Den Bosch Garbage Collection Group Project Report

Mikołaj Hilgert & Edde Jansen

JBP031-B-6 Data-structures and Algorithms 2023

Contents

1 The Garbage Collection Problem

Clean streets and public spaces bring various benefit to communities, such as improved health and well-being [1]. In The Netherlands, it is considered the responsibility of the respective municipality to keep the streets clean for their citizens by periodically collecting the garbage [2]. It remains, however, a challenge for municipal administrations to find the best way of placing the bins, as placing too many bins on each street intersection might cause dissatisfaction among the garbage collectors.

This project will analyze the most efficient algorithm for placing garbage bins in Den Bosch, a city in the south of the Netherlands. The municipal administration of Den Bosch agreed with the garbage collectors that the city can place as many bins as possible, but at most one bin will be placed on adjacent intersections. Furthermore, it is known that on each intersection a maximum of four streets can meet. Being aware of the compromise and the city structure of Den Bosch, this project will examine the aforementioned problem by choosing an efficient and exact algorithm and analyzing its correctness and complexity.

2 Mathematical Context

The compromise of efficiently placing as many bins (k) as possible with only one bin being placed on adjacent intersections, resembles the mathematical challenge in **graph theory** of finding a **independent set** of size k of an **undirected graph**. In some cases, the k corresponds to the Maximum Indepedent Set (MIS). Table 1 denotes some of these key mathematical definitions as well as their relation to the context of the case study.

Table 1: Definitions of Graph Theory and the Maximum Independent Set (MIS) problem.

Definition	**Mathematical Definition [3]**	**Relation to Garbage Collection in Den Bosch**
Graph	A **graph** G can be represented as $G = (V, E)$ where V is the set of vertices of G, and the subset $E \subset V \times V$ is the set of edges of G.	The City of Den Bosch (G) = (Intersections V, Streets E)
Undirected Graph	An **undirected graph** $G = (V, E)$ where the edge set E consists of *unordered* pairs of vertices, i.e. (u, v) is the same edge as (v, u).	Den Bosch in which each street is a two-way street.
Adjacent Vertices	If $(u, v) \in E$ then vertex u is **adjacent** to vertex v and the two vertices are thus not joined by an edge.	Intersections joined by a street.
Independent Set	In a graph $G = (V, E)$, a set of vertices $S \in V$ is **independent** if there are no two adjacent vertices in S.	A set of garbage bins that are placed in such a way that there are no adjacent intersections.
The Maximum Independent Set (MIS) problem	The challenge to find a **maximum-sized independent set** in G.	Placing a set of as many non-adjacent garbage bins as possible.

There also applies a domain constraint in the city of Den Bosch, which will be referred to as *Constraint A*. This is a constraint on the number of intersections: $|V|$, number of streets: $|E|$ and number of ordered bins k in a city graph G. The domains are as follows (retrieved from the given project description document *):

$$(1 \leq |V| \leq 20,\ 1 \leq |E| \leq 100,\ 1 \leq k \leq 20).\ *$$

As Table 1 highlights, the challenge of placing as many bins as possible with no adjacent intersection can be an equivalent of the MIS problem. The MIS problem becomes more clear when visualized in a graph representation, as can be seen on the next page. Figure 1a demonstrates an undirected graph with 5 vertices and 7 edges, whereas Figure 1b and Figure 1c represent this through an adjacency matrix and adjacency list respectively [3]. Whereas each vertex in Figure 1a could be considered on its own an independent set, the graph represents three maximum independent sets, namely $S = \{1, 3\} \vee \{1, 4\} \vee \{3, 5\}$.

Hence, for the design of an algorithm that places as many non-adjacent garbage bins as possible, the size of garbage bins k should be smaller or equal to the maximum independent set. In other words, if $k \leq |S|$, the algorithm should return *possible*, and if $k > |S|$, the algorithm should return *impossible*. After discussing the most appropriate data structure for the problem of placing garbage bins, the following will discuss the various algorithms that can be used to find a solution for the MIS problem, from which eventually one algorithm - one that is exact and as fast as possible - will be chosen and further examined analyzing its correctness and complexity.

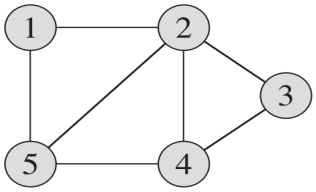

Figure 1a: Example of an undirected graph G.

	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

Figure 1b: The adjacency matrix representation of G.

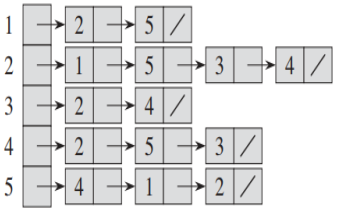

Figure 1c: The adjacency list representation of G.

3 Data Structure

As previously demonstrated, graphs can be represented both in an adjacency matrix (Figure 1b) and adjacency list (Figure 1c). When working with graphs, it is vital to select an appropriate data structure as it depends on the context which data structure is more suitable. For the particular context of Den Bosch and the fact that intersections have at maximum 4 streets, it can be derived that the graphs are rather sparse. That is, there is a relatively small number of edges compared to the maximum number of edges a graph with n vertices could consist of (i.e. V^2). To illustrate this argument, Table 2 compares the two graph representations in terms of their complexity.

Table 2: Time and Space Comparison for Adjacency List and Adjacency Matrix.

Property	**Adjacency List**	**Adjacency Matrix**
Space	$\theta(V+E)$	$\theta(V^2)$
Time (to list all vertices adjacent to u)	$\theta(\text{degree}(u))$	$\theta(V)$
Time (to check if $(u,v) \in E$)	$\theta(\text{degree}(u))$	$\theta(1)$

Given the maximum of 4 streets at each intersection in an urban context, and Table 2 above, the samples are relatively sparse and for a sparse graph, an adjacency list is a more sensible data structure. This is because all the 0's within a large adjacency matrix would cause a large memory complexity, whereas the complexity for adjacency lists is reduced from $\theta(V^2)$ to $\theta(V+E)$. Furthermore, with a maximum degree of 4 for each intersection, running time cannot be expected to be a major issue as sets are used to store the neighbors. Hence, this study will use an adjacency list as its data structure for the MIS problem of placing garbage bins in the city of Den Bosch.

4 Chosen Algorithm

This study compares three main algorithms for the MIS problem that the city of Den Bosch faces, namely Brute Force, Greedy and Backtracking. Table 3 explain how they operate as well as provides an indication of their correctness and complexity (worst case running time, noted in Big-O).

Table 3: Three algorithms for the MIS problem and their correctness & complexity, where $n = |V|$

Algorithm	**Definition [4]**	**Exactness**	**Time Complexity**
Brute Force	Iterates on every possibility available to solve the problem and checks whether a candidate meets the problem statement.	Always	$O(2^n * n)$
Greedy	Solves the problem part by part, by always choosing the next candidate that is the local optimal choice (with the highest benefit).	Not always due to locally optimal choices	$O(n^2)$
Backtracking	Solves problems recursively by incrementally building candidates to a solution, abandoning a candidate as soon as it determines that the candidate cannot possibly be completed to a valid solution.	Always	$O(2^n)$

Having defined and chosen these three algorithms for the sake of comparing and justifying the most appropriate one, this study started with the **Brute Force algorithm**. This algorithm was chosen first as it is known for its correctness, given that it will try all possible combinations of subsets for a certain problem (Appendix A.1). However, upon trying larger samples, it took an extremely long computing time. This can be explained by its long running time, noted with its exponential time complexity of $O(2^n * n)$ (Appendix A.2). As a result, another algorithm known for its fast running time was tried, the Greedy algorithm.

Though the use of the **Greedy algorithm** had a significantly faster computation ($O(n^2)$) (Appendix B.1), it could not be proven to be correct, as the Greedy algorithm is one of approximation rather than exactness. This becomes visible through the fact that it looks for *locally* optimal solutions, whereas this may not be the optimal solution *globally*. Appendix B.2 contains an extensive proof of why the Greedy algorithm would be inappropriate for the specific context of identifying an independent set of garbage bins that is *exactly* size k.

Given that the Brute Force and Greedy algorithms were deemed inappropriate for the foregoing reasons, a third algorithm was evaluated: the **Backtracking algorithm**. Due to its correctness as well as its relatively faster running time (because it will explore all possibilities in a depth first manner, as opposed to iterating every available possibility *first* as in Brute Force), this algorithm was deemed the most appropriate for the problem of efficiently placing garbage bins in Den Bosch. Algorithm 1 below shows the pseudocode of the main algorithm and a respective Algorithm 2 based on the Backtracking Algorithm that will find a MIS.

Note: For the Algorithms below, graph structure G contains a graph of (V, E), an adjacency list based on (V, E) and target integer k that represents the number of bins to be placed.

Algorithm 1 Validate if k bins can be placed

Require: A graph structure: G
Ensure: An independent set S of size k, if one exists

```
current_set ← ∅
if BACKTRACK(0, current_set, G) then
    output "possible"
else
    output "impossible"
end if
```

Algorithm 2 Backtrack (Find Maximum Independent Set in G)

Require: Current vertex: *vertex*, Current independent set: *current_set*, A graph structure: G
Ensure: Return True if there exists an independent set larger or equal than size k in G, otherwise False

```
if vertex = n then                                   ▷ Base case: if all vertices have been considered
    if |current_set| ≥ k then
        return True
    else
        return False
    end if
end if
if vertex does not have any adjacent vertices in current_set then
    Add vertex to current_set
    if Backtrack(vertex + 1, current_set, G) then    ▷ Include the vertex in current_set
        return True
    end if
    Remove vertex from current_set
end if
if Backtrack(vertex + 1, current_set, G) then        ▷ Don't include the vertex in current_set
    return True
end if
return False
```

Our **'Validate if k bins can be placed'** algorithm first requires the input of a graph G with the number of nodes, edges and bins respectively, as well as lines below representing two vertices (intersections) that form together each edge (street) in Den Bosch. Before the backtracking function will be called, it sets the current MIS set *current_set* to $\emptyset$. After calling the function, if an independent set exists of size k, the algorithm will return *'possible'*, and if it not exists, it will return *'impossible'*.

Regarding Algorithm 2 used in Algorithm 1, the **Backtrack (Find Maximum Independent Set in G)** algorithm will first check if all the vertices have been considered, to ensure the function reaches its base case. If this is the case, it checks if the *current_set* equals or is larger than k bins and returns True if this is the case (line 2-3). If it has not reached the base case, then it will check if the current vertex has any adjacent intersections that are in the current independent set (line 8). If this is *not* the case, it will add the vertex to the current set (line 9) and call the algorithm again (line 10-11) with the new vertex included, thereby looking for next vertices that do not have any adjacent vertices in the current set. If this current path with this vertex included does not reach to finding an MIS, then, it backtracks and the vertex is removed from the current set (line 13) and calls the function again, yet now without this particular vertex (line 15-16). This ensures that all combinations are tried, as both cases are considered for each vertex. Having justified the most appropriate data structure and algorithm as well as defined our algorithm for the present study, the following section will be dedicated towards further elucidating our algorithm, by analyzing its correctness and run time complexity.

5 Algorithm Analysis

5.1 Algorithm Correctness

This section will examine the correctness of our algorithm. The proof of correctness is essential for this algorithm, because as discussed previously, the objective of the present study is to find an exact algorithm, which by definition will always return an independent set of size k if and only if such a set exists in *any* graph G. It is important to note that **the proof below will be regarding the** `Algorithm 2`, as Algorithm 1, only trivially prints 'possible' or 'impossible' based on the result of the `Algorithm 2`.

For this function, we will prove the following two main properties:

- If an independent set of size k exists in the graph, the algorithm will successfully identify it.
- If no independent set of size k exists in the graph, the algorithm will exhaustively explore all possible combinations of vertices.

These two properties correspond to the *exactness* (every solution found is valid) and *completeness* (every valid solution can be found) of the function.

Lemma 1: If an independent set of size k exists in the graph, the function will successfully identify it.

We claim that for any value of $k \geq 0$ the function correctly explores all subsets of graph G = (V,E) and correctly determines whether an independent set of size k exists. Moreover, if that is the case, it will return True, otherwise False.

Proof: We will prove this by strong induction:

Base Case: k = 0

In this case, the function will trivially return True as any $|current\_set| \geq k$, since $k = 0$.

Induction Hypothesis (IH): Lets define the hypothesis P(i) as: For all graphs G with n vertices and a target k, if an independent set of size k exists, the function correctly determines it for all $i \leq k$.

Inductive Step: Assume P(j) is True for all j, $0 \leq j < i$. We must prove P(i) is True.

In the function, we have two possible scenarios for the vertex 'v' (called *vertex* in line 9):

`Case 1`: The vertex v is included in the independent set. The function adds vertex v to the current set. This reduces the problem to finding an independent set of size $i - 1$ in the remaining graph, which, by our IH, the function will correctly do if such a set exists.

`Case 2`: The vertex v is not included in the independent set. The function skips vertex v and moves on to the next vertex. This reduces the problem to finding an independent set of size i in the remaining graph, which, by our IH, the algorithm will correctly do if such a set exists.

Thus, in both cases, the algorithm works as expected, thereby proving that P(i) holds for i when it holds for all j and will subsequently correctly return True or False depending if such an independent set that is larger or equal to k exists. As determined when vertex $v = n$ (lines 1-6).

Lemma 2: If no independent set of size k exists in the graph, the function will exhaustively explore all possible combinations of vertices.

Proof: We will prove this by contradiction:

Assume that there is a graph G = (V, E) that does not have an independent set of size k, and assume that the algorithm does not try all combinations of vertices. This means that there exists a subset of size k or larger that is not found, lets call it set L.

We have two possibilities, whether L is an independent set or not.

If L is an independent set of size k, this contradicts our initial assumption that no independent set of size k exists in G. Thus, this leads to a contradiction.

If L is not an independent set of size k, the function correctly did not return True, as the requirement of the size k was not met. Therefore, the function works correctly. However, we made the assumption that the function did not try all combinations, this implies that there is an a combination that was not attempted. This however, contradicts *Lemma 1.* As such, we again have a contradiction.

Hence, in both cases, we reach a contradiction. Therefore, our initial assumption that our function will not try all combinations of vertices is incorrect. Consequently, if no independent set of size k exists in the graph, the function will indeed exhaustively explore all possible combinations before returning False.

To conclude this algorithm correctness analysis, a combination of **Lemmas 1 and 2** provides a rigorous demonstration of the accuracy and correctness of this exact backtracking function that works for any graph.

5.2 Algorithm Running time

In order to analyse the running time, one needs to identify what type of algorithm it is. Given that at line 11 and 16, the function will call itself, it means that this is a recursive function. As such, to properly evaluate its running time, one must construct a recurrence relation. This can be done by analysing the cost associated with the execution of a single recursive call as well as determine how many times the function can call itself in the worst case. Firstly, we can analyse analyse the cost elementary operations within the function body:

- **Line 2**: Checking if all vertices have been considered. This operation takes $O(1)$ time.
- **Line 3**: Checking if the size of the current set is $\geq k$. This operation takes $O(1)$ time.
- **Line 9**: Checking if the vertex does not have any adjacent vertices in the current set. This operation takes $O(1)$ time, a vertex can have a maximum degree of 4, we can argue that this takes constant time as it does not grow with the input size n.
- **Line 10**: Adding a vertex to the current set. This operation takes $O(1)$ time.
- **Line 11 & 16**: Recursive call to the Backtrack function. ▷ This is evaluated later.
- **Line 14**: Removing a vertex from the current set. This operation takes $O(1)$ time.

Given that we have a worst case where the graph has no edges, and we're given a k that is n+1 (which is impossible), this ensures that every vertex has to be considered and that each vertex has to be both included and not included in the 'current set', As in this case we will always satisfy the condition at line 9. Consequently, this means we will always have to make two recursive calls for each vertex ($O(n-1) + O(n-1)$), which results in a recurrence relation of $T(n) = 2T(n-1) + 1$. The +1 represents the cost per recursive call, which we concluded in the above section is all in constant time. Thus, to deduce the total worst case running time, we can solve the recurrence $2T(n-1) + 1$ using the substitution method:

$$T(n) = \begin{cases} 1 & n = 0 \\ 2T(n-1) + 1 & n > 0 \end{cases} \tag{1}$$

Note: The left section precedes the right in the process below

$$\begin{aligned}
T(n) &= 2T(n-1) + 1 \\
T(n) &= 2[2T(n-2) + 1] + 1 \\
T(n) &= 2^2T(n-2) + 2 + 1 \\
T(n) &= 2^2[2T(n-3) + 1] + 2 + 1 \\
T(n) &= 2^3T(n-3) + 2^2 + 2 + 1 \\
&\vdots \\
T(n) &= 2^iT(n-i) + 2^{i-1} + 2^{i-2} + \cdots + 2^2 + 2 + 1
\end{aligned}$$

$$\begin{aligned}
&\text{Assume } n - i = 0, \text{ so } n = i \\
&\quad T(n) = 2^nT(0) + 1 + 2 + 2^2 + \cdots + 2^{i-1} \\
&\quad\quad = 2^n * 1 + 2^i - 1 \\
&\quad\quad = 2^n + 2^n - 1 \\
&\quad T(n) = 2^{n+1} - 1 \\
&\quad\quad \therefore O(2^n)
\end{aligned}$$

Hence, using the substitution method, the foregoing analysis reveals that our algorithm for finding a maximum independent set in the city of Den Bosch has a worst-case running time of $O(2^n)$.

6 Related Work

The MIS problem presented in the present study is a popular phenomenon in computational complexity theory. Multiple studies indicate that this problem is NP (Non-Deterministic Polynomial) hard, meaning that solutions can be found by exhaustive search, yet there is no solution solving it in polynomial time eg, (n^2) [5–7]. The first known design of an exact algorithm solving the MIS problem dates back to 1976, with a time complexity of $O(2^{(n/3)})$ [8]. Ever since, faster algorithms have been on the rise, with in particular significant contributions to algorithms for degree-bounded graphs, such as an algorithm that finds MIS in a graph with maximum 3 degrees (MIS-3) in $O(1.0836^n)$ [9]. These contributions have also enabled solving MIS-4 - similar to the present study - in $O(1.1376^n)$ and MIS-5, MIS-6, and MIS-7 in $O(1.1737^n)$, $O(1.1893^n)$, and $O(1.1970^n)$ respectively [7].

Despite these contributions that have found faster exact algorithms for the MIS problem and despite having reviewed them prior to the algorithmic analysis, the present study has designed (an arguably more naive) algorithm for various reasons. First, it may happen that the design of the city of Den Bosch changes to one where there are more then 4 streets at each intersection, thereby rendering such a degree-bounded algorithm inappropriate. Second, the present study proposed not only an algorithm that is correct with a favourable running time, yet also one that is relatively easy to understand, test, and implement by the municipal administration of Den Bosch, to prevent the utilization of an algorithm that is not explainable to and understood by those that implement it. Third, the backtracking algorithm proposed in the present study may also be auspicious to the application to other contexts for Den Bosch, such as the placement of other municipal facilities.

7 Conclusion & Discussion

To conclude, in order to satisfy the garbage collectors while also correctly placing a maximum amount of bins, this study proposes a backtracking algorithm for the municipality of Den Bosch in their process of effectively placing their garbage bins. As this project examined, the backtracking algorithm was deemed the most appropriate, as it compared to a Brute Force and Greedy algorithm had a more beneficial running time and found a globally correct solution, compared to one that approximates by making locally optimal choices, such as the Greedy. Moreover, compared to the Brute Force algorithm, the proposed algorithm also does not require the computation of the power set $P(V)$ ahead of time, meaning that in most cases if a set of size k exists, it will find it much faster. After analyzing its correctness and complexity, samples were tested with our algorithm, yielding an accuracy metric of 0.90 (Appendix C). Despite a correct classification for the majority of the samples, it failed to quickly demonstrate that the large impossible test samples (Appendix C) were indeed impossible due to the $O(2^n)$ time complexity of our algorithm.

Dealing with time complexity in certain algorithmic problems is a challenge that humankind will most likely keep facing in the foreseeable future. Although faster computers can bring us closer to get a correct solution within less running time, it is the time complexity of the algorithm that determines the actual efficiency [3]. Future research or adaptations from the algorithm proposed here might benefit from this reality. For instance, the backtracking algorithm could be enhanced by integrating a pruning element. In light of the present study, pruning would mean that the algorithm quits adding vertices when the number of bins could no longer be added to a subset, as the number of needed bins as indicated through the input > the number of vertices that still needs to be searched (i.e. the algorithm could terminate quickly if the municipality wants to place 15 bins, the algorithm found 8 vertices in one subset, while it still needs to iterate over 5 more vertices ($15 > 13$)). Another thing that might be promising for the present study would be the utilization of a hybrid algorithm approach, which would first utilize a Greedy algorithm ($O(n^2)$, Appendix B) and would only employ the Backtracking algorithm once the Greedy algorithm would return 'impossible'.

These could be potential approaches to solving a NP-hard problem like the MIS problem in the present study, or for other well-known NP hard problems, such as the well-known minimum vertex cover problem (MVCP) and travelling salesman problem (TSP) [5, 10]. If such an approach, however, is not possible or deemed inappropriate, the proposed backtracking algorithm has shown to be felicitous for the MIS problem. The given algorithm could additionally be a promising algorithm for other municipal challenges that would require an exact solution for the MVCP or TSP, such as placing a minimal amount of municipal facilities due to financial limitations or designing a shortest possible route for garbage collectors after the garbage bins have been placed. Similar to the present study, it remains however paramount that in future research and applications of these algorithms to the municipality of Den Bosch, one carefully considers the municipal context including its different key stakeholders, such as the municipal administration, those that work in the municipality, as well as its population.

References

[1] Ferronato, N., Portugal Alarcón, G. P., Guisbert Lizarazu, E. G., &; Torretta, V. (2021). Assessment of Municipal Solid Waste Collection in Bolivia: Perspectives for avoiding uncontrolled disposal and boosting waste recycling options. *Resources, Conservation and Recycling*, 167, 105234.

[2] Larper, C. (2023, March 21). *Garbage collection and recycling in the Netherlands.* Expatica. https://www.expatica.com/nl/living/household/recycling-in-the-netherlands-133948/

[3] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to algorithms.* Mit Press.

[4] Dobhal, R. (2022, November 23). *Discover 8 Types of Algorithms for Efficient Problem Solving.* Coding Hero. https://codinghero.ai/discover-8-types-of-algorithms-for-efficient-problem-solving/

[5] Woeginger, G. J. (2003). Exact Algorithms for NP-Hard Problems: A Survey. In: Juenger, M., Reinelt, G., & Rinaldi, G. (2003). *Combinatorial optimization.* Springer, 185-207.

[6] Liu, Y., Lu, J., Yang, H., Xiao, X., & Wei, Z. (2015). Towards maximum independent sets on massive graphs. *Proceedings of the VLDB Endowment*, 8(13), 2122–2133.

[7] Xiao, M., & Nagamochi, H. (2017). Exact algorithms for maximum independent set. *Information and Computation*, 255, 126–146.

[8] Tarjan, R. E., & Trojanowski, A. E. (1976). Finding a maximum independent set. *Computer Science Department, School of Humanities and Sciences, Standford University*, 1-22.

[9] Xiao, M., & Nagamochi, H. (2013). Confining sets and avoiding bottleneck cases: A simple maximum independent set algorithm in degree-3 graphs. *Theoretical Computer Science*, 469, 92–104.

[10] Bhargava, A. Y. (2016). *Grokking Algorithms: An illustrated guide for programmers and other curious people.* Manning.

[11] Toida, S. (n.d.). *Size of Powerset.* https://www.cs.odu.edu/ toida/nerzic/content/induction/example6/example6.html

A The Brute Force Algorithm

This appendix highlights the correctness and complexity of the Brute Force algorithm. This approach will find and evaluate all possible subsets of size k and evaluate whether any of those subsets are a valid independent set [4].

Algorithm 3 Brute Force Independent Set Algorithm

Require: A graph $G = (V, E)$ and an integer k
Ensure: An independent set S of size k, if one exists

```
1:  for all subsets S ⊆ V do
2:      isIndependent ← True
3:      for all u, v ∈ S with u ≠ v do
4:          if (u, v) ∈ E then
5:              isIndependent ← False
6:              break
7:          end if
8:      end for
9:      if isIndependent and |S| = k then
10:         output "possible"
11:     end if
12: end for
13: if not isIndependent then
14:     output "impossible"
15: end if
```

A.1 Brute Force Correctness Proof

The brute force algorithm is an exact algorithm by definition, as it exhaustively will compute and try every possible subset of the power set $P(V)$, The power set of a set S is the set of all possible subsets of S, including the empty set and S itself. As a result, the algorithm can guarantee that it will explore all possible subsets (S) and definitively check if an independent set of size k can exist or not for the given graph G. This is because per every pair of vertices in S, we must check if an edge exists between them in set E [6].

A.2 Brute Force Complexity Analysis

As mentioned in the correctness proof, this algorithm will exhaustively search every possible subset combination. This requires the computation of the power set. If S is a finite set with n elements, then P(S) has 2^n distinct subsets, as proven by Shunichi Toida [11]. This means that in the worst case, we will have to execute lines 2-11 2^n times, which itself has the run-time complexity of O(n). As such, the overall time complexity of this algorithm is O($2^n * n$).

B The Greedy Algorithm

This section highlights the complexity and correctness of the Greedy Algorithm. The greedy approach is often fast and efficient because it makes locally optimal choices without considering the long-term consequences. Since it doesn't need to examine all possible solutions, it can often run faster than other algorithms that are more exhaustive in their search.

Algorithm 4 Greedy Independent Set Algorithm

Require: A graph $G = (V, E)$ and an integer k
Ensure: An independent set S of size k, if one exists
1: $S \leftarrow \emptyset$ (Indepedent Set)
2: $C \leftarrow V$ (Candidate Set)
3: **while** $|S| < k$ and $C \neq \emptyset$ **do**
4: Select a vertex $v \in C$ with the minimum degree in subgraph $G[C]$
5: **if** v is non-adjacent to all vertices in S **then**
6: Add v to S
7: Remove v and its neighbors from C
8: **else**
9: Remove v from C
10: **end if**
11: **end while**
12: **if** $|S| = k$ **then**
13: **output** "possible"
14: **else**
15: **output** "impossible"
16: **end if**

B.1 Greedy Complexity Analysis

We can evaluate the worst case running time of this algorithm by counting all the time steps taken within the pseudo-code of **Algorithm 1** :

1. Line 1: Initializing empty set S (independent set) takes O(1) time.
2. Line 2: Initializing C (candidate set) takes O(n) time..
3. Lines 3-11: The while loop runs until either $|S|$ reaches k or C becomes empty. In the worst case, this loop can run n times.
4. Line 4: Selecting a vertex with the minimum degree in subgraph G[C] can take O(n) time in the worst case.
5. Lines 5-10: Line 5 involves a case distinction for the if statement. The if statement itself can take O(n) time.
   (a) Line 6: Adding a vertex to S takes O(1) time.
   Line 7: In the worst case, removing a vertex and its neighbors from C takes O(1) time, as degree is maximum 4.
   (b) Line 8: Removing a vertex from C takes O(1) time.

   At worst case, we will execute case 'a' every time, which takes a summed O(n) time.
6. Lines 12-16: Comparing $|S|$ to k and outputting the result takes O(1) time.

As mentioned, if we consider the worst-case scenario, the while loop (lines 3-11) runs n times with each iteration taking O(n) time. Thus, the **total** run-time complexity in the worst case of this algorithm is:

$$f(n) = (n * (n + n)) + n + 1 + 1 = 2n^2 + n + 2$$
$$= O(n^2)$$

Note: The running time could even be lowered to O(n * log(n)), if a min-heap data-structure is used to build the candidate set, this allows for log(n) time to pop the next vertex to try, as opposed having to search for the vertex with the next lowest degree.

That's great, a polynomial time solution is very fast and efficient! It's important to note however, that the greedy approach is one of approximation, rather than one of exactness. This can be explained by the algorithm making

locally optimal choices (Line 4 in **Algorithm 1**, selecting a vertex with minimum degree). Which may not actually be the *globally* optimal solution. Meaning, the greedy approach could lead to the selection of incorrect vertices. As such, we will attempt to prove the correctness of this algorithm for our goal.

B.2 Greedy Correctness Proof

Claim: While the greedy approach can give us an result in polynomial type, it may not always provide the exact best solution. In our context, an Independent set of exactly size k.

Suppose $G = (V, E)$ is a graph, and let k be a positive integer. An independent set of size k is a subset $S \subseteq V$ such that $|S| = k$ and no two vertices in S are adjacent.

Consider the graph $G = (V, E)$ with $V = \{0, 1, 2, 3, 4, 5\}$ and $E = \{(0, 3), (0, 5), (3, 1), (3, 2), (5, 1), (5, 2), (1, 2),$

$(1, 4), (2, 4)\}$. Let $k = 3$.

We **assume** that G has an independent set of size k, which is $\{3, 4, 5\}$. This is true because $\{3, 4, 5\}$ is a subset of V and $|\{3, 4, 5\}| = 3$ and more importantly, by the definition of independent set these three vertices share *no* edge.

Now consider running the greedy algorithm to find an independent set of size $k = 3$ in G. Consider an arbitrary execution of the algorithm:

Firstly, the algorithm selects vertex 0 from C as it has the smallest degree $|0| = 2$. Since the set S is currently empty, vertex 0 is added to S, and adjacent vertices 3 and 5 are removed from C. In the next iteration, the algorithm will select vertex 4 as it has the smallest degree $|4| = 2$. Then, checking if vertex 4 is adjacent to any vertex in S. Since vertex 4 shares no edge with vertex 0, $S = \{0,4\}$. Consequently, the adjacent vertices to vertex 4, vertices 1 and 2 are removed from C. At this point, $|S| = 2$ and $C = \emptyset$, as such there are no more candidate vertices to consider and the algorithm terminates without finding an independent set of size $k = 3$.

Hence, we can establish by means of a **proof by contradiction** using a counterexample that the greedy algorithm is not appropriate for identifying an independent set of *exactly* size k.

C Test Samples & Results

This appendix provides an overview of the test samples that have been used in the present study to find a maximum independent set. The samples consist of big, small and extra samples and are supposed to resemble this project's study context, namely the city of Den Bosch. Table 4 below displays the edges, vertices, number of bins that are tested and the graph density. The **graph density** for the undirected graphs has been calculated using the following formula:

$$D = \frac{2 * |E|}{|V| * (|V| - 1)}$$

in which D is the Graph Density, $|E|$ is the number of Edges and $|V|$ the number of Vertices in the graph.

Table 4 below also shows that the output from our algorithm is correct for all of the samples, except for two big 'impossible' samples (Big 4 and Big 6) where the output is undefined due to the extremely long running time. This gives us an accuracy of:

$$\frac{|\text{Correctly Classified Samples}|}{|Samples|} = \frac{18}{20} = \mathbf{0.90}$$

Table 4: Overview of used test samples and their density, with results of the backtracking algorithm

Sample Name	**Edges**	**Vertices**	**Number of bins**	**Graph Density**	**Expected Output**	**Actual Output**
Big 1	90	45	15	0.091	Possible	Possible
Big 2	90	45	15	0.091	Possible	Possible
Big 3	125	72	18	0.049	Possible	Possible
Big 4	125	72	19	0.049	Impossible	—
Big 5	180	90	18	0.045	Possible	Possible
Big 6	180	90	19	0.045	Impossible	—
Small 1	10	7	3	0.476	Possible	Possible
Small 2	10	7	4	0.476	Impossible	Impossible
Small 3	16	9	2	0.444	Possible	Possible
Small 4	16	9	3	0.444	Possible	Possible
Small 5	16	9	4	0.444	Impossible	Impossible
Small 6	4	5	3	0.4	Possible	Possible
Extra 1	13	8	3	0.464	Possible	Possible
Extra 2	77	39	15	0.104	Possible	Possible
Extra 3	36	18	6	0.235	Possible	Possible
Extra 4	9	6	3	0.6	Possible	Possible
Extra 5	30	19	10	0.175	Possible	Possible
Extra 6	30	19	11	0.175	Impossible	Impossible
Extra 7	22	13	6	0.282	Possible	Possible
Extra 8	22	13	7	0.282	Impossible	Impossible
Average	55.05	29.75	—	0.27		
Minimum	4	5	2	0.045		
Maximum	180	90	19	0.6		

Note: For all cases above without '-' the computation was in the milliseconds.