.3), but a slower speed (2,800 sentences per second). However, this is up to the developer, if they decide to use the Sentence Transformers library, and they're able to choose any model, including the ones in Table 1.

TABLE I
Sentence-Transformers Pretrained Models

Model Name	Avg. Performance	Speed	Model Size (MB)
all-mpnet-base-v2	63.30	2800	420
multi-qa-mpnet-base-dot-v1	62.18	2800	420
all-distilroberta-v1	59.84	4000	290
all-miniLM-L12-v2	59.76	7500	120
multi-qa-distilbert-cos-v1	59.41	4000	250
all-MiniLM-L6-v2	58.80	14200	80
multi-qa-MiniLM-L6-cos-v1	58.08	14200	80
paraphrase-multilingual-mpnet-base-v2	53.75	2500	970
paraphrase-albert-small-v2	52.25	5000	43
paraphrase-multilingual-MiniLM-L12-v2	51.72	7500	420
paraphrase-MiniLM-L3-v2	50.74	19000	61
distiluse-base-multilingual-cased-v1	45.59	4000	480
distiluse-base-multilingual-cased-v2	43.77	4000	480

For our purpose, we get two embeddings, one is a list of just the destination page (end), and the other is a list of each of the links on the page. This allows us to get the cosine similarities between the two vectors, giving us a list of each link and its similarity to the end.

A cosine similarity is a distance measurement, like the Euclidean or Manhattan distance. The reason that we're using the cosine similarity instead of the other two distance measurements is because the cosine similarity disregards the size of the vector [8]. This is useful because we want to compare how similar a link is to the end, irrespective of how long the link title is. This algorithm is given in Equation 3 and measures the cosine angle between two vectors (the link and the end), disregarding the length of the vector [10].

$$similarity(A, B) = \cos(\theta) = \frac{A \cdot B}{||A||||B||} \quad (3)$$

We are then able to use this cosine similarity as *h* from a link to the end. This will help us both in our GBFS and A* search algorithms to choose the next node to visit.

D. Costs

The cost, *g*, is a bit more complicated to measure than *h*. As mentioned before, it's used in the A* algorithm to calculate *f* of a child node, but it's not used in GBFS. The cost, *g*, is meant to represent the cost it's already taken to get to the current node. Thus, we represent the cost of each node using Equation 4. However, this has the potential to overestimate the cost of the start to the current node, making the path of A* not optimal. It would be beneficial to try to research other alternatives in the future to see what a good cost measurement would be.

$$current(g) = parent(f) \quad (4)$$

This way the cost, *g*, of the current node is the total cost, *f*, that it takes to get there. Then we can simply use Equation 2 to calculate *f* for the current node.

E. Breadth-First Search

Breadth First Search (BFS) starts from a root node and examines all the nodes in the next level before moving on to the next level [11]. We can do this using two queues, one for the nodes we want to visit, and one for the nodes we've already visited. This allows us to ensure that we don't end up in an infinite loop if the graph contains a cycle. Then, when we reach the goal, we can examine the tree that we've created to obtain our solution.

BFS is optimal and complete, however, it has a higher time and space complexity than other algorithms. The complexities can be represented as a time complexity of O(V+E), given that V is the number of vertices and E is the number of edges, and the space complexity is O(V). This is in a worst-case scenario where the algorithm examines every node in the tree.

We don't expect this algorithm to perform very well, as each Wikipedia page is going to have hundreds of links, so it'll likely take a long time to even get 2 levels into the search.

F. Depth First Search

Depth First Search (DFS) also starts from the root node; however, it doesn't examine all the nodes in the level. Instead, it chooses the first child and goes down that entire branch, only backtracking if we reach a leaf node [12]. The same two queues as BFS are also initialized so that we avoid loops, and it has the same space and time complexity in a worst-case scenario. However, DFS is neither optimal nor complete.

One thing to note is that if the tree is wider than it is long in an average case, then BFS will have a worse space complexity than DFS, and vice versa. However, both time complexities are the same, and in a worst-case scenario, both time and space complexities are the same.

We also don't expect this algorithm to be effective due to the sheer number of pages within Wikipedia. By searching the first page that appears every time, we'll search through most, if not all the webpages that start with the letter *a,* and if that's not our solution doesn't start with an *a*, then we most likely won't find it in a reasonable amount of time.

G. Greedy Best First Search

Greedy Best First Search (GBFS) "greedily" searches the "best" node that it can find [13]. This is done by using the heuristic mentioned in Part C and putting the nodes in a priority queue sorted by that heuristic from the lowest to greatest. There is also a visited set to ensure that we don't run into issues with loops.

We store each parent node and its child nodes as a key-value pair in a dictionary so that when we finish, we can backtrack to find the solution. It's not optimal or complete, but it does tend to have a low time and space complexity, being fast and efficient.

This algorithm should perform better BFS and DFS since it doesn't search naively, but rather it searches through the best options. However, because it's not optimal, we don't expect it to have the best distance, although it may be "good enough." It should reach the solution very quickly though.

H. A*

A* is like GBFS, but its f is different. It doesn't simply take the h, but instead considers the cost it takes to get to the current node from the very start [14]. Thus, we use Equation 2 to calculate f and this causes the A* search algorithm to be complete and optimal as long as the heuristic is admissible (it never overestimates the distance). However, our heuristic is an estimate between the end and the link, so we can't tell for certain if our heuristic is an underestimate of the distance between the pages.

We would expect that A* would perform better than GBFS since it has a better calculated total cost, but as we'll see in the results, this isn't the case.

IV. EVALUATION / RESULTS

A full directory of our results and source code can be found at [15].

A. Criteria

As mentioned in Section 1, we measure the algorithms based on the distance and the time it takes to complete. The distance is simply the number of steps that it takes to get from the start to the end.

The time is measured using the "time" library. We get the current time using the command `time.time()` and save it into a variable `start_time` at the start of the algorithm. Then, when we find a solution, we return the `time.time() − start_time`, which is the time that the algorithm has been running.

B. Scenarios

Each scenario is simple, we take the current daily Wikipedia Game start and end point (we'll refer to this as a *sample*), then run our algorithms on it. Our samples were run on a 2020 Macbook Air, with an M1 chip. For sanity's sake, the max time is set to 900 seconds (15 minutes) and the max nodes are set to 200. We can also put in our input if we wish to examine certain test cases. We used our own input to test specific examples such as a target page already being on the start page or checking our algorithms performances when given short or long start and end points. Each scenario looks like this:

```
Usage: python3 wikiSearch start end algorithm [verbosity]

Output: (path_list, distance, time)
```

By setting the verbosity to 1, we output the link name at each step. If the verbosity is 2, then we additionally print out the current node's tree level, distance, and elapsed time. BFS includes everything mentioned above, in addition to the queue size. GBFS and A* also include everything mentioned above, but the queue is a priority queue, and the heuristic / cost + heuristic is also included. These details help our analysis and look at how each algorithm performs, and what they struggle on.

Additionally, we create a graph of cosine similarities vs. time for each scenario (during GBFS and A* if verbosity = 2), so that we can see how the algorithm is performing and when. However, it's worth noting that our cosine similarity heuristic is still an estimate that is not the exact distance between the current page and the end.

Regardless, a .csv file is created with every step and its details, as well as including if the algorithm failed due to exceeding the time or node limit, or if it succeeded, then the path list. Both summaries are added at the end of the .csv file.

Most of our results were derived from the daily Wikipedia Game, however we did use one example that proved that A* is complete and optimal, while GBFS can "miss" the end and get lost down a wrong branch. This sample is "Conseil supérieur de l'audiovisuel (Belgium)" to "HADOPI law."

C. BFS and DFS

We ran our algorithms for many days, starting from 3/1/24. That day's Wikipedia Game was going from the Wikipedia page "Rudolf Nureyev" to "Addition." When using Wenger's 6 Degrees of Wikipedia, we can see that there are 6 optimal paths for this sample, that have a length of 3, and they were found in 1.14 seconds.

Both BFS and DFS proved to be unfruitful as they both exceeded the max time limit.

TABLE 2
First and Last 5 Nodes of BFS for "Rudolf Nureyev" to "Addition"

Node	Level	Queue Size	Distance	Elapsed Time
Rudolf Nureyev	0	0	0	0.000
AIDS	1	244	1	4.448
Adolphe Adam	1	244	2	4.763
Afternoon of a Faun (Nijinsky)	1	757	3	13.676
Alexander Ivanovich Pushkin	1	1212	4	21.058
…	…	…	…	…
Royal Academy of Dance	1	32280	178	870.265
Royal Danish Ballet	1	32405	179	875.495
Rudolph Valentino	1	32440	180	879.684
Russian gay propaganda law	1	32641	181	884.843
Russian language	1	32856	182	890.702

TABLE 3
First and Last 5 Nodes of DFS for "Rudolf Nureyev" to "Addition"

Node	Level	Elapsed Time
Rudolf Nureyev	0	0.000
AIDS	1	4.291
HIV/AIDS	2	4.593
ABC News	3	16.625
20/20 (American TV program)	4	28.872
…	…	…
101st Tank Division (Soviet Union)	196	892.930
102nd Anti-Aircraft Artillery Division (Soviet Union)	196	893.082
102nd Cavalry Division (Soviet Union)	196	893.230
102nd Guards Rifle Division	196	893.385
65th Rifle Division	197	893.533

It's easily visible to see that in Table 2, BFS never got past the first page of links, and is purely searching through the links in alphabetical order. This makes sense because the list of links is returned from the query in alphabetical order. It's also worthwhile to note that we can also see that BFS does take up a large amount of memory, reaching a queue size of 32,856.

DFS on the other hand, doesn't take up as much memory but easily got lost down the first branch it saw, shown by Table 3. It takes the very first link on the page and travels down to that branch, which happens to eventually reach Wikipedia pages that start with numbers. This is because numbers come before letters in the list of links.

We saw the same results in both BFS and DFS for 3/2/24 and stopped trying those AI search algorithms starting on 3/4/24, as it's obvious what the patterns are. Keeping our max time limit and max nodes the same, they will most likely always run into a max limit exceeded error. This makes these algorithms unfeasible, as it would take an unreasonable amount of time to naively search through almost every Wikipedia page before we can reach the end.

D. GBFS

For 3/1/24, GBFS expanded 26 nodes in 163.717 seconds (about 2 and a half minutes) and found a path of 15 links:

```
['Rudolf Nureyev', 'Kremlin', 'Red Square', 'Gilding', 'Gypsum', 'Lead', 'Coin', 'Monetary', 'Money', 'Money multiplier', 'Cash', 'Accrual', 'Interest', 'Sumer', 'Arithmetic', 'Addition']
```

TABLE 4
First and Last 5 Nodes of GBFS for "Rudolf Nureyev" to "Addition"

Node	Level	Total Cost	PQueue Size	Distance	Elapsed Time
Rudolf Nureyev	0	0	0	0	0.000
AIDS	1	0.715	244	1	8.827
Kremlin	1	0.745	244	2	10.293
Metochion	2	0.726	535	3	17.152
Red Square	2	0.737	559	4	18.765
…	…	…	…	…	…
Accrual	9	0.643	4367	22	112.433
Interest	10	0.627	4465	23	115.445
Sumer	11	0.561	4876	24	125.535
Arithmetic	12	0.442	6509	25	156.427
Addition	13	0.000	6869	26	163.717

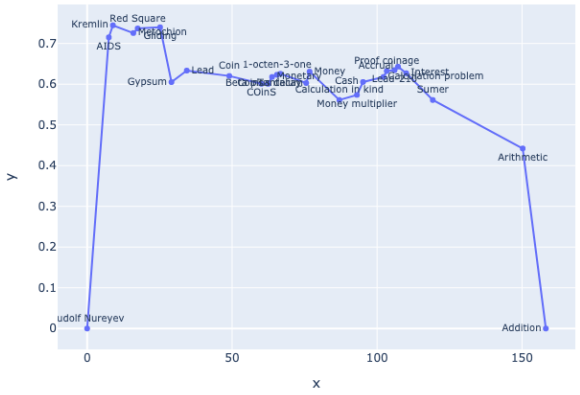

Fig. 1. Cosine Similarity vs. Time (s) for GBFS of "Rudolf Nureyev" to "Addition"

The priority queue size remains manageable and in the last 5 links, we can see that it reaches pages very similar to the word

"addition" in Table 4. We can also see that there's a downward trend in Figure 1, which means the algorithm is also getting closer over time due to its heuristic, and not moving further away. However, the one downside that we expected is that the path distance is very long.

A similar trend can be seen in the rest of our samples, where GBFS tends to finish, and faster than A*, but will almost never find the optimal path. This is not ideal, as our main goal is to try to find the optimal path in the least amount of time possible, however the time GBFS takes to find its path is very reasonable, compared to BFS, DFS, and A*. Even though we wish to find the optimal path, because BFS, DFS, and A* take so long, depending on how much time you're willing to wait and if it's acceptable to find a path and not the optimal path, GBFS is the best option of the 4.

*E. A**

When we tried using A* on 3/1/24, it exceeded the time limit of 15 minutes.

TABLE 5
First and Last 5 Nodes of A* for "Rudolf Nureyev" to "Addition"

Node	Level	Total Cost	PQueue Size	Distance	Elapsed Time
Rudolf Nureyev	0	0	0	0	0.000
AIDS	1	0.715	244	1	7.618
Kremlin	1	0.745	244	2	8.980
Dacha	1	0.756	535	3	15.466
Bashkortostan	1	0.758	661	4	19.283
…	…	…	…	…	…
André Malraux	1	0.859	33924	102	846.263
Karen Kain	1	0.859	34325	103	855.147
Ninette de Valois	1	0.860	34386	104	857.908
The Independent	1	0.861	34566	105	862.672
Mikhail Gorbachev	1	0.861	34935	106	870.650

When analyzing Table 5, it appears that A* seems to be very similar to BFS. However, this isn't the case, as A* is searching through the nodes by their total cost. The reason that A* appears to be "stuck" on the first page is because there are so many links (244) that our max time limit of 15 minutes is reached before A* can expand each one. A*'s total cost causes it to search through the most similar nodes in each level before moving onto the next level, however it still takes time to search through up to 244 nodes. This is necessary because otherwise, A* might miss the optimal path if it travels through a different node in the current level.

Thus, A* isn't really feasible within a 15-minute time frame, although it can be very good with more time. This is shown in Part F, where A* can perform well.

*F. GBFS vs. A**

Starting on 3/5/24, we extended the max time limit to 36,000 seconds (10 hours) and the max nodes to 10,000 to see if A* does in fact perform more optimally than GBFS. As shown in Tables 6 and 7, A* does manage to either perform equally as well or outperform GBFS in terms of distance.

TABLE 6
First and Last 5 Nodes of GBFS for "Chemical Engineering" to "Drake (Musician)"

Node	Level	Total Cost	PQueue Size	Distance	Elapsed Time
Chemical Engineering	0	0	0	0	0.000
Chemical engineering	1	0.906	0	1	2.206
George E. Davis	2	0.737	397	2	10.803
George Davis (disambiguation)	3	0.741	416	3	12.561
George Davis	4	0.697	416	4	14.030
…	…	…	…	…	…
Dan Minogue	21	0.477	9989	89	307.167
Dannii Minogue	22	0.499	10314	90	314.049
Kylie Minogue	23	0.462	10839	91	324.992
Prince (musician)	24	0.426	11784	92	343.651
Drake (musician)	25	0.000	13330	93	373.339

TABLE 7
First and Last 5 Nodes of A* for "Chemical Engineering" to "Drake (Musician)"

Node	Level	Total Cost	PQueue Size	Distance	Elapsed Time
Chemical Engineering	0	0	0	0	0.000
Chemical engineering	1	0.906	0	1	3.372
George E. Davis	2	1.643	397	2	21.274
John Wiley & Sons	2	1.660	416	3	24.127
Audio engineer	2	1.661	416	4	26.282
…	…	…	…	…	…
Songwriter	3	2.118	108553	402	5326.293
Nick Davis (music producer)	3	2.146	108834	403	5339.090
Mike Stone (record producer)	3	2.171	108834	404	5340.594
Global Recording Artist of the Year	3	2.192	108894	405	5344.930
Drake (musician)	4	2.192	109300	406	5362.151

On 3/5/24, GBFS found a distance of 25, while A* found a distance of 4. A*'s path consisted of:

```
['Chemical Engineering', 'Chemical engineering',
'Audio engineer', 'Global Recording Artist of the
Year', 'Drake (musician)']
```

This is the optimal path, however (the optimal path has a distance of 3) due to a capitalization error within the links 'Chemical Engineering' and 'Chemical engineering.' A* does indeed perform much better than GBFS, with 22 less distance in its path, at the cost of a bigger queue (95,970 elements larger) and a longer time that takes an extra 4,988.812 seconds.

However, we can't tell whether our A* will always find the optimal path because our heuristic is estimated. This would result in a situation where A* finds a very good path, but it's not the optimal one. However, A* is still much better at finding the optimal (or in our case, near-optimal) path than GBFS.

On the other hand, will almost always take much longer, as mentioned before. This is shown by Figures 2 and 3, during 3/6/24's Wikipedia Game. This sample was from "Red algae" to "Wolfgang Amadeus Mozart."

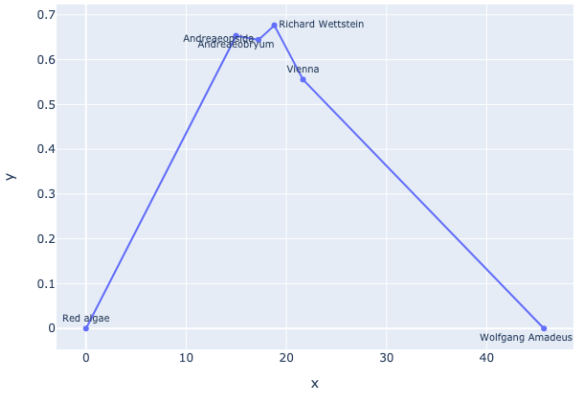

Fig. 2. Cosine Similarity vs. Time (s) for GBFS of "Red algae" to "Wolfgang Amadeus Mozart"

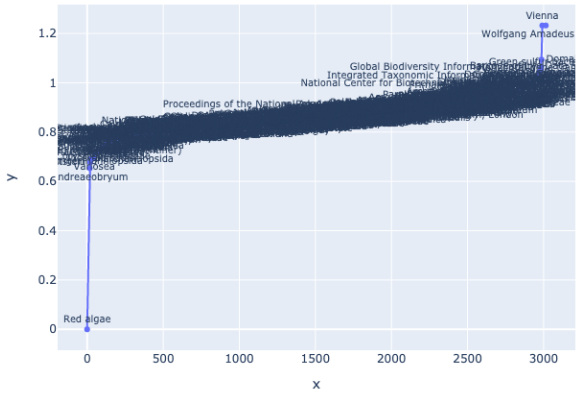

Fig. 3. Cosine Similarity vs. Time (s) for A* of "Red algae" to "Wolfgang Amadeus Mozart"

We can see that if the optimal path is simple (in terms of the cosine similarities between words for each link), then GBFS will take a lot less time and space to find the optimal path. In this case, the optimal path had a distance of 3, and GBFS only took 45.707 seconds to find it, while A* took 3,014.728 seconds, although it was guaranteed to find a near-optimal path, while GBFS wasn't. However, the time it took A* to find the path was long (almost an hour), but that's the tradeoff between guaranteeing a near-optimal path.

GBFS's downside in it not being complete or optimal can be seen from a sample that we created. This is from "Conseil supérieur de l'audiovisuel (Belgium)" to "HADOPI law" and A* found the optimal path with a distance of 2 in 168.726 seconds (Figure 4). On the other hand, GBFS exceeded the normal max time limit of 15 minutes, since it went down the wrong path and couldn't recover because the page titles it found were similar in cosine similarity, but not similar in context.

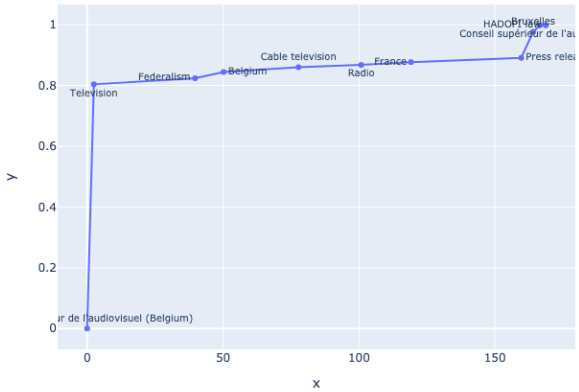

Fig. 4. Cosine Similarity vs. Time (s) for A* of "Conseil supérieur de l'audiovisuel (Belgium)" to "HADOPI law"

G. Analysis and Special Cases

We can see that GBFS performs with the best elapsed time, while A* performs with the best path distance, but it's important to see why.

Due to our heuristic being a cosine similarity, if GBFS finds a similar node that isn't similar in context (e.g. Law of Japan and HADOPI Law), then it'll keep searching the similar nodes without ever actually finding the end. A* fixes this by adding the cost it takes to get to the current node from the start, so it doesn't go down an almost infinite wrong path, which is why it works so well in our created example.

Both algorithms suffer from some edge cases, however, such as names, acronyms, and numbers. This is again due to the cosine similarity not adjusting for the context of words, as can be seen in the 3/6/24 Wikipedia Game, trying to get to "Drake (Musician)." The overall performance of the algorithms is about what we expect, with BFS and DFS performing poorly due to their naïve way of searching, GBFS having the best elapsed time, and A* having the best path distance.

Additionally, the reason that some of the steps take so long is that every step involves us querying the Wikipedia API. This is a slow process, and a solution would be like Wenger's, where we query the entire database in advance, and then perform our search algorithms on the copied database. Without this solution, computing power would be a determining factor in how fast the algorithms run.

V. CONCLUSIONS

A. Algorithms

Our algorithms perform as well as expected, with BFS and DFS being infeasible on such a large tree of links, while GBFS performs well with time and A* performs well with path distance. These algorithms could be improved by coming up with a better heuristic than the cosine similarity of word embeddings, however, we're still able to see the effects of a worst-case scenario with the Wikipedia site on AI search algorithms.

We were prepared for BFS and DFS to perform badly, but we didn't expect A* to take an exorbitant amount of time. This means that if a tree has hundreds of child nodes for every node, then A* will take a while to perform.

Additionally, the estimated heuristic plays a big factor in whether the algorithms are efficient and optimal. Because we don't know the actual distance from the current Wikipedia page to the end, an estimate is necessary, unless we already have a database of all the Wikipedia links. This, in conjunction with our cost actually being an overestimation can eventually end up in our A* algorithm not being optimal.

B. Six Degrees of Freedom

From our samples, we can conclude that Wikipedia pages are mostly connected by 6 steps. This doesn't necessarily mean that every topic known to humans is connected by 6 steps, but it's a good indication that they very well could be. However, we do have a very small sample size and it would be beneficial to try to gather more samples.

C. Application

For any developer wishing to use these AI search algorithms, if they're looking into GBFS and A*, then a measured or very well-estimated heuristic is a must. If the heuristic isn't perfect or near perfect, then there are plenty of ways that the AI can stray from the end goal.

Additionally, if the steps take a large amount of time, then it's worthwhile to use GBFS instead of A*, even though A* is complete and optimal. GBFS will find a possible solution in much less time than A* will find one, but if completeness and optimality are a requirement, then it should be expected that the algorithm can take hours to complete. This is especially true if the game requires an AI search algorithm to compute a result in less than a second to ensure that the user experience is satisfactory. For example, if we click a button in a game, we wouldn't want to wait an hour for an NPC to come up with a response.

Most searches will most likely not face the scenarios that the Wikipedia Game creates, we have created some good numerical values and analysis to be able to reference for searching with AI algorithms in a worst-case scenario.

VI. Future Work

A. Search Algorithms

Our research was limited to a few standard AI search algorithms. To extend upon this, future explorations of search algorithms can involve others that we did not use, like Bi-Directional search. This extension can also investigate the performance of other methods of traversing the graph of values in Wikipedia, like reinforcement learning algorithms, as seen in a blog from OpenAI [16]. These algorithms can look at reinforcement learning as a method to traverse the graph so that the objective can be reached via a reward-based system. Using

this paper as a basis, future work can then compare the effectiveness of searching with reinforcement learning models to show what the most feasible, near-optimal paths would be.

B. Changing our rules of the game

Currently, as outlined above, our algorithms simply try to find a path to the end of the game. Traditionally, the rules of this game are to get to given pages in as few clicks, and in as short of a time as possible. This is currently not how our algorithms operate, as they pass through pages to scan for their cost and heuristic based on the algorithm. An algorithm that can find optimal links on pages without having to look through other pages would be a step up from our work here. Future work could apply their algorithm compared to what we have found here to compare how such a system would differ from traditional search algorithms.

REFERENCES

[1] G. Morse, "The Science Behind Six Degrees," *Harvard Business Review*, Feb. 2003. https://hbr.org/2003/02/the-science-behind-six-degrees

[2] J. Wenger, "Six Degrees of Wikipedia," *www.sixdegreesofwikipedia.com*. https://www.sixdegreesofwikipedia.com/?source=Deodorant&target=Parachute# (accessed Mar. 04, 2024).

[3] "The Wikipedia Game," *www.thewikipediagame.com*. https://www.thewikipediagame.com/

[4] J. Wenger, "jwngr/sdow," *GitHub*, Mar. 04, 2024. https://github.com/jwngr/sdow (accessed Mar. 04, 2024).

[5] "API:Main page - MediaWiki," *www.mediawiki.org*. https://www.mediawiki.org/wiki/API:Main_page

[6] "Using Sentence Transformers at Hugging Face," *huggingface.co*. https://huggingface.co/docs/hub/en/sentence-transformers (accessed Mar. 05, 2024).

[7] "sentence-transformers/all-MiniLM-L6-v2 · Hugging Face," *huggingface.co*, Jan. 18, 2024. https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2#:~:text=This%20is%20a%20sentence%2Dtransformers (accessed Mar. 05, 2024).

[8] "Cosine Similarity," *GeeksforGeeks*, Oct. 02, 2020. https://www.geeksforgeeks.org/cosine-similarity/

[9] "Pretrained Models — Sentence-Transformers documentation," *www.sbert.net*. https://www.sbert.net/docs/pretrained_models.html

[10] F. Karabiber, "Cosine Similaritity," *www.learndatasci.com*. https://www.learndatasci.com/glossary/cosine-similarity/

[11] GeeksforGeeks, "Breadth First Search or BFS for a Graph - GeeksforGeeks," *GeeksforGeeks*, Feb. 04, 2019. https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/

[12] GeeksforGeeks, "Depth First Search or DFS for a Graph - GeeksforGeeks," *GeeksforGeeks*, Feb. 04, 2019. https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/

[13] "Greedy Best first search algorithm," *GeeksforGeeks*, Dec. 15, 2022. https://www.geeksforgeeks.org/greedy-best-first-search-algorithm/

[14] R. Belwariar, "A* Search Algorithm - GeeksforGeeks," *GeeksforGeeks*, Sep. 07, 2018. https://www.geeksforgeeks.org/a-search-algorithm

[15] https://drive.google.com/drive/folders/1D5vZNjP2xqIUgZjwJ14lGi6ldy_ZhjW9?usp=sharing

[16] "Algorithms — Spinning Up documentation," *spinningup.openai.com*. https://spinningup.openai.com/en/latest/user/algorithms.html