Student Names:
Jonas P. Dyreby
Menadjlia Sofiane
Jonathan Adshead

Applied Algorithmics - 4TIN915U

Solving DS problem on graph of France major highways
Graph: 1.116 Vertices, 8.094 edges, $|DS| = 117$

Contents

1 Submission 1

1.1 Data simplification

Extracting data
At the beginning we made the deliberate choice to work with a smaller dataset by focusing on service stations and street nodes in Bordeaux. This decision allowed us to manage the data more effectively. Once we were confident in the data cleaning process, our plan was to expand our analysis to the Aquitaine region. But as expected, the computation time for the edge calculations did increase when using a larger dataset. For our final submission, we covered the entirety of France. This strategic shift proved advantageous, as the unaltered graph of France, devoid of any simplification featured 2000 service stations, over 500,000 street nodes and 55,000 ways. Notably, a significant proportion of the street nodes functioned as junctions, serving the sole purpose of connecting various ways.

1.2 Create Graph of Service Stations based on the highways

Initial strategy
The initial concept involved commencing the project with a focus on highways. Within the dataset, ”ways” were defined as lists of IDs representing the path. Every Service station was a ”way”, containing a few nodes. We decided to just take the first node, because in the end we thought of writing a function, which gets a list of coordinates and returns the IDs of the service stations. We associated each service station with its nearest street node and subsequently eliminated all other street nodes. This process yielded arrays of service station nodes located on the same highway. However, challenges arose during the creation of edges between junctions to properly connect the highways.

We discovered two main flaws in our implementation. Firstly, multiple service stations were mapped to the same street node or merged due to proximity. Secondly, the resulting graph lacked connectivity, with junctions only providing partial connections. After further inspection, we realized a highway contains only one or two nodes after every node except the mapped service stations and junctions has beed removed. The reason was that a highway was represented through multiple ways.

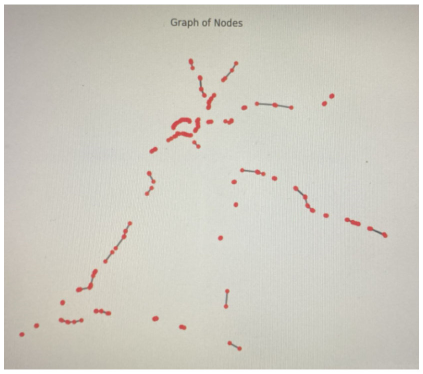


Figure 1: Not connected Graph of Aquitaine

Refined strategy
Upon this realization, our approach was revised. In our new approach, we constructed a complete graph based on the junctions, considering that just some junction nodes possessed a "motorway junction" property. We used the fact, that all junctions were present in at least two ways. With this information, we generated a comprehensive graph. Subsequently, we remapped the service stations onto highway nodes, which were part of the previously saved ways, enabling their placement between two junction nodes.

Despite these adjustments, the resultant graph retained a level of complexity. The count of street nodes decreased from over 500,000 to nearly 53,000 nodes. This reduction stemmed from two primary factors. Firstly, every street node not designated as representing a service station was removed. Secondly, junctions, which previously appeared multiple times in distinct ways, now manifested a singular presence due to way merging.

The service stations or now marked street nodes also experienced a decrease, shifting from 2000 to 1600. This decline can be attributed to the presence of some service stations appearing twice in the dataset, representing each direction of the street. To address this, we merged service stations located near identical marked street nodes. Additionally, our refined graph featured over 53,000 edges, indicative of its interconnected nature.

Reducing graph complexity using a clustering algorithm
In our pursuit of refining the graph structure, we introduced a clustering algorithm aimed at reducing complexity, specifically for NP-complete problems like the dominating set or the k-coloring on the graph. This algorithm is instrumental in grouping nodes with similar characteristics, in our case the position, thereby creating clusters that can be treated as single entities.

Such a simplification proves advantageous in enhancing computational efficiency and resource utilization. In our case, the number of street-nodes and junction nodes went down from 53.000 to 2337 nodes. We decided to set the radius in which **the nodes are merged to 5 km**. With this high radius, we went from a huge number of nodes to a computable graph where we can compute np-complete algorithms in reasonable time. Out of the 2337 nodes, 1116 were service stations and the rest were junctions.

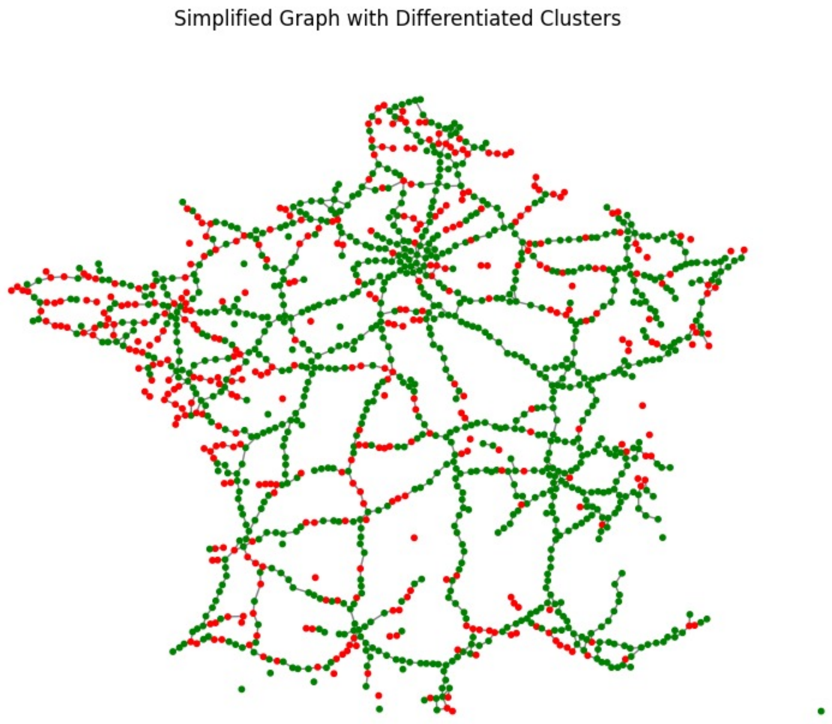


Figure 2: clustering within 5 km radius with junction in red

The process of junction removal presented more complexities than expected. Our initial approach involved the deletion of a junction, along with its associated edges, followed by the connection of all nodes that previously linked to the junction. However, this method revealed certain drawbacks. The primary challenge was the considerable time investment required. Our strategy involved systematically examining each node and, in the case of identifying a junction, meticulously traversing through every connected edge for deletion. Given the scale of our network with over 2000 nodes and edges, this process incurred a significant temporal overhead.

Another notable complication stemmed from the generation of extraneous edges during the elimination process. This unintended consequence necessitated subsequent removal, including edges linking a node to itself or instances of duplicate edges between two nodes. These intricacies added an additional layer of complexity to the overall process, demanding careful attention and corrective measures.

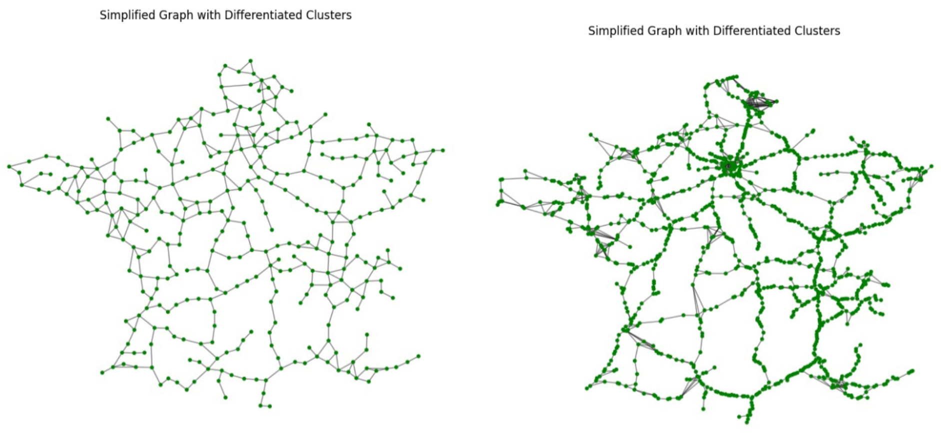

Simplified Graph with Differentiated Clusters
Simplified Graph with Differentiated Clusters

Figure 3: left: 30 km radius, right: 5 km radius

Theoretical Foundation of clustering algorithms

The clustering algorithm relies on principles from graph theory, specifically the identification and grouping of nodes based on similarity metrics. Let G be a graph with vertices V and edges E, and let C represent the resulting clusters. The algorithm aims to maximize intra-cluster similarity while minimizing inter-cluster similarity, creating cohesive groups of nodes.
Formally, the intraclustering process is formalized as follows:

$$\max\left\{\sum_{u,v\in C} \parallel v-u \parallel\right\} \leq 5km,\ \text{for } u \neq v$$

In our case, if the cluster had a service station in it, the cluster was made to a service station and if the complete cluster consists only of junctions, it is a junction.

Complexity Analysis

The time complexity of the clustering algorithm is determined by its double loop. The function iterates over all nodes $|V|$ in the graph and checks for proximity with other nodes to form clusters. In the worst case, it iterates over all pairs of nodes in the graph, resulting in a complexity of $O(|V|^2)$. The complexity is further affected by the nested loop when adding edges to the simplified graph. However, this depends on the structure of the input graph. *The Haversine Distance Calculation* is used to calculate the distance between the nodes, but its time complexity is generally considered $O(1)$, because the number of operations remains constant regardless of the size of the input. The used networkx function *nx.connected_components* has a complexity of $O(|V+E|)$, where V is the number of nodes and E is the number of edges in the graph.

Overall the most significant party is the double loop with a time complexity of $O(|V|^2)$.
The space complexity, the storage required for the simplified graph is $O(|V+E|)$, but depending on the input graph, the storage of the clusters can scale up.

The time complexity of the clustering algorithm depends on the specific algorithm that is used, but in general, it is influenced by the number of nodes and edges in the graph. Many clustering algorithms have polynomial time complexity, making them suitable for moderately sized graphs. The space complexity is typically determined by the storage requirements for the resulting clusters and any additional data structures used during the clustering process.

1.3 Floyd-Warshall algorithm

The Floyd-Warshall algorithm is instrumental in computing the shortest paths between all pairs of nodes in a graph. However, for our specific application, an easy optimization is possible. Rather than considering all edges, our focus lies on identifying edges with distances shorter than 60 km. This stems from our objective: ensuring that for every freeway journey exceeding 60 km, there exists at least one accessible charging station within proximity.

The Floyd-Warshall algorithm iterates through the edges, calculating the shortest path each time. It remains indifferent to the absence of an edge at the current iteration, marking it with positive infinity or a sufficiently large number in the matrix. After constructing the matrix and determining the shortest paths, we filter the results, retaining only those edges with a distance/weight less than 60 km.

After the calculations, the graph still had 1116 nodes, which represent the service stations, the number of edges went up to 8094 and the maximum degree went up to 58. This dramatic increase of edges and the degree is mainly because of the big cities like Paris, Bordeaux, Toulouse or Lyon since we create edges between all nodes, which can be reached in 60 km via the streets.

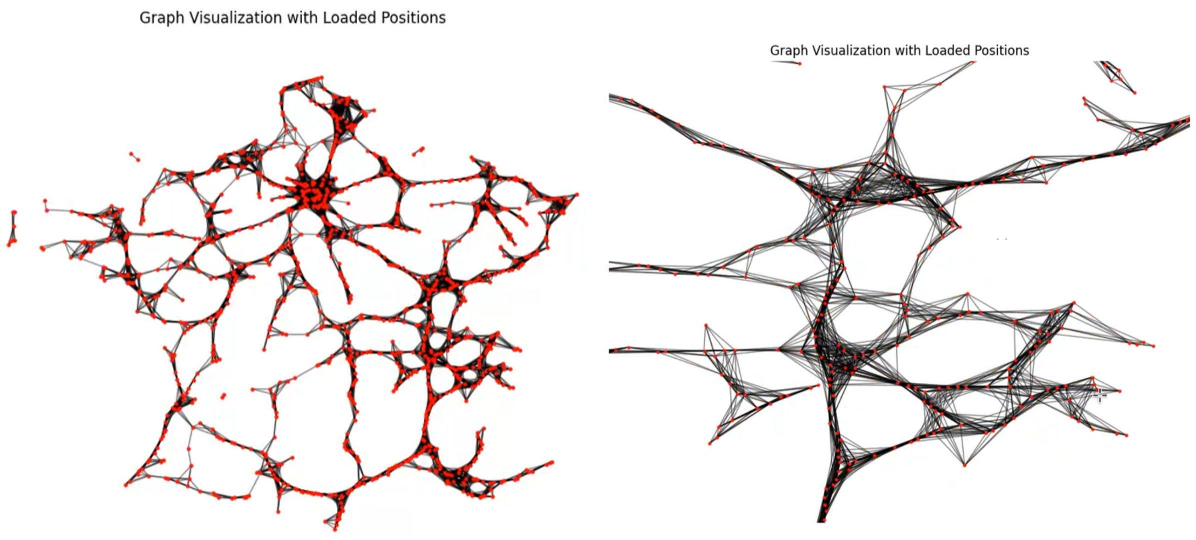

Graph Visualization with Loaded Positions
Graph Visualization with Loaded Positions

Figure 4: Final post Floyd-Warshall graph.

Floyd-Warshall Algorithm in theorie

The Floyd-Warshall algorithm, a dynamic programming approach, computes the shortest paths between all pairs of vertices in a weighted graph. Let G be a graph with vertices V and edges E, and let d_{ij} represent the shortest distance from vertex i to vertex j. The algorithm operates on an adjacency matrix D, initialized with edge weights were available and set to positive infinity otherwise. The recursive formula for the Floyd-Warshall algorithm is given by:

$$d_{ij}^{(k)} = \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right)$$

where k represents the intermediate vertices. The algorithm iterates through all vertices as intermediate points, updating the shortest paths until the complete matrix D is obtained.
In our application, the optimization involves filtering the results based on a distance threshold of 60 km, ensuring that only edges meeting the charging station proximity criterion are considered.

Algorithm 1 Floyd-Warshall Algorithm

- $dist$: a matrix of minimum distances initialized to ∞
- $w(u, v)$: the weight of the edge (u, v)
- $|V|$: the number of vertices in the graph

```
for all $(u, v)$ in edges do
    $dist[u][v] \leftarrow w(u, v)$                    ▷ The weight of the edge $(u, v)$
end for
for all vertices $v$ do
    $dist[v][v] \leftarrow 0$
end for
for $k$ from 1 to $|V|$ do
    for $i$ from 1 to $|V|$ do
        for $j$ from 1 to $|V|$ do
            if $dist[i][j] > dist[i][k] + dist[k][j]$ then
                $dist[i][j] \leftarrow dist[i][k] + dist[k][j]$
            end if
        end for
    end for
end for
```

Complexity Analysis of Floyd-Warshall

The time complexity of the Floyd-Warshall algorithm is determined by its triple-nested loop structure. For each of the $|V|$ vertices, the algorithm considers all pairs of vertices as intermediate points, resulting in a cubic time complexity of $O(|V|^3)$. This arises from the need to iteratively update the shortest paths using the recursive formula.

Additionally, the space complexity is $O(|V|^2)$, stemming from the necessity to maintain a matrix of distances between all pairs of vertices. It's important to note that while the Floyd-Warshall algorithm is efficient for small to moderately sized graphs, its cubic time complexity makes it less suitable for larger graphs where more optimized algorithms may be preferred.

2 Submission 2

2.1 Implement a checker for DS

Formally the dominating set is on a graph $G = (V, E)$ is a subset $S \subseteq V$ such that all $v \in V$, either $v \in S$ or a adjacent vertex u to v is in S.

Algorithm 2 Dominating Set Checker Algorithm for k = 1

- *unmarked*: list of all *nodes* of the *graph*
- *solution*: the *dominating set* to check

```
for all node in domnating set do
    if node in graph then
        remove current node and all neighbors from unmarked
    end if
end for
if unmarked not empty then
    return $false$
end if
return $true$
```

Complexity Analysis of Dominating Set Checker
Let $|N|$ be the number of nodes in a graph, and $|M|$ represent the size of the dominating set solution. In the worst-case scenario, the dominating set solution encompasses all nodes in the graph, resulting in a time complexity of $O(|N|)$. The verification of node membership in the graph incurs a worst-case time complexity of $O(|M|)$. Consequently, the overall algorithmic complexity is $O(|N| + |M|)$. Although the node existence check may be deemed unnecessary for achieving the fastest algorithmic performance, its inclusion enhances the robustness of the code, especially when dealing with large-scale graphs.

2.2 A greedy ln-approximation of smallest of DS

We implemented a polynomial time greedy force algorithm for finding a small (*not a smallest*) dominating set [1]. The dominating set is computed in the following simple steps:

Given our graph $G = (V, E)$ with maximum degree denoted deg(G), we start by initializing the dominating set $S = \emptyset$ and greedily choose vertices to add to this set.
That is, starting from the most connected vertex $v \in V$, we 'mark' all the neighboring verticies $\mathbf{u}$ as visited and add v to S. Then we look for a new $v_{new} \in V/\{v_{previous}, \mathbf{u}_{previous}\}$ and so on. Our Greedy algorithm does this in time complexity $\mathcal{O}(n^2)$. See *Algorithm3* below:

[1] ac.informatik.uni-freiburg.de/teaching/ss$_1 2/netalg/lectures/chapter7.pdf$

Algorithm 3 Polynomial time greedy Dominating Set algorithm

```
$V \leftarrow V(G)$
$S \leftarrow \emptyset$
**while** $\exists$ unmarked vertex **do**                                  ▷ From unmarked verticies
    Pick $v \in \{v \big| \deg(v) = \max_{v \in V}(\deg(v))\}$          ▷ Pick most connected vertex
    Mark $\mathbf{u}$ neighbors to $v$
    $S \leftarrow S \cup \{v\}$
    $V \leftarrow V/\{v, \mathbf{u}\}$
**end while**
```

As stated we have the maximum degree of our graph $\deg(G) = 58$. Now, for an undirected graph with maximum degree $deg(G)$ the greedy algorithm above (*Algorithm* 3) returns a $\ln(\deg(G) + 2)$ - approximation of the *smallest* dominating set. That is, in the worst case we have the following relationship between the size of greedy solution $|S|$ and the size of the optimal smallest dominating set $|S^*|$:

$$\frac{|S|}{|S^*|} \leq \ln(\deg(G)) + 2$$
$$\Rightarrow |S| \leq (\ln(58) + 2)|S^*|$$

This approximation result is a theorem presented in Freiberg University course notes[Algorithms and complexity, Theorem 7.2][2].

This upper bound to the size of the approximated dominating set, is not very restrictive. This is mainly due to large maximum degree of our graph, which can be attributed to areas that are densely populated with service stations and restareas. Looking at the Floyd Warshall graph *Figure 4*, some examples of particularly dense areas are Lille, Paris and Lyon.

In pratice, Algorithm 3 does far better than the upper bound might suggest, *See section 'Run Our Project'; Tables.*

Our greedy Dominating Set solution

Our Dominating Set solution, employing our greedy algorithm, yields a dominating set with a size of 117. Despite its modest size, this set effectively covers the highways spanning the entirety of France. However, it is important to acknowledge that our solution is not flawless, as we incorporated certain approximations. These include the mapping of service stations to street nodes and the clustering applied to enhance graph manageability.

It's noteworthy that our algorithm outperforms the Dominating Set algorithm, implemented in the networkx library only considering the size of the solution. It achieves a superior solution with 119 nodes compared to 154 nodes of the networkx implementation. Initially impressed by our greedy algorithm's efficacy, we later observed that the networkx algorithm completes its computation in about 5 seconds, whereas ours takes nearly 1 minute. This suggests that networkx employs an even more approximated algorithm. This characteristic becomes advantageous when dealing with larger graphs. Had we not employed the clustering algorithm and remained with around 20,000 nodes out of the initial 53,000, our greedy algorithm would have experienced prolonged execution times,

[2]https://ac.informatik.uni-freiburg.de/teaching/ss$_1$2$/netalg/lectures/chapter$7$.pdf$

while the networkx algorithm would have maintained a reasonable processing time. Consequently, our algorithm demonstrates better suitability for smaller graphs.

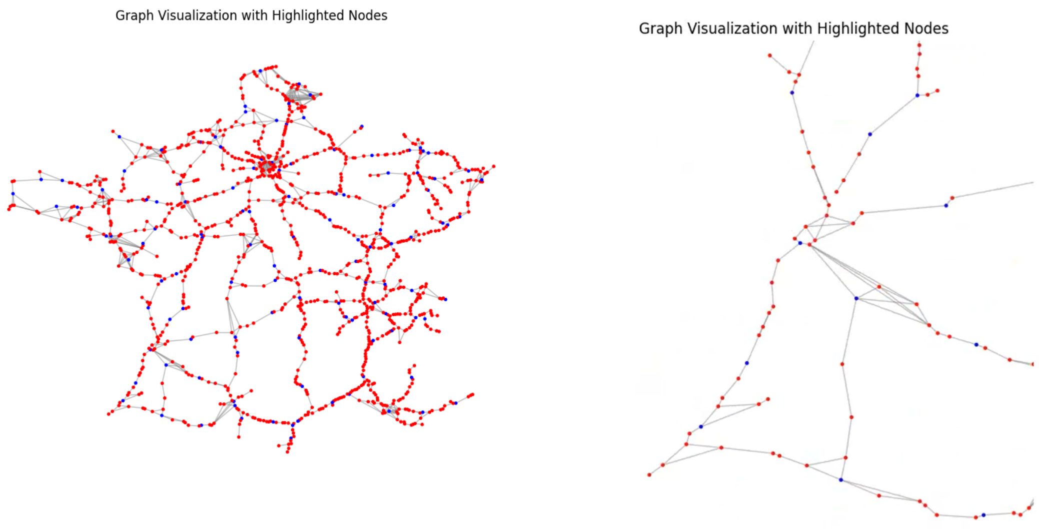

Graph Visualization with Highlighted Nodes
Graph Visualization with Highlighted Nodes

Figure 5: Greedy DS-solution for France, right: zoomed in Aquitaine

2.3 Brute force Dominating Set algorithm

The application of the brute force Dominating Set algorithm aims to derive optimal or near-optimal solutions for dominating sets. However, due to the NP-complete nature of the problem, as of now, there exists no algorithm capable of computing the best solution within polynomial time. The brute force Dominating Set algorithm systematically computes every conceivable combination and yields the most favorable outcome. While this is viable for small-scale graphs, the computational demands become significant in our scenario, necessitating a considerable amount of time for computation.

Algorithm 4 Brute Force Dominating Set algorithm

```
all_vertices ← set of all vertices in graph
dominating_set ← None
min_dominating_size ← $\infty$
for $i$ from 0 to $2^{\text{len(all\_vertices)}} - 1$ do
    candidate_set ← {vertex | $(i \gg j)\&1$ for $j$, vertex in enumerate(all_vertices)}    ▷ Binary operator
    if IsDominatingSet(graph, candidate_set) and len(candidate_set) < min_dominating_size then
        dominating_set ← candidate_set
        min_dominating_size ← len(candidate_set)
    end if
end for
return dominating_set
```

Complexity of the Brute Force Dominating Set algorithm

This algorithm can be divided in two parts. First, the generation of all subsets. The loop goes from 0 to $2^{\text{len(all\_vertices)}}$ - 1 and for each iteration, it generates 2^n subsets of vertices, which each take $O(n)$ to construct. Therefore, the complexity this algorithm is $O(2^n \cdot n)$. The second calculation is checking if a subset is a dominating set. This function (3.1) has a complexity of $O(|n| + |m|)$ and since we have way more edges than nodes, it is $O(n)$.
Therefore, the overall time complexity is approximately $O(2^n \cdot n \cdot m)$.

2.4 Parameterized complexity

So far we have computed an approximation to the smallest dominating set. For general graphs, computing the smallest dominating set is well known to be NP complete, however, for certain classes of problems like covering/domination. Often these can be reduced to a kernel depending linearly on some parameter k. Thus the k-parameter allows us to have some control the complexity, but often at some cost of solution accuracy. This presents us with trade-offs worth exploring.

Fixed-Parameter Tractable formulations

In the case of problems of exponential complexity or higher, at the expense of some accuracy of our solution, the problem can be parameterized such that the exponential part of the complexity depends only on a *fixed parameter* k whilst the polynomial part only depends on the input size n of the problem.

Formally, a problem is said to be FPT in k if its complexity can be reduced to the following expression:

$$f(k) \cdot n^{\mathcal{O}(1)} \quad \text{for some arbitrary function } f$$

That is, the exponential part is controlled by the parameter k and the polynomial part by the input size n.

The problem of finding a smallest dominating set can be parameterized in multiple ways and there exists extensive research literature on the subject. In the following we will discuss two such parameterizations:

i) Finding Dominating set of size $|S| \leq k$ for degenerate graphs

In their 2008 paper[3] Noga Alon and Shai Gutner prove that the dominating set problem is fixed-parameter tractable for d-degenerate graphs. That is parameterizing the problem on the graph degeneracy d and a maximum size of the dominating set k. Arriving at the complexity $nk^{\mathcal{O}(dk)}$ linearly dependent on input size n.

A graph is said to be *d-degenerate* if every subgraph has a vertex of at most degree d. Recall that our graph is degenerate[4] with $d = 26$. Thus from Noga Alon and Shai Gutner we can fix the maximum size of the dominating set k we wish to find, and have a guaranteed worst-case running time of: $nk^{\mathcal{O}(26k)}$.

[3]https://arxiv.org/abs/0806.4735
[4]This was checked with library:

ii) Finding the k−dominating set

Recalling the definition of a dominating set, that is a subset $S \subseteq V$ such that all $v \in V$, either $v \in S$ or adjacent vertex u to v is in S.

We can 'relax' adjacency constraint, which requires that each dominated vertex is a direct neighbor to a vertex in S. Rather we can allow for a dominated vertex to be k distance from a dominating node in S.

Now, with respect to our problem, solving the k-dominating set for $k = 2$ corresponds to a doubling of the maximum allowed highway distance. The dominating set solution doesn't signify a coverage of 60km but rather extends to 120km.

When $k = 2$, the size of the dominating set decreases to 57, a tad smaller than half of the 119 achieved by our greedy algorithm for $k = 1$. It's crucial to note that both are essentially greedy algorithms, operating as approximations within a specific range contingent on the graph's unique properties. As k increases to 3, the size of the dominating set is 41 and for $k = 4$ the size is 26.

For graphs and the respective k-dominating sets visualized, see Appendix.

Minimizing the k-Dominating set
Though we have not implemented a minimizing method, we find the following noteworthy. In the paper [Nguyen, M.H., Hà, M.H., Nguyen, D.N.: Solving the k-dominating set problem on very large-scale networks][5] the authors took a *mixed integer linear programming* approach to solve the k dominated set problem. Their problem formulation looks as follows:

$$\begin{aligned} &\min \sum_{v \in V} z_v \\ st. \quad &\sum_{v \in \mathcal{N}(u,k)} z_v \geq 1, \quad \forall u \in V \\ &z_v \in \{0,1\}, \forall v \in V \end{aligned}$$

With set of vertices V, binary variable z_v and neighboring vertices u at k distance from v: $\mathcal{N}(u,k)$. However, solving this problem is considerably more involved than the simple problem formulation would indicate.
The authors go through 3 phases: *"pre-processing phase to reduce the graph size, construction phase to build a k-dominating set that will be reduced in the post-optimization phase by removing redundant vertices"*.

2.5 Run our Project

In our GitHub Repository at Janus124/Applied-Algorithm-Project, our code is housed. The main file, launch.py, orchestrates the execution of various algorithms and accommodates additional graphs specified by the path to a .dot file.

Simply launch **pip install requirements.txt**, then launch **./launch –help** for more information on how to launch the code.

[5] Comput Soc Netw 7, 4 (2020). https://doi.org/10.1186/s40649-020-00078-5

Algorithm	**France**	**Aquitaine**	**Gironde**	**Complexity**
Brute Force	-	-	5	$O(2^n \cdot n \cdot m)$
Greedy	117	18	6	$O(n^2)$
Benchmark	153	19	6	$O(n^2)$
K-Dominating Set = 2	41	9	4	$O(n^2)$
K-Dominating Set = 3	26	6	3	$O(n^2)$

Table 1: Size of dominating set result

3 Conclusion

In summary, our exploration into solving the Dominating Set problem for France's major highways has yielded both commendable results and highlighted significant challenges. The initial data simplification in Bordeaux paved the way for comprehensive coverage of the entire country. Despite challenges in mapping service stations and addressing junction complexities, our refined strategies showcased adaptability to large-scale network intricacies.

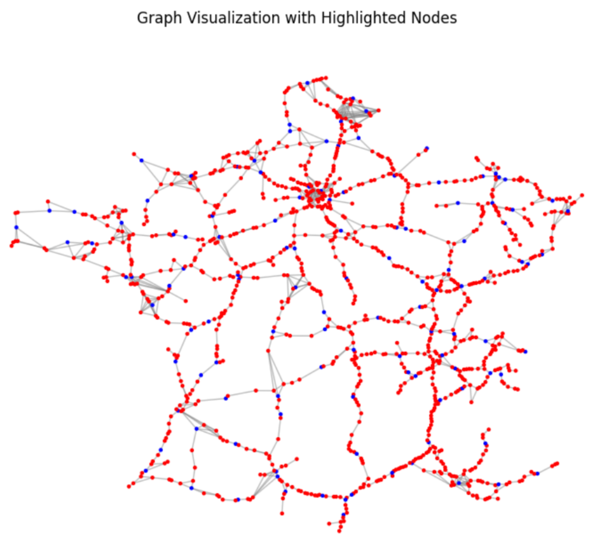
The integration of a clustering algorithm within a 5 km radius substantially improved graph manageability, emphasizing the importance of spatial optimization. The Floyd-Warshall algorithm played a crucial role in ensuring charging station proximity, resulting in an interconnected graph that facilitated effective analysis.

However, challenges persisted, notably in the junction removal process, emphasizing the need for nuanced strategies in handling complex spatial relationships.

Our dominating set solutions, achieved through a greedy algorithm and (theoretical) brute force, showcased notable results. The greedy algorithm outperformed networkx in the size of the dominating set, but the brute force, while ensuring optimal solutions, faced computational demands due to NP-complete complexity.

Finally, we explored approximations and parameterizations as well as possibilities for further research.

4 Appendix



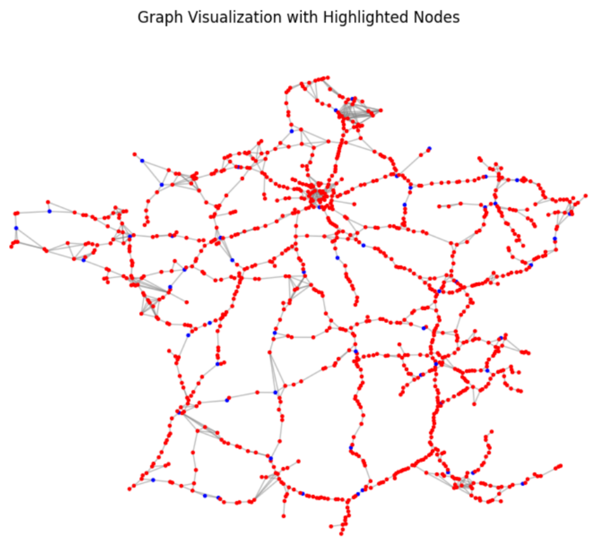
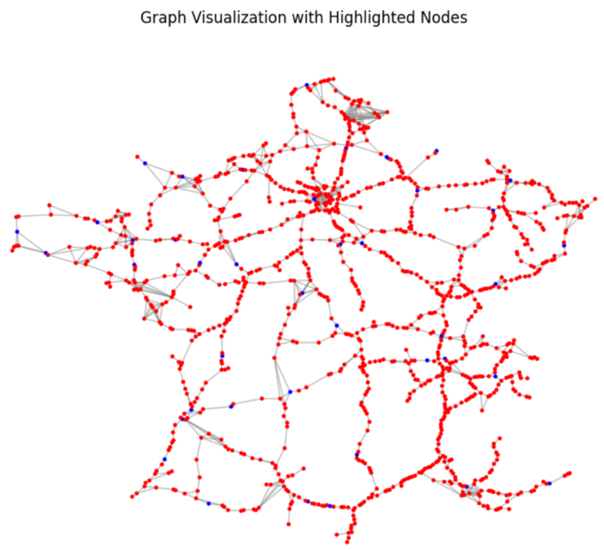


Figure 6: dominating set for $k = 1$, dominating set for $k = 2$



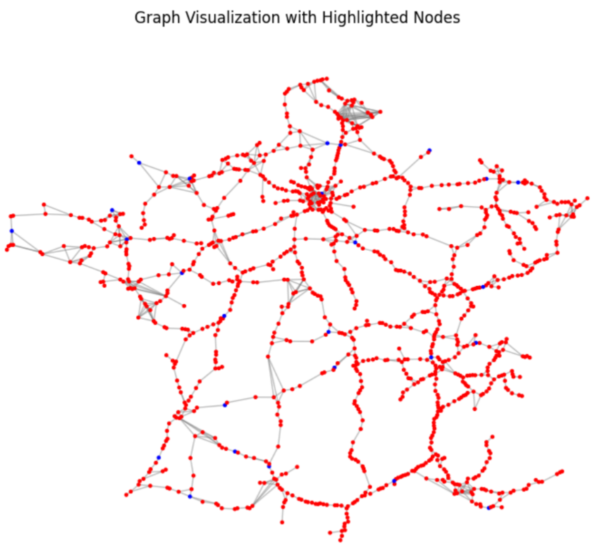


Figure 7: dominating set for $k = 2$, dominating set for $k = 3$