Graph Learning with a Nearest Neighbor Approach*

Sven Koenig and Yury Smirnov
School of Computer Science, Carnegie Mellon University
Pittsburgh, PA 15213-3891, USA
skoenig@cs.cmu.edu, smir@cs.cmu.edu

Abstract

In this paper, we study how to traverse all edges of an unknown graph $G = (V, E)$ that is bi-directed and strongly connected. This problem can be solved with a simple algorithm that traverses all edges at most twice, and no algorithm can do better in the worst case. Artificial Intelligence researchers, however, often use the following on-line nearest neighbor algorithm: "repeatedly take a shortest path to the closest unexplored edge and traverse it." We prove bounds on the worst-case complexity of this algorithm. We show, for example, that its worst-case complexity is close to optimal for some classes of graphs, such as graphs with linear or star topology and dense graphs with edge lengths one. In general, however, its complexity can grow faster than linear in the sum of all edge lengths, although not faster than $\log(V)$ times the sum of all edge lengths.

1 Introduction

In this paper, we study how to explore unknown environments that can be modeled as bi-directed graphs. Bi-directed graphs are directed graphs with symmetrical edge lengths that can be obtained from undirected graphs by replacing each undirected edge with a pair of directed edges ("twin edges"), one for each direction. The exploration of bi-directed graphs differs from the exploration of undirected graphs in that one does not learn the twin edge when traversing an edge. We study the following problem:

Definition 1 On-Line Bi-Directed Chinese Postman Problem *Explore all edges of an unknown graph that is bi-directed and strongly connected and return to the starting vertex. One always has a map of all vertices that one has visited and edges that one has explored so far, can recognize them if one sees them again, and knows how many unexplored edges leave each visited vertex, but does not know which vertices they lead to until one has traversed them at least once.*

On-Line Bi-Directed Chinese Postman Problems can be used to model a variety of robot exploration problems, such as mobile robots that have to learn a topological map of an unknown corridor environment. The resulting graph is bi-directed, since corridors can be traversed in both directions. We assume that the robots are not able to recognize a corridor from the opposite direction when they have traversed it in one direction, unless they have already traversed it in the opposite direction as well. Different from piecemeal learning [2], we do not require the robots to return periodically to their starting position for recharging. Similar exploration problems arise in distributed computing when exploring unidirected networks [6] and in task learning, for example when a skill acquired by a newborn does not imply the opposite skill. Examples are opening and closing a jar or switching a TV set on and off.

To measure the complexity of graph learning algorithms, we use the lengths of their exploration tours (closed walks). This is reasonable, because the time that robots need to explore their environment is completely dominated by the time it takes them to move around.

We use the following terminology: Unless stated otherwise, "graph" refers to a graph $G = (V, E)$ that is **strongly connected, weighted, and bi-directed**. A bi-directed graph has twice the number of edges that the corresponding undirected graph has. Bi-directed graphs are special cases of Eulerian graphs (graphs for which the number of incoming edges equals the number of outgoing edges for each vertex.) "Edge" refers to a directed edge $e \in E$. In figures, we often do not label an edge with its length if it has length one. $weight(G)$ denotes the weight of G (the sum of all edge lengths), and $length_G(v, v')$ denotes the smallest length of any path from $v \in V$ to $v' \in V$ on G.

*This research is sponsored in part by the Wright Laboratory, Aeronautical Systems Center, Air Force Materiel Command, USAF, the Advanced Research Projects Agency (ARPA) under grant number F33615-93-1-1330, and the National Science Foundation under grant number IRI-9502548. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations or the U.S. government.

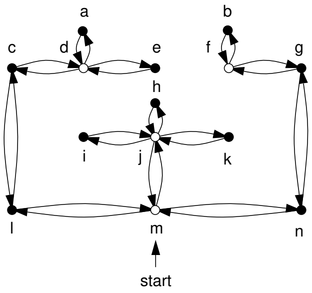

Figure 1: A Simple Bi-Directed Tree

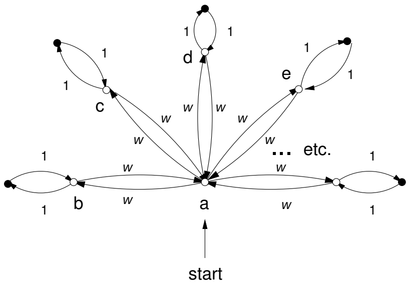

Figure 2: A Simple Bi-Directed Star

The rest of the introduction provides the motivation behind our research. Various graph learning strategies for On-Line Bi-Directed Chinese Postman Problems have been described in the literature. We highlight the advantages of the algorithm that we study in this paper and explain why it has been used by researchers from Artificial Intelligence, although its worst-case complexity has been unknown so far [4]. (The remainder of this section can be skipped by readers who are only interested in the results.)

Consider the following two graph learning strategies:

Definition 2 Depth-First Search (DFS) *"Take unexplored edges whenever possible (ties can be broken arbitrarily). If you are stuck (i.e. cannot take an unexplored edge any longer), backtrack the last unbacktracked edge traversal and repeat the procedure recursively until you are stuck at the starting vertex."*

Definition 3 The "Building a Eulerian Tour" Algorithm (BETA)[1] *"Take unexplored edges whenever possible (ties can be broken arbitrarily). If you are stuck, retrace the tour of unexplored edges just completed, stop at all vertices with outgoing unexplored edges, and apply the algorithm recursively from each such vertex."*

BETA is similar to DFS, but retraces its earlier moves instead of backtracking its later moves when it gets stuck. This has the advantage that BETA can explore arbitrary Eulerian graphs – backtracking is not always possible in Eulerian graphs that are not bi-directed. DFS is able to explore bi-directed graphs, because backtracking is possible on bi-directed graphs and DFS knows how to backtrack when it is stuck, since – at that point in time – it knows all edges that enter and leave its current vertex.

We illustrate BETA and DFS using the graph shown in Figure 1. If both graph learning algorithms start at vertex m, then they can traverse the following sequence of vertices m, l, c, d, c, l, m, j, m, n, g, f, g, n, m, and then get stuck at m. BETA can now continue with l, c, d, a, d, e, d, a, d, e, d, c, l, m, j, i, j, h, j, k, j, i, j, h, j, k, j, m, n, g, f, b, f, b, f, g, n, m, while DFS can explore the rest of the graph as follows: n, g, f, b, f, b, f, g, n, m, j, i, j, h, j, k, j, k, j, h, j, i, j, m, l, c, d, a, d, e, d, e, d, a, d, c, l, m.

Both algorithms traverse every edge of a graph exactly twice and thus the length of their exploration tour is 2 $weight(G)$. The star shown in Figure 2 shows that every graph learning strategy has (asymptotically) at least this complexity in the worst case. Any graph learning strategy can traverse the following sequence of vertices a, b, a, c, a, d, a, ... and eventually get stuck at a. To explore all edges of length one and return to its starting vertex, it is then forced to traverse all edges of length w again. If there are k rays, the length of the exploration tour is at least $4kw + 2k$, the weight of the graph is $2kw + 2k$, and the ratio of the two quantities approaches two for large w.

Thus, it appears that the On-Line Bi-Directed Chinese Postman Problem has been solved already. Researchers in Artificial Intelligence, however, often use the following heuristic (common-sense) graph learning strategy:

Definition 4 The On-Line Nearest Neighbor Algorithm (OnNNA) *"Repeatedly take a shortest path to the closest untraversed edge and traverse it (distances are measured with respect to the explored portion of the graph, ties can be broken arbitrarily), and return on a shortest path to the starting vertex once you have explored all edges."*

A possible sequence of vertices that OnNNA can traverse on the graph from Figure 1 is given in the proof of Theorem 2.

Both Incremental Best-First Search (IBFS) [9] and the Dynamic A* (D*) algorithm [12] are versions of OnNNA. The Learning Real-Time A* (LRTA*) algorithm [7], Prioritized Sweeping [8], and the navigation method by Benson and Prieditis [1] are fast approximations of the behavior of OnNNA. We explain this interest in OnNNA as follows: OnNNA is intuitive, as easy to implement as BETA and DFS, and has three advantages over them: it is able to operate on both non-Eulerian graphs and dynamic graphs that change slowly, and can easily be extended to a goal-directed exploration algorithm that utilizes heuristic knowledge to guide the search towards a goal state (in fact, most of the applications of OnNNA fall into this category).[2]

[1]The exact origin of the algorithm is unclear. [4] and [6] stated it explicitly as a graph learning algorithm, but it has been used earlier as part of proofs about Eulerian tours, for example in [5].

[2]This is done by associating a heuristic value with each unexplored edge that estimates the cost of getting to a goal state after OnNNA has traversed the edge. OnNNA then decides which unexplored edge to traverse next by minimizing the sum of the length of a shortest path from its current vertex to an unexplored edge plus

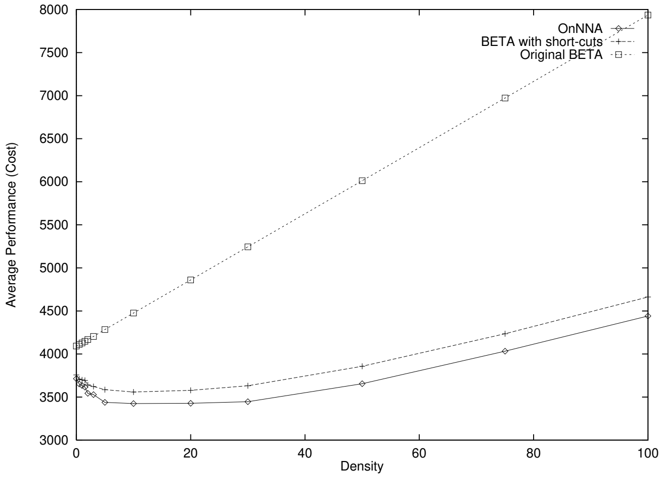

Figure 3: Average-Case Performances

To the best of our knowledge, the performance of OnNNA has never been empirically compared to that of other graph learning algorithms. Our experience is that (uninformed) OnNNA (on static graphs) usually outperforms BETA by far and has a similar performance as "BETA with shortcuts." BETA with shortcuts traverses all edges for the first time in the same order as BETA, but always takes shortest known paths between two consecutive first time edge traversals. To continue our example with the graph shown in Figure 1, once BETA with shortcuts is stuck in m for the first time (after it has traversed the sequence of vertices given earlier), it continues with l, c, d, a, d, e, d, c, l, m, j, i, j, h, j, k, j, m, n, g, f, b, f, g, n, m (if ties are broken in the same way as for BETA) and is 12 edge traversals faster than (the original) BETA.

Figure 3 shows the performance (measured in edge traversals) of OnNNA, BETA, and BETA with shortcuts for learning planar mazes of size 32×32 with different density (the start state was always in the lower right corner). They were constructed by first generating an acyclic maze (tree) and then adding edges. The x axis shows the fraction of edges added (0: the maze is a tree; 100: the maze became a complete grid). For each data point, 50 runs were averaged (10 runs each on 5 different mazes). The figure shows that OnNNA consistently outperformed BETA on mazes, even if BETA was allowed to use shortcuts. However, we also found that the advantage of OnNNA diminishes for simpler topologies. On stars, for example, its performance is very close to that of BETA and almost indistinguishable from that of BETA with shortcuts.

2 Overview of the Results

We are interested in comparing the worst-case complexity of OnNNA to that of other approaches to the On-Line Bi-Directed Chinese Postman Problem. This graph learning problem can be solved with a worst-case complexity of two times the weight of the graph, and no algorithm can do better in the worst case. We therefore pursue the question whether the worst-case complexity of OnNNA is linear in the weight of the graph. More formally,

Definition 5 The On-Line Problem: *Let $G = (V, E)$ be an arbitrary (strongly connected, weighted, and bi-directed) graph and s be any vertex in V. The* **On-Line Nearest Neighbor Algorithm (OnNNA)** *starts at s; it repeatedly takes a shortest path to the closest unexplored edge and traverses it (distances are measured with respect to the explored portion of the graph, ties can be broken arbitrarily); and it returns on a shortest path to its starting vertex once it has explored all edges.*[3] *The resulting tour is called an* **On-Line Nearest Neighbor Algorithm (s, G)-Exploration Tour (OnNNA (s, G)-Exploration Tour)**. *The length of the longest such tour is denoted by $L_{OnNNA}(s, G)$ and determines the complexity of OnNNA. What is the worst-case complexity of OnNNA?*

We show that the worst-case complexity of OnNNA is indeed linear in the weight of the graph for special classes for graphs, such as graphs with linear or star topology and dense graphs with edge lengths one. For graphs with linear or star topology, for example, its worst-case complexity is bounded by $5/2$ times the weight of the graph, although OnNNA does not necessarily traverse every edge only a constant number of times. In general, however, its worst-case complexity is not linear in the weight of the graph: it is $\Omega(\frac{\log |V|}{\log \log |V|} \, weight(G))$ even for simple graphs with tree topology. Since it is also $O(\log |V| \; weight(G))$, it cannot grow more than logarithmically faster than the worst-case complexity of any other graph learning algorithm.

This paper is structured as follows: We first show how to reduce the complexity analysis of OnNNA to that of an off-line TSP algorithm. This has two advantages: First, it is easier to think about the off-line problem. Second, it allows us to utilize results from the literature. We then analyze the off-line problem in depth, and finally use our reduction to obtain results about the worst-case complexity of OnNNA.

In particular, we reduce the on-line problem to the following off-line problem:

Definition 6 The Off-Line Problem: *Let $G = (V, E)$ be an arbitrary graph, $W \subseteq V$ be an arbitrary set of vertices, and s be any vertex in W. The* **Off-Line Nearest Neighbor Algorithm (OffNNA)** *starts at s; it repeatedly takes a shortest path to the closest unvisited vertex in W (distances are measured with respect to the whole graph, ties can be broken arbitrarily); and it returns on a shortest path to its starting vertex once it has visited all vertices in W. The resulting tour is called an* **Off-Line Nearest Neighbor Algorithm (s, W, G)-Exploration Tour (OffNNA (s, G)-Exploration Tour)**. *The length of the longest such tour is*

the value of this edge. In this paper, we are basically studying the uninformed case, where all edge values are zero. However, we are able to reduce the complexity analysis of OnNNA with edge values to this case [11].

[3] If we dropped the requirement that OnNNA return to its starting vertex, its complexity could decrease by at most the weight of the graph, which would not change the nature of our results.

denoted by $L_{OffNNA}(s, W, G)$ and determines the complexity of OffNNA. What is the worst-case complexity of OffNNA?

The off-line problem assumes that the graph is known, the on-line problem assumes that it is unknown. Thus, bi-directed graphs are indistinguishable from undirected graphs for the off-line problem, but not for the on-line problem.

3 Reducing OnNNA to OffNNA

In this section, we show that $L_{OnNNA}(s, G) \leq weight(G) + \max_{W \subseteq V : s \in W} L_{OffNNA}(s, W, G)$ for arbitrary Eulerian graphs $G = (V, E)$ and any $s \in V$. The key idea behind the proof is the following: Any exploration strategy has to traverse every edge at least once before it has explored the unknown graph completely. Every additional traversal constitutes overhead. We call the sequence of edge traversals with the first time traversal of every edge removed the overhead of the exploration tour. We show that the overhead of an OnNNA (s, G)-exploration tour on a Eulerian graph $G = (V, E)$ corresponds to an OffNNA (s, W, G)-TSP tour for some $W \subseteq V$ with $s \in W$. The bound follows immediately. We also give an example that shows that there are cases where one has indeed to maximize over almost all $W \subseteq V$ with $s \in W$ to compute the bound on $L_{OnNNA}(s, G)$.

In later sections, we use this relationship between OnNNA and OffNNA as follows: We first study special classes of graphs on which the worst-case complexity of OffNNA is $O(weight(G))$ for all $W \subseteq V$. Thus, there exists a constant $c > 0$ such that $L_{OffNNA}(s, W, G) \leq c\, weight(G)$ for all $W \subseteq V$. It then follows that, in these cases, $L_{OnNNA}(s, G) \leq (c + 1)\, weight(G)$ and the worst-case complexity of OnNNA is $O(weight(G))$ as well.

Theorem 1 *Let $G = (V, E)$ be an arbitrary (not necessarily bi-directed) Eulerian graph and s be an any vertex in V. Then, the overhead of an OnNNA (s, G)-exploration tour is an OffNNA (s, W, G)-TSP tour for some $W \subseteq V$ with $s \in W$.*

Proof Sketch: We say that OnNNA is stuck at its current vertex when it cannot take an unexplored edge to leave the vertex. Consider an arbitrary OnNNA (s, G)-exploration tour on a Eulerian graph. Since OnNNA always takes an unexplored edge when one is available at its current vertex, it gets first stuck at its starting vertex $s = v_0$. Until then, it has not produced any overhead. OnNNA then moves from v_0 to some vertex v_1 with an unexplored edge. This constitutes overhead, since the edges along the path from v_0 to v_1 have already been explored. When OnNNA has moved to v_1, the unexplored subgraph is Eulerian and the procedure repeats: OnNNA takes only unexplored edges until it gets stuck at v_1. It then moves to some v_2 with an outgoing unexplored edge, and so on until it has explored all edges. Finally, it returns to its starting vertex. Let $W = \{v_0, v_1, \ldots, v_n\}$ be the set of vertices that OnNNA was stuck at. The subsequence of edge traversals that constitute overhead is an OffNNA (v_0, W, G)-TSP tour that visits the vertices v_i in sequence, as is shown in the following:

When OnNNA is stuck at v_i, all vertices v_j with $j > i$ still have at least one incoming unexplored edge (otherwise OnNNA could not get stuck at them later). The unexplored part of G is Eulerian, and thus all of these vertices also have at least one outgoing unexplored edge. Since OnNNA moves from v_i to v_{i+1}, v_{i+1} is the vertex closest to v_i that has an outgoing unexplored edge (this statement holds no matter whether distances are measured with respect to the known part of the graph or the whole graph). Thus, v_{i+1} is the vertex that is closest to v_i among all vertices in W that have not yet been visited by the overhead even if distances are measured with respect to the whole graph. ■

An upper bound on the worst-case complexity of OnNNA follows immediately:

Corollary 1 *Let $G = (V, E)$ be an arbitrary (not necessarily bi-directed) Eulerian graph and s be any vertex in V. Then, $L_{OnNNA}(s, G) \leq weight(G) + \max_{W \subseteq V : s \in W} L_{OffNNA}(s, W, G)$.*

Note that one does not need to maximize over all sets $W \subseteq V$ with $s \in W$ to calculate the bound on $L_{OnNNA}(s, G)$. It is sufficient to maximize over all sets W of vertices at which OnNNA can get stuck during a single run (s is necessarily an element of this set). The following theorem, however, shows that OnNNA can get stuck at almost every subset $W \subseteq V$ with $s \in W$.

Theorem 2 *Let $G = (V, E)$ be an arbitrary (bi-directed) graph that corresponds to an undirected tree, $W \subseteq V$ be an arbitrary set of non-leaf vertices, and s be any vertex in W. Consider an arbitrary OffNNA (s, W, G)-TSP tour. Then, there exists an OnNNA (s, G)-exploration tour with the following properties: its overhead is the given OffNNA (s, W, G)-TSP tour and the set of vertices that OnNNA got stuck at is W.*

Proof Sketch: We say that an edge (a, b) with $a, b \in V$ is reachable from v iff it is unexplored and both the unique shortest path from v to a and the shortest path from v to b (distances are measured with respect to G) do not pass through any vertices in W (although they can have vertices in W as endpoints). Consider an arbitrary OffNNA (s, W, G)-TSP tour. The following algorithm imitates a possible behavior of OnNNA:

1. Set $v := s$.
2. If $W = \emptyset$, then return on a shortest path to s (distances are measured with respect to G) and stop.
3. Perform a Eulerian walk that traverses all edges that are reachable from v exactly once and returns to v.
4. Go to the vertex $v' \in W \setminus \{v\}$ that the OffNNA (s, W, G)-TSP tour visits after v.
5. Delete v from W, set $v := v'$, and go to Step 2.

The overhead of this OnNNA (s, G)-exploration tour is the given OffNNA (s, W, G)-TSP tour, and the set of vertices that OnNNA gets stuck at is W.

As an example, consider the graph shown in Figure 1 and assume that OnNNA should get stuck at the hollow vertices. The proof constructs the following behavior of OnNNA: m, l, c, d, c, l, m, j, m, n, g, f, g, n, m (now OnNNA is stuck at m and has to go to j next), j, i, j, h, j, k, j (now OnNNA is stuck at j and has to go to d (or f) next), m, l, c, d, a, d, e, d (now OnNNA is stuck at d and has to go to f next), c, l, m, n, g, f, b, f (now OnNNA is stuck at f but has traversed all edges and can return to m), g, n, m. ■

4 The Complexity of OffNNA

We first obtain general upper and lower bounds on the worst-case complexity of OffNNA and then study some special classes of graphs on which its worst-case complexity is lower.

4.1 The General Case

In the next two sections, we show that the worst-case complexity of OffNNA is $\Omega(\frac{\log|V|}{\log\log|V|}\,weight(G))$ in general and $O(\log|V|\ weight(G))$. Thus, it is not $O(weight(G))$ in general.

4.1.1 An Upper Bound

In this section, we show that the worst-case complexity of OffNNA is $O(\log|V|\ weight(G))$ for arbitrary graphs $G = (V, E)$, arbitrary $W \subseteq V$, and any $s \in W$. The key idea behind the proof is the application of a result by Rosenkrantz, Stearns, and Lewis for undirected cliques $G = (V, E)$ whose distances satisfy the triangle inequality. They showed that the length of any OffNNA (s, V, G)-TSP tour on such cliques is at most $\frac{1}{2}\lceil\log_2|V|\rceil + \frac{1}{2}$ times the length of a shortest (standard) TSP tour on G, and that this bound is tight [10].

Theorem 3 *Let $G = (V, E)$ be an arbitrary graph, $W \subseteq V$ be an arbitrary set of vertices, and s be any vertex in W. Then, $L_{OffNNA}(s, W, G)$ is at most $\frac{1}{2}\lceil\log_2|W|\rceil + \frac{1}{2}$ times the length of a shortest tour on G that visits all vertices in W.*

Proof: Construct an undirected clique $G' = (W, E')$. The length of an edge $e \in E'$ equals the length of a shortest path between both of its vertices on G. These lengths are symmetrical and satisfy the triangle inequality. The result by Rosenkrantz, Stearns, and Lewis [10] applies to G': the length of any OffNNA (s, W, G')-TSP tour on G' is at most $\frac{1}{2}\lceil\log_2|W|\rceil + \frac{1}{2}$ times the length of a shortest (standard) TSP tour on G'. The theorem then follows immediately, since every OffNNA (s, W, G)-TSP tour on G corresponds to an OffNNA (s, W, G')-TSP tour on G' of the same length that visits the vertices in W for the first time in the same order. ■

The length of a shortest tour that visits all vertices in W is bounded by $weight(G)$, since a Eulerian walk on G visits all vertices in V and traverses every edge exactly once. The following upper bound on the worst-case complexity of OffNNA follows immediately:

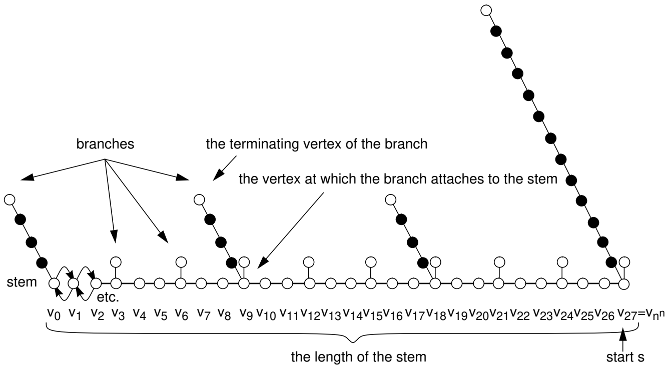

Figure 4: Graph G_1 for $n = 3$ (hollow vertices belong to W)

Corollary 2 *Let $G = (V, E)$ be an arbitrary graph, $W \subseteq V$ be an arbitrary set of vertices, and s be any vertex in W. Then, the worst-case complexity of OffNNA is $O(\log|V|\ weight(G))$.*

For the worst-case example presented by Rosenkrantz, Stearns, and Lewis [10], OffNNA traverses every edge at most once and thus it holds that $L_{OffNNA}(s, W, G) \leq weight(G)$. Consequently, their results do not show whether the worst-case complexity of OffNNA is $O(weight(G))$.

4.1.2 A Lower Bound

In this section, we show that the worst-case complexity of OffNNA is $\Omega(\frac{\log|V|}{\log\log|V|}\,weight(G))$. The proof is by example and uses a simple graph with tree topology.

Theorem 4 *The worst-case complexity of OffNNA is $\Omega(\frac{\log|V|}{\log\log|V|}\ weight(G))$.*

Proof (by example): Consider the graph $G_1 = (V_1, E_1)$ with edge lengths one that corresponds to the following undirected graph: The graph is a tree that consists of a "stem" with several "branches," see Figure 4. The stem has length n^n for some integer $n > 1$ and consists of the vertices $v_0, v_1, \ldots, v_{n^n}$. Table 1 enumerates all branches.

In general, for each integer i with $1 \leq i \leq n$ there are n^{n-i} branches of length $\sum_{j=0}^{i-1} n^j$ each. These branches attach to the stem at the vertices $v_{j\,n^i}$ for integers j; if i is even, then $0 \leq j \leq n^{n-i} - 1$, otherwise $1 \leq j \leq n^{n-i}$. Thus, there are a total of $\sum_{i=1}^{n} n^{n-i} = \frac{n^n - 1}{n-1}$ branches. The weight of G_1 is

$$\begin{aligned}
weight(G_1) &= |E_1| \\
&= 2n^n + 2\sum_{i=1}^{n}\left[n^{n-i}\sum_{j=0}^{i-1} n^j\right] \\
&= 2n^n + 2\sum_{i=1}^{n}\sum_{j=0}^{i-1} n^{n+j-i}
\end{aligned}$$

number of branches	length of each branch	vertices at which the branches attach to the stem
n^{n-1}	1	$v_n = v_{1n}, v_{2n}, v_{3n}, \ldots, v_{n^{n-1}n} = v_{n^n}$
n^{n-2}	$n+1$	$v_{n^n-n^2} = v_{(n^{n-2}-1)\,n^2}, \ldots, v_{2n^2}, v_{1n^2}, v_{0n^2} = v_0$
n^{n-3}	n^2+n+1	$v_{n^3} = v_{1n^3}, v_{2n^3}, v_{3n^3}, \ldots, v_{n^{n-3}n^3} = v_{n^n}$
n^{n-4}	n^3+n^2+n+1	$v_{n^n-n^4} = v_{(n^{n-4}-1)\,n^4}, \ldots, v_{2n^4}, v_{1n^4}, v_{0n^4} = v_0$
...	...	...

Table 1: Branches of Graph G_1

$$= 2n^n + 2\sum_{i=1}^{n} i\, n^{i-1}$$
$$= \frac{(2n^n + 2\sum_{i=1}^{n} i\, n^{i-1})(n-1)^2}{(n-1)^2}$$
$$= \frac{4n^{n+2} - 6n^{n+1} + 2}{(n-1)^2}$$

Now consider the following OffNNA (v_{n^n}, W, G_1)-TSP tour on G_1, where W contains the vertices v_i for the integers $0 \le i \le n^n$ and the terminating vertices of all branches ($|W| = (n^n+1) + \frac{n^n-1}{n-1} = \frac{n^{n+1}+n-2}{n-1}$): OffNNA starts at $s = v_{n^n}$ and visits the vertices from v_{n^n} to v_0 in decreasing order while traveling along the whole stem. It then travels along the whole stem in the opposite direction and visits the terminating vertices of all branches of length one for the first time (in the order in which they are listed in the table above). Next, it switches directions again, travels along the whole stem in the original direction, and visits the terminating vertices of all branches of length $n+1$ for the first time (again, in the order in which they are listed in the table above), and so forth. Once OffNNA has traversed all edges, it returns to v_{n^n}. Thus, it traverses each branch twice, once in each direction. When all of the edges have been traversed, the stem has been traversed $n+1$ times. Then, OffNNA is either at vertex v_{n^n} (and done) or at vertex v_0. In the latter case, it has to traverse the stem once more to get back to its starting vertex v_{n^n}. We do not take this distance into account when calculating a lower bound for $L_{OffNNA}(v_{n^n}, W, G_1)$:

$$L_{OffNNA}(v_{n^n}, W, G_1)$$
$$\ge (n+1)\, n^n + 2\sum_{i=1}^{n}\left[n^{n-i}\sum_{j=0}^{i-1} n^j\right]$$
$$= (n+1)\, n^n + 2\sum_{i=1}^{n} i\, n^{i-1}$$
$$= \frac{((n+1)\, n^n + 2\sum_{i=1}^{n} i\, n^{i-1})(n-1)^2}{(n-1)^2}$$
$$= \frac{n^{n+3} + n^{n+2} - 3n^{n+1} - n^n + 2}{(n-1)^2} \qquad (1)$$

It follows that

$$\frac{L_{OffNNA}(v_{n^n}, W, G_1)}{weight(G_1)}$$

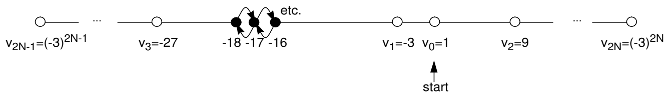

Figure 5: Graph G_2 (hollow vertices belong to W)

$$\ge \frac{n^{n+3} + n^{n+2} - 3n^{n+1} - n^n + 2}{4n^{n+2} - 6n^{n+1} + 2}$$
$$= \frac{1}{4}n + \frac{5}{8} + \frac{3n^{n+1} - 4n^n - 2n + 3}{16n^{n+2} - 24n^{n+1} + 8}$$

Since n is $\Omega(\log |V| / \log\log |V|)$, the worst-case complexity of OffNNA is $\Omega(\frac{\log |V|}{\log\log |V|}\, weight(G))$ even for simple graphs with tree topology. ■

4.2 Special Cases

In the previous section, we have shown that the worst-case complexity of OffNNA is not $O(weight(G))$ in general. The next two sections present two special cases for which it is $\Theta(weight(G))$, namely graphs with linear or star topology and dense graphs with edge lengths one.

4.2.1 Graphs with Linear or Star Topology

Graphs have star topology iff they correspond to undirected graphs that contain at most one vertex (their center) with a degree that is larger than two. In this section, we first give a simple example that shows that OffNNA does not necessarily traverse every edge of a star only a constant number of times – even if the graph is linear, a special case of a star topology. Then we show that, nevertheless, $L_{OffNNA}(s, W, G) \le \frac{3}{2}\, weight(G)$ for arbitrary stars $G = (V, E)$ with center $s \in V$ and arbitrary $W \subseteq V$ with $s \in W$.

To see that OffNNA does not necessarily traverse every edge of a star only a constant number of times, define the points $v_i = (-3)^i$ for $i = 0, 1, \ldots, 2N$ where $N \ge 1$ is an integer, and consider the linear graph G_2 (with edge lengths one) between the integer points from v_{2N-1} to v_{2N} on the real line, see Figure 5. If OffNNA is started at v_0 and has to visit the vertices $W := \{v_i : i = 0, 1, \ldots, 2N\}$, then it visits the vertices v_i for the first time in the order $s = v_0, v_1, \ldots, v_{2N}$ and finally returns to v_0. Thus, the directed edge from 0 to -1 has been traversed $2\,N$ times, which is not a constant [3].

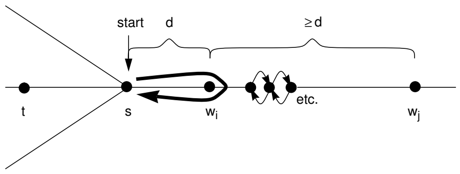

Figure 6: A Ray of a Star

Theorem 5 *Let $G = (V, E)$ be an arbitrary star with center $s \in V$ and edge lengths one and $W \subseteq V$ be an arbitrary set of vertices with $s \in W$. Then, $L_{OffNNA}(s, W, G) \leq \frac{3}{2}\, weight(G)$ and the worst-case complexity of OffNNA is $\Theta(weight(G))$.*

We defer the proof of the theorem to the appendix because it contains quite a few technicalities. Here, we show the idea behind the proof by proving the less tight bound $2weight(G)$:

Proof Sketch: Let $G = (V, E)$ be a star with center $s \in W$ for a given $W \subseteq V$ and consider each ray of the star separately. Assume that OffNNA enters the ray from s, moves as far as vertex w_i, and then moves towards s again, see Figure 6. Assume that there is still at least one unvisited vertex in W on the ray and call the one closest to the center w_j. Since all vertices between s and w_i have already been visited and OffNNA does not visit w_j next, we know that the closest unvisited vertex in W, call it t, is on another ray. In particular, $d := length_G(w_i, s) \leq length_G(w_i, t) \leq length_G(w_i, w_j)$. The length of the continuous path that OffNNA traveled along this ray, from entering the ray to leaving the ray, was $2\, length_G(s, w_i)$. When OffNNA enters the ray next time, it moves at least as far as w_j and, thus, travels at least a distance of $2\, length_G(s, w_j)$ until it leaves the ray again. Since $2\, length_G(s, w_j) = 2\ (length_G(s, w_i) + length_G(w_i, w_j)) \geq 2\,(length_G(s, w_i) + length_G(s, w_i)) = 4\, length_G(s, w_i)$, the length of a continuous path along any ray is at least twice as long as the length of the previous continuous path along the same ray. The total distance traveled along the ray is, therefore, at most twice the length of the last continuous path along the ray, which – in turn – is not larger than the number of edges of the ray. Thus, we have shown that $L_{OffNNA}(s, W, G) \leq 2\,|E| = 2\, weight(G)$ for arbitrary $W \subseteq V$. ■

The theorem generalizes to stars with arbitrary (positive) integer or rational edge lengths:

Corollary 3 *Let $G = (V, E)$ be an arbitrary star with center $s \in V$ and $W \subseteq V$ be an arbitrary set of vertices with $s \in W$. Then, $L_{OffNNA}(s, W, G) \leq \frac{3}{2}\, weight(G)$ and the worst-case complexity of OffNNA is $\Theta(weight(G))$.*

Proof: A graph with (positive) integer edge lengths can be transformed into a graph with edge lengths one by splitting all edges into several edges of length one. This does not change the graph weight, and every path on the original graph has the same length as the corresponding path on the transformed graph. Any OffNNA (s, W, G)-TSP tour on the original graph remains a valid OffNNA (s, W, G)-TSP tour on the transformed graph with the same length and vice versa, for arbitrary $W \subseteq V$. Similarly, a graph with (positive) rational edge lengths can be transformed into one with integer edge lengths by scaling all edge lengths equally. Any OffNNA (s, W, G)-TSP tour on the original graph remains a valid OffNNA (s, W, G)-TSP tour on the transformed graph and vice versa. The transformation changes the graph weight and the length of any path by the same factor. Thus, any linear relationship between the two quantities remains intact, and the theorem follows immediately from Theorem 5. ■

Notice that the corollary also directly applies to linear graphs, no matter at which vertex OffNNA is started, since they are special cases of stars whose center is the starting vertex of OffNNA.

4.2.2 Dense Graphs with Edge Lengths One

We call a graph dense iff $|E| = \Omega(|V| \log |V|)$. In this section, we show that the worst-case complexity of OffNNA is $\Theta(weight(G))$ for arbitrary graphs $G = (V, E)$ that are dense and have edge lengths one, arbitrary $W \subseteq V$, and any $s \in W$. This result follows from Theorem 3.

Theorem 6 *Let $G = (V, E)$ be an arbitrary graph that is dense ($|E| = \Omega(|V| \log |V|)$) and has edge lengths one, $W \subseteq V$ be an arbitrary set of vertices, and s be any vertex in W. Then, the worst-case complexity of OffNNA is $\Theta(weight(G))$.*

Proof: Pick an arbitrary bi-directed minimum spanning tree T of G. Use a Eulerian tour on T to visit all vertices in V and return to s. It traverses every edge of T exactly once, for a total of $2|V| - 2$ edge traversals. Hence, the length of a shortest tour on G that visits all vertices in W is at most $2|V| - 2$. Thus, $L_{OffNNA}(s, W, G) \leq (\frac{1}{2}\lceil \log_2 |W| \rceil + \frac{1}{2})(2|V| - 2) \leq O(|V| \log |V|) = O(|E|) = O(weight(G))$ according to Theorem 3. ■

5 The Complexity of OnNNA

In this section, we first obtain general upper and lower bounds on the worst-case complexity of OnNNA and then study some special classes of graphs on which its worst-case complexity is lower. The key idea behind the proofs is the application of Corollary 1 to transfer our results from OffNNA to OnNNA.

Theorem 7 *Let $G = (V, E)$ be an arbitrary graph and s be any vertex in V. Then, the worst-case complexity of OnNNA over all such problems is $\Omega(\frac{\log |V|}{\log \log |V|}\ weight(G))$ and $O(\log |V|\ weight(G))$.*

Proof: The upper bound follows immediately from Corollary 2 in conjunction with Corollary 1.

The lower bound can be proved by adapting Graph G_1, the example that we used to prove a lower bound on the worst-case complexity of OffNNA, see Figure 4. Consider the tree

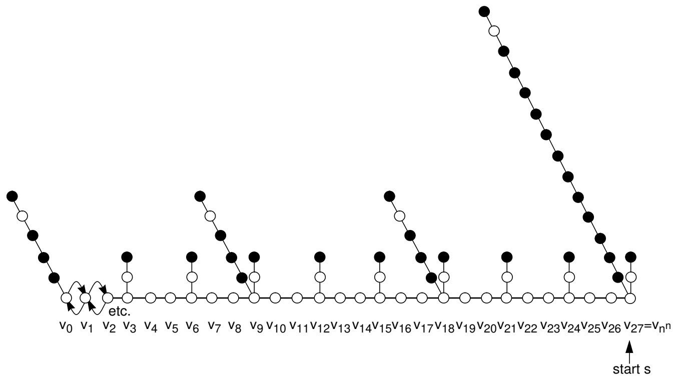

Figure 7: Graph G_3 for $n = 3$ (OnNNA gets stuck at the hollow vertices)

$G_3 = (V_3, E_3)$ with edge lengths one that can be obtained from G_1 by increasing the length of every branch by one, see Figure 7. The weight of G_3 can be calculated from the weight of G_1, since each branch is extended by two edges (one in each direction):

$$\begin{aligned}
&weight(G_3) = |E_3| \\
&= weight(G_1) + 2\frac{n^n - 1}{n - 1} \\
&= \frac{4n^{n+2} - 6n^{n+1} + 2}{(n-1)^2} + 2\frac{n^n - 1}{n - 1} \\
&= \frac{4n^{n+2} - 4n^{n+1} - 2n^n - 2n + 4}{(n-1)^2}
\end{aligned}$$

Now consider the set W of non-leaf vertices that contains all vertices of the stem of G_3 plus the vertices that connect to the terminating vertices of all branches. W is the same set of vertices that we used to show that there exists an OffNNA (v_{n^n}, W, G_1)-TSP tour on G_1 whose length is not $O(weight(G_1))$. This tour is also an OffNNA (v_{n^n}, W, G_3)-TSP tour on G_3. Consequently, Theorem 2 applies and there exists an OnNNA (v_{n^n}, G_3)-exploration tour whose overhead is the OffNNA (v_{n^n}, W, G_3)-TSP tour. The proof of Theorem 2 shows how this behavior of OnNNA can be obtained. In this case, OnNNA traverses every edge of G_3 once and incurs as additional overhead the length of the OffNNA (v_{n^n}, W, G)-TSP tour that we calculated earlier in Inequality 1:

$$\begin{aligned}
&L_{OnNNA}(v_{n^n}, G_3) \\
&\geq |E_3| + \frac{n^{n+3} + n^{n+2} - 3n^{n+1} - n^n + 2}{(n-1)^2} \\
&= \frac{n^{n+3} + 5n^{n+2} - 7n^{n+1} - 3n^n - 2n + 6}{(n-1)^2}
\end{aligned}$$

It follows that

$$\begin{aligned}
&\frac{L_{OnNNA}(v_{n^n}, W, G_3)}{weight(G_3)} \\
&\geq \frac{n^{n+3} + 5n^{n+2} - 7n^{n+1} - 3n^n - 2n + 6}{4n^{n+2} - 4n^{n+1} - 2n^n - 2n + 4} \\
&= \frac{1}{4}n + \frac{3}{2} - \frac{n^{n+1} - n^2}{8n^{n+2} - 8n^{n+1} - 4n^n - 4n + 8}
\end{aligned}$$

Since n is $\Omega(\log|V|/\log\log|V|)$, the worst-case complexity of OnNNA is $\Omega(\frac{\log|V|}{\log\log|V|}\, weight(G))$ even for simple graphs with tree topology. ■

The theorem implies that the worst-case complexity of OnNNA is not $O(weight(G))$ in general. The next theorems present two special cases for which it is $\Theta(weight(G))$, namely graphs with linear or star topology and dense graphs with edge lengths one.

Theorem 8 *Let $G = (V, E)$ be an arbitrary star with center $s \in V$. Then, $L_{OnNNA}(s, G) \leq \frac{5}{2}\, weight(G)$ and the worst-case complexity of OnNNA is $\Theta(weight(G))$.*

Proof: The theorem follows immediately from Corollary 3 in conjunction with Corollary 1. ■

The theorem also applies to linear graphs, no matter at which vertex OnNNA is started. The example from Figure 5 can be used in conjunction with Theorem 2 to show that OnNNA does not necessarily traverse every edge only a constant number of times (even for linear graphs).

The theorem implies that the worst-case complexity of OnNNA is close to optimal on stars, since – in the worst case – no graph learning algorithm can asymptotically do better than $2\, weight(G)$ on stars (see Section 1).

Theorem 9 *Let $G = (V, E)$ be an arbitrary graph that is dense ($|E| = \Omega(|V|\log|V|)$) and has edge lengths one and s be any vertex in V. Then, the worst-case complexity of OnNNA is $\Theta(weight(G))$.*

Proof: The theorem follows immediately from Theorem 6 in conjunction with Corollary 1. ■

6 Conclusion

In this paper, we have analyzed the following simple graph learning algorithm, called the on-line nearest neighbor algorithm: "Repeatedly take a shortest path to the closest unexplored edge and traverse it." We have argued that the flexibility of this algorithm provides a good basis for its use in practical applications that require capabilities to learn graphs or find goal states in unknown graphs ("treasure hunting").

We have shown that the worst-case complexity of the nearest neighbor algorithm is $\Omega(\frac{\log|V|}{\log\log|V|}\, weight(G))$ on bi-directed graphs $G = (V, E)$ even for simple graphs with tree topology. Since the graph learning problem can be solved with a worst-case complexity of only $\Theta(weight(G))$, the flexibility of the on-line nearest neighbor algorithm comes at the cost of a loss of performance in the worst case. However, the worst-case complexity of the on-line nearest neighbor algorithm is $O(\log|V|\ weight(G))$, which implies that it cannot grow more than logarithmically faster than the worst-case complexity of any other graph learning algorithm. We also described special classes of graphs on which its worst-case complexity is only $\Theta(weight(G))$.

To summarize, we have shown that the worst-case complexity of OnNNA is $\Omega(\frac{\log|V|}{\log\log|V|}\ weight(G))$ and $O(\log|V|\ weight(G))$. It is currently an open problem whether one of these bounds is tight.

In our current work, we are pursuing two directions: First, we are extending our complexity analysis of the nearest neighbor algorithm to other learning capabilities of the algorithm (such as being able to learn all edges in the vicinity of a traversed edge), other tasks (such as partially informed treasure hunting), and more realistic special classes of graphs (such as mazes). Second, we are developing algorithms that maintain the flexibility of the on-line nearest neighbor algorithm, but are able to make performance guarantees that are linear in the weight of the graph [11].

Acknowledgements

Thanks to Avrim Blum and Lonnie Chrisman for stimulating discussions and to Peter Stone for helpful comments.

References

[1] G.D. Benson and A. Prieditis. Learning continuous-space navigation heuristics in real time. In *Proceedings of the Conference on Simulation of Adaptive Behavior: From Animals to Animats*, 1992.

[2] M. Betke, R. Rivest, and M. Singh. Piecemeal learning of an unknown environment. *Machine Learning*, 18(2/3), 1995.

[3] A. Blum, P. Chalasani, D. Coppersmith, B. Pulleyblank, P. Raghavan, and M. Sudan. The minimum latency problem. In *Proceedings of the Symposium on Theory of Computing*, pages 163–171, 1994.

[4] X. Deng and C.H. Papadimitriou. Exploring an unknown graph. In *Proceedings of the Symposium on Foundations of Computer Science*, pages 355–361, 1990.

[5] C. Hierholzer. Über die Möglichkeit, einen Linienzug ohne Wiederholung und ohne Unterbrechung zu umfahren. *Mathematische Annalen*, 6:30–32, 1873 (sic!).

[6] E. Korach, S. Kutten, and S. Moran. A modular technique for the design of efficient distributed leader finding algorithms. *ACM Transactions on Programming Languages and Systems*, 12(1):84–101, 1990.

[7] R.E. Korf. Real-time heuristic search. *Artificial Intelligence*, 42(2-3):189–211, 3 1990.

[8] A.W. Moore and C.G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13:103–130, 1993.

[9] J.C. Pemberton and R.E. Korf. Incremental path planning on graphs with cycles. In *Proceedings of the AI Planning Systems Conference*, pages 179–188, 1992.

[10] D.J. Rosenkrantz, R.E. Stearns, and P.M. Lewis II. An analysis of several heuristics for the traveling salesman problem. *SIAM Journal of Computing*, 6(3):563–581, 1977.

[11] Y. Smirnov, S. Koenig, M. Veloso, and R. Simmons. Efficient goal-directed exploration. In *Proceedings of the National Conference on AI*, page to appear, 1996.

[12] A. Stentz. The focussed D* algorithm for real-time replanning. In *Proceedings of the IJCAI*, pages 1652–1659, 1995.

Appendix: Proof of Theorem 5

Theorem 5 *Let $G = (V, E)$ be an arbitrary star with center $s \in V$ and edge lengths one and $W \subseteq V$ be an arbitrary set of vertices with $s \in W$. Then, $L_{OffNNA}(s, W, G) \leq \frac{3}{2}\ weight(G)$ and the worst-case complexity of OffNNA is $\Theta(weight(G))$.*

Proof: Let $G = (V, E)$ be a star with center $s \in W$ for a given $W \subseteq V$ and consider each ray of the star separately. Assume that OffNNA enters the ray for the ith time ($i = 1, 2, \ldots, n$ for an $n \geq 0$). When OffNNA enters a ray, it always moves along the ray up to some previously unvisited vertex $w_i \in W$, and then returns to s and leaves the ray. On its way from s to w_i (including the endpoint) it visits at least one previously unvisited vertex in W (and potentially more than one). Call the first such vertex that OffNNA encounters on this trip $v_i \in W$.

Assume that there are still at least two unvisited vertices in W when OffNNA leaves the ray, one on the ray and another on some other ray. The one on the ray that is closest to s is v_{i+1}. It holds $length_G(s, v_i) \leq length_G(s, w_i) < length_G(s, v_{i+1})$. We denote the one on some other ray that is closest to s by t_i. Since OffNNA was at s and decided to visit v_i next instead of t_i, we know that $length_G(s, v_i) \leq length_G(s, t_i)$. When OffNNA is at w_i and decides to visit t_i next instead of v_{i+1}, we know that $length_G(w_i, t_i) \leq length_G(w_i, v_{i+1})$. Put together, it follows that

$$\begin{aligned}
& 3\,(length_G(s, w_{i+1}) - length_G(s, w_i)) \\
= \;& 2\,length_G(s, w_{i+1}) + length_G(s, w_{i+1}) - 3\,length_G(s, w_i) \\
= \;& 2\,length_G(s, w_{i+1}) + length_G(s, w_i) + length_G(w_i, v_{i+1}) \\
& + length_G(v_{i+1}, w_{i+1}) - 3\,length_G(s, w_i) \\
\geq \;& 2\,length_G(s, w_{i+1}) + length_G(s, w_i) + length_G(w_i, t_i) \\
& + length_G(v_{i+1}, w_{i+1}) - 3\,length_G(s, w_i) \\
= \;& 2\,length_G(s, w_{i+1}) + length_G(s, w_i) + length_G(w_i, s) \\
& + length_G(s, t_i) + length_G(v_{i+1}, w_{i+1}) - 3\,length_G(s, w_i) \\
\geq \;& 2\,length_G(s, w_{i+1}) + length_G(s, w_i) + length_G(w_i, s) \\
& + length_G(s, v_i) + length_G(v_{i+1}, w_{i+1}) - 3\,length_G(s, w_i) \\
= \;& 2\,length_G(s, w_{i+1}) + length_G(v_{i+1}, w_{i+1}) + length_G(s, v_i) \\
& - length_G(s, w_i) \\
= \;& 2\,length_G(s, w_{i+1}) + length_G(v_{i+1}, w_{i+1}) \\
& - length_G(v_i, w_i) \qquad (2)
\end{aligned}$$

Now we can calculate an upper bound on the total length $2\sum_{i=1}^{n} length_G(s, w_i)$ of the OffNN (s, W, G)-TSP tour that OffNNA travels along the ray. It holds that

$$
\begin{aligned}
& 2\sum_{i=1}^{n} length_G(s,w_i) \\
\leq\quad & 2\,length_G(s,w_1) + 2\sum_{i=2}^{n} length_G(s,w_i) \\
& + length_G(s,w_1) + length_G(v_n,w_n) - length_G(v_1,w_1) \\
& \text{(because } length_G(s,w_1) \geq length_G(v_1,w_1)\text{)} \\
=\quad & 3\,length_G(s,w_1) + \sum_{i=2}^{n}[2\,length_G(s,w_i) \\
& + length_G(v_i,w_i) - length_G(v_{i-1},w_{i-1})] \\
\overset{\text{see (2)}}{\leq}\quad & 3\,length_G(s,w_1) \\
& + 3\sum_{i=2}^{n}[length_G(s,w_i) - length_G(s,w_{i-1})] \\
=\quad & 3\,length_G(s,w_n) \qquad (3)
\end{aligned}
$$

which is less than or equal to $\frac{3}{2}$ times the number of edges of the ray. If we sum over all rays, we obtain $L_{OffNNA}(s,W,G) \leq \frac{3}{2}\,|E| = \frac{3}{2}\,weight(G)$. ■