A close up of a logo

Description automatically generated

Analysing Properties of a Wikipedia Page Graph

By Clayton Herbst (22245091)

Graphs are data structures that consist of a finite set of vertices and a set of ordered or unordered edges that form the link between these vertices (Wikipedia 2019). A single Wikipedia page (the node) can be considered as a member of a set of Wikipedia pages (the graph) interconnected through hyperlinked URL’s (the directed edges) from which various characteristics and properties can be determined.

## Shortest Path Solution

In order to find the minimum number of URL links a user must follow to navigate from one Wikipedia page to another, a Breadth First Search (BFS) approach can be implemented starting at the specified starting vertex and ending at the destination vertex. The BFS algorithm starts at a vertex and construct a spanning tree for the given graph called the breadth-first tree (CITS2200 Topic 12) using a first-in-first-out queue abstract data structure. Modifying this common graph search technique allows the algorithm to keep track of the distance of each node within the breadth-first spanning tree from its root. Considering a Wikipedia graph, G, the pseudo code to find the shortest path between vertex *v* and vertex *y* is as follows:

*Procedure BFS(v, y) //v is the starting node, y is the destination node*

*Push v on to the tail of Q*

*dist[v] 🡨 0*

*while Q is not empty*

*Pop vertex w from the head of Q for each vertex x adjacent to w do*

*if colour[x] is white then dist[x] ← dist[w]+1*

*if x equals y then return dist[x]*

*end if*

*colour[x] ← grey*

*Push x on to the tail of Q*

*end if*

*end for*

*colour[w] ← black*

*end while*

The time complexity of this shortest path algorithm implementation is O(V + E) in the worst case where V is the number of vertices in the graph and E is the number of edges in the graph. This has been derived by summing the time complexity of enqueuing all vertices and examining all edges which are O(V) and O(E) respectively (CITS2200 Topic 12). The BFS provides optimal time complexity across an unweighted graph (Wikipedia 2019).

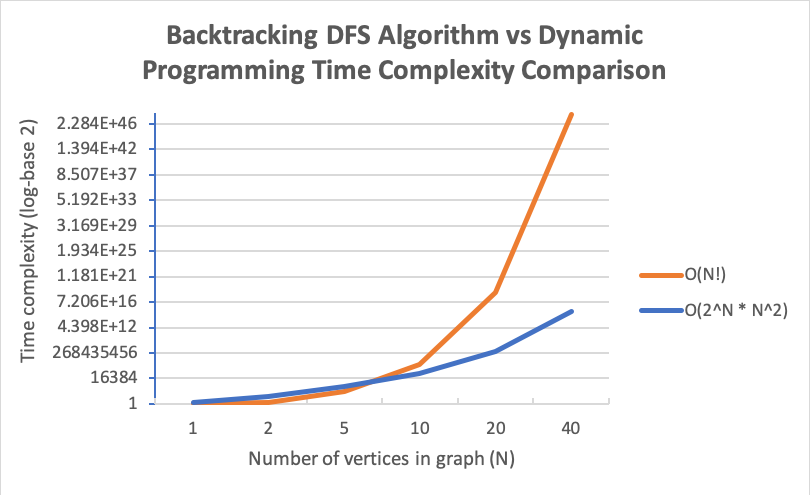
## Hamiltonian Path

Hamiltonian paths are paths within a graph that visits each vertex exactly once (Wikipedia 2019). The naive solution to find Hamiltonian paths within a graph is to explore all possible configurations or permutations of the set of vertices within the Wikipedia graph. As a result, there will be *n!* possible configurations where n is equal to the number of vertices in the graph. The naive solution hence has a time complexity of O(n!). One such implementation is the Backtracking Algorithm which performs a DFS starting at every vertex contained within the graphs’ vertex set, pushing all nodes visited on to the stack if at least one adjacent vertex is set as ‘unvisited’ (HackerEarth n.d). If the number of elements contained within the stack at any point is equivalent to the number of elements set as ‘visited’, a Hamiltonian path exists. This solution is extremely inefficient due to the repetitive computation of known results.

Dynamic programming addresses this computational problem by keeping records that map previously computed results (Zaveri M 2019) and thus optimising computational time. My Hamiltonian path algorithm uses an adapted implementation of the Bellman-Held-Karp algorithm by implementing their assumption that *“every sub-path of a path of minimum distance is itself of minimum distance”* (Wikipedia 2019*)* using tabulation to memorise previously computed results. This is achieved by using a loop to set particular bits in a bit-set data structure in order using a *bottom-up* approach (Zaveri M 2019). The *bottom-up* approach starts with the smallest sub-problem, computing a solution for these problems and building on these solutions until the larger problem is solved (Joshi V 2017). Hence, the *bottom-up* approach directly implements the underlying assumption of the Bellman-Held-Karp algorithm. The resulting worst-case time complexity of this implementation is enumerating though all possible sets of function calls (Joshi V 2017) comprising of each bit-set mask of size and each possible configured directed edge of time complexity . Finally, in the worst case, if a Hamiltonian path does exist, the ‘Hamiltonian walk’ is constructed by backtracking from the destination node to the starting node consuming order-n time (). The resulting solution is hence an optimised solution of the naive backtracking method.

**Figure 1**

Time complexity comparison to established whether a Hamiltonian path exists



## Strongly Connected Components

## Graph Centres

## References

Graph definition: <https://en.wikipedia.org/wiki/Graph_(abstract_data_type)>

Shortest Path: <https://en.wikipedia.org/wiki/Shortest_path_problem#Unweighted_graphs>

Hamiltonian Path: <https://en.wikipedia.org/wiki/Hamiltonian_path>

Hamiltonian Path: <https://www.hackerearth.com/practice/algorithms/graphs/hamiltonian-path/tutorial/>

Dynamic Programming: <https://medium.com/free-code-camp/an-intro-to-algorithms-dynamic-programming-dd00873362bb>

Keld-Karp Bottom-up Approach: <https://medium.com/basecs/speeding-up-the-traveling-salesman-using-dynamic-programming-b76d7552e8dd>