Failure detection with distance sensors

Graphs and discrete structures project

B. DEROO, A. HEAM, T. TERRIEN

December 16, 2022

1 Introduction

1.1 Definition

"Consider a graph G and a set S of vertices equipped with "distance sensors" (that is, the pairwise distances between the vertices of S in G can be determined at all times by the sensors). We say that S can detect edge-failures if for any edge e, the removal of e in G changes the distance between ***at least*** *one pair of vertices of S. What is the minimum size of a set of vertices that can detect edge-failures ?"*

1.2 Application

Let us imagine a pipe network. It could be represented as a graph, where edges correspond to pipes, and vertices correspond to pipe's intersections. The degree of a vertex would then be the number of pipes intersecting at that location.
It could be interesting to equip some intersections with sensors so that if a problem occurs (for example a change of flow rate, due to a leak), we know easily which pipes can have defects and thus take action quicker and more efficiently. Solving our problem would give the minimal number of sensors needed to be able to detect all possible leakages.

1.3 Notations

For every section, we will use the following notations. $G = (V, E)$ is a graph whose vertices are the elements of V and edges are the elements of E. $|V|$ can be denoted as n and $|E|$ can be denoted as m. We only consider graphs where $n \geq 2$ since otherwise, E would be empty which is not interesting.

S represents the subset of vertices of V that are equipped with distance sensors.

We denote by S_G the set of all possible sets of minimal size that can detect edge-failure in G. We denote as n_G the size of one element of S_G.

We call *solution* a subset $S \subseteq V$ that detects edge-failure in G. We call *optimal solution* such a S of minimal size (it is thus an element of S_G). We say that an edge is *detected* if its removal changes the distance between two vertices of S (i.e. the number of edges between those two vertices, we don't consider weighted edges here).

Lemma 1.1. *Let $G = (V, E)$. Let $S \subseteq V$ be the set of vertices equipped with distance sensors. A pair of vertices (u, v) of S detects at most one uv-path.*

Proof. If there is no uv-path, no path is detected. If there are many uv-paths of different sizes, only the shortest path is detected (cutting an edge of a longer uv-path will not change the distance between

u and v). If there are multiple shortest uv-paths, none of them are detected (cutting an edge in one of them would not change the distance between u and v). □

Corollary 1.1.1. *In any graph G, we have:*

$$S \text{ detects edge-failure in } G \Leftrightarrow \text{every edge is in a unique shortest } uv\text{-path } (u, v \in S)$$

Proof. Let's suppose, that $\forall u, v \in S$, $\exists e \in E$ s.t. there is at least one shortest path p between u and v that doesn't contain e. Let such e, u, v, p exist. If e gets cut, p is unaffected, thus the distance between u and v is still $|p|$. So S doesn't detect edge-failure. This gives the direct sense.
Let's suppose, that $\forall e \in E$, $\exists u, v \in S$ s.t. there is a unique shortest path, p, between u and w and it contains e. Let such e, u, v, p exist. If e gets cut, then p isn't feasible anymore, by uniqueness of the shortest path, thus the remaining paths between u and v are all strictly longer than p, which means the distance between u and v is strictly higher than prior to the cut. Hence S detects edge-failure in G. This gives the indirect sense. □

2 Bounds

2.1 Bounds on basic graphs

In this section, we are going to prove some bounds on basic graphs.

Universal bounds: $2 \leq n_G \leq n$

Proof. Let $S \in S_G$, since $S \subseteq V$, then $|S| \leq |V|$. Hence $n_G \leq n$.
Let $S \subset V$, if $|S| = 0$, then there is no sensor and no edge is detected. Likewise, if $|S| = 1$, then there is only one vertex in S, so there is no pairwise distance, hence $n_G \geq 2$.
These results prove that for any graph G, S_G and n_G exist. □

Paths: $n_G = 2$ and $|S_G| = 1$

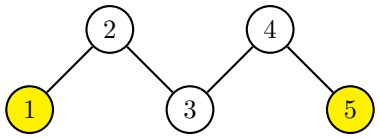


Figure 1: Example of 5-vertices path. $S_G = \{\{1, 5\}\}$ and $n_G = 2$.

Proof. Let G be a path, then there exist k vertices $u_1 \dots u_k$, such that $V = \{u_1, \dots, u_k\}$ and $E = \{(u_1, u_2), \dots, (u_{k-1}, u_k)\}$. Let's prove that $S_G = \{\{u_1, u_k\}\}$.
First, let's prove that $\{u_1, u_k\} \in S_G$. Let $e = (u_i, u_{i+1}) \in E$. Since G is a path, e is in the sole path between u_1 and u_k. So if it gets cut, the distance between u_1 and u_k goes from k to $+\infty$, thus $\{u_1, u_n\}$ detects edge failure. $\{u_1, u_k\}$ is of minimal size, 2, so $\{u_1, u_k\} \in S_G$ and $n_G = 2$.
Now, let's prove that $S_G \subset \{\{u_1, u_k\}\}$. Let $\{u_i, u_j\} \in S_G$; by construction of a path, the only edges detected are the (u_k, u_{k+1}) for $k \in \{i, j-1\}$. Hence, since (u_1, u_2) and (u_{k-1}, u_k) are detected, $i = 1$ and $j = k$, respectively. Hence $S_G \subset \{\{u_1, u_k\}\}$.
Hence the equality, and $|S_G| = 1$. □

.

Complete graphs: $n_G = n$

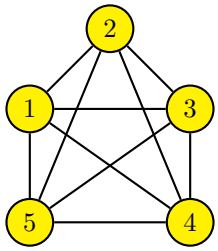


Figure 2: Example of K_5. $S_G = \{\{1,2,3,4,5\}\}$ and $n_G = n = 5$.

Proof. Let G be complete, then $\exists\{u_1,\ldots,u_n\}$ such that $V = \{u_1,\ldots,u_n\}$ and $E = \{(u_i,u_j)|u_i,u_j \in V, u_i \neq u_j\}$. There exist $S = \{v_1,\ldots,v_{n_G}\} \in S_G$ and $u \in V$. If $\forall v,w \in S$, the distance between v and w is 1 (because G is a complete graph). Hence, if $v \in S$ and (u,v) gets removed, since $S \in S_G$, (u,v) is detected, thus a distance changes. Thus $\exists w_1,w_2 \in S$ such that (u,v) is in the shortest path between w_1 and w_2. As this shortest path's length is 1, $u = w_1$ or $u = w_2$, thus $u \in S$, thus $S = V$. Hence $n_G = n$. □

Cycles with $n \in \{3,4\}$: $n_G = n$

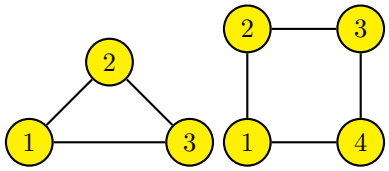


Figure 3: Examples of C_3 and C_4.
$S_{C_3} = \{\{1,2,3\}\}$ and $n_{C_3} = n = 3$.
$S_{C_4} = \{\{1,2,3,4\}\}$ and $n_{C_4} = n = 4$

Proof. If $n = 3$, G is complete, hence $n_G = n$.
If $n = 4$, $\exists u_1,u_2,u_3,u_4$ s.t. $V = \{u_1,u_2,u_3,u_4\}$ and $E = \{(u_1,u_2),(u_2,u_3),(u_3,u_4),(u_4,u_1)\}$.
Let $S \in S_G$, let $u \in V$, since $|V - \{u\}| = 3$ and $d_u = 2$, $\exists v \in S$ s.t. $(u,v) \in E$. Let's suppose that $u \notin S$. There are only two remaining vertices, w_1 and w_2 s.t. $(u,w_1),(v,w_2) \in E$. If (u,v) gets cut, the distance between v and w_2 remains equal to 1, likewise for the one between w_1 and w_2, thus $w_1 \in S$ because sensors on w_2 wouldn't detect edge failure with (u,v). However, the distance between w_1 and v remains the same, equal to 2 when (u,v) gets cut, as there are two shortest paths between both vertices; (w_1,w_2,v) and (w_1,u,v). Hence the edge failure isn't detected if $u \notin S$. Thus, by apagogical argument, we've proved that $u \in S$, hence $n_G = n$. □

Cycles with $n \geq 5$: $n_G = 3$

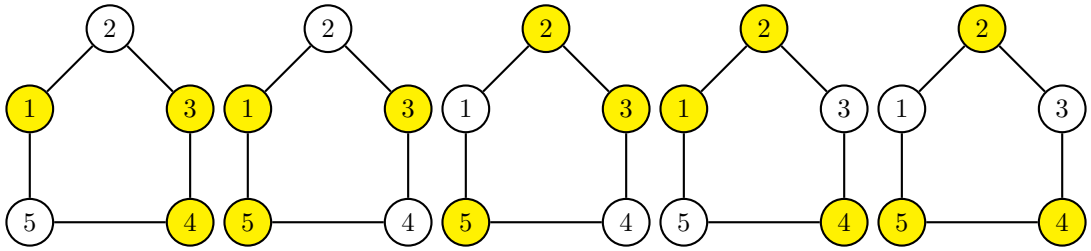


Figure 4: Example of C_5. $S_{C_5} = \{\{1,3,4\},\{1,3,5\},\{2,3,5\},\{1,2,4\},\{2,4,5\}\}$ and $n_G = 3$.

Proof. Let G be a path, then there exist n vertices $u_1, u_2, \ldots, u_n$, such that $V = \{u_1, u_2, \ldots, u_n\}$ and $E = \{(u_1, u_2), \ldots, (u_{n-1}, u_n), (u_n, u_1)\}$.
Let's suppose $n_G = 2$, then there exists $S = \{u_i, u_j\} \in S_G$ with $i \neq j$. By construction of a cycle, there exist two paths between u_i and u_j. One, p_1 contains (u_n, u_1), while the other, p_2, contains all the edges that aren't in p_1 and isn't empty; i.e $p_2 = E - p_1 \neq \emptyset$.

- if $|p_1| = |p_2|$, then if any $e \in E$ gets deleted, the size of the remaining shortest path will remain the same, as e cannot be on both p_1 and p_2.
- if $|p_1| \leq |p_2|$, then if $e \in p_2$ gets deleted, p_1 remains untouched and so does the distance. Hence edge failure isn't detected.
- if $|p_1| \geq |p_2|$, likewise, the edge failure isn't detected.

Hence $n_G \neq 2$, by apagogical argument.

Let's now prove that for every cycle of size $n \neq 4$, there exist $u_i, u_j, u_k \in V$ s.t. $i \neq j$, $i \neq k$, and $j \neq k$, and that if a, b, c are the shortest paths between (u_i, u_j), (u_j, u_k), and (u_k, u_i), respectively, $\{a, b, c\}$ is a partition of E, and $|a|, |b|, |c| \leq n/2 - 1$. Notice that with $n = 4$, the first condition would give $|a| + |b| + |c| = 4$, while the second would give $|a| + |b| + |c| \leq 3$, thus it is infeasible.

- if n is even, $u_1, u_{n/2}, u_{n/2+2}$ fits, because $\{a, b, c\}$ is by construction a partition of E, while $|a| = n/2 - 1, |b| = 2, |c| = n/2 - 1$, with $n \geq 6$.
- if n is odd, $u_1, u_{(n+1)/2}, u_{(n+3)/2}$ fits likewise.

Hence the proof.

Let's take such u_i, u_j, u_k. If an edge is removed in a, then the shortest path between u_i and u_j becomes b+c. However $|b + c| = |b| + |c| = n - |a| \geq n/2 > |a|$, thus the sensors between u_i and u_j detect the edge failure in a. Likewise for b and c. Hence $n_G = 3$. □

Trees: $S_G = \{u \in G | d_G(u) = 1\}$

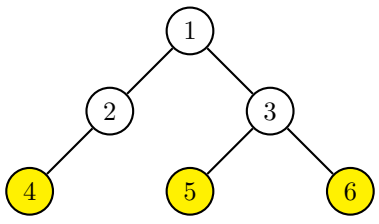


Figure 5: Example of a tree with 3 leafs. $S_G = \{\{4,5,6\}\}$ and $n_G = 3$.

Proof. Let G be a tree, by definition, it is acyclic and connected. We denote as $L = \{u \in G | d_G(u) = 1\}$ the set of its leaves.
Let's prove that L detects edge failures. Let $e = (u, v) \in E$. If e gets cut, we obtain two disjoint trees, G_u and G_v with $u \in G_u$ and $v \in G_v$. Let $l_u \in L \cap G_u$ and $l_v \in L \cap G_v$. There is a path in G between l_u and l_v. However, since $l_u \in G_u \cap G$ and $l_v \in G_v \cap G$ and G_u and G_v are disjoint, any path between l_u and l_v goes through (u, v). Thus, the distance between l_u and l_v would change if (u, v) gets cut. Hence L detects edge failures.
Let's prove that $\{L\} = S_G$. Let $S \in S_G$. Let's take $l \in L$, $\exists u \in V$ s.t. $(u, l) \in V$. Due to $d_G(l) = 1$, the distance between two vertices $v, w \neq l$ is kept unchanged if (u, l) gets cut. As such, to detect an edge failure with (u, l), l needs to be in S. Thus, $L \subset S$. Hence $\{L\} = S_G$. □

Non connected graphs: can be reduced to connected graphs.

Proof. $\exists (G_i)_i \subset G$ s.t. $\forall i, G_i$ is connected and $G = \bigsqcup_i G_i$. Let's prove that $n_G = \sum_i n_{G_i}$.
First, let's consider $\forall i$ $S_i \in S_{G_i}$, if $e \in E$, then $\exists i$ s.t. $e \in E_i$. As such, an edge failure involving e would be detected by S_i. Thus $\bigsqcup_i S_i$ detects edge failures in G. Hence $n_G \leq \sum_i n_{G_i}$.
Now, let's prove that $n_G \geq \sum_i n_{G_i}$. Let's suppose $n_G < \sum_i n_{G_i}$, then $\exists S \in S_G$ and i s.t. $|S \cap G_i| < n_{G_i}$. Thus $S \cap G_i \notin S_{G_i}$, and $\exists e \in E_i$ s.t. it is not detected by $S \cap G_i$ if cut. Since $S \in S_G, \exists u, v \in S$ s.t. the distance between u and v is changed when e gets cut. Thus, there is a path between u and v, thus $u, v \in G_i$, and there is a contradiction. Hence $n_G \geq \sum_i n_{G_i}$.
Combining both proves, we have $n_G = \sum_i n_{G_i}$, and working on non connected graphs can be reduced to working on their connected components. □

Grids: $|S| = n$

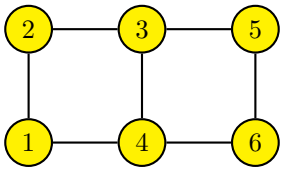


Figure 6: Example of a grid made of two C_4. $S_G = \{\{1, 2, 3, 4, 5, 6\}\}$ and $n_G = n = 6$.

Proof. Akin to the 4-cycle demonstration. □

Bipartite graphs: no straightforward general bound

A path is a particular case of bipartite graph and so are disjoints paths. Those two examples let us reach the general lower and upper bound.

2.2 Bounds on general graphs

2.2.1 Lower bound

Lemma 2.1. *k arc-disjoint paths need at least $2k$ vertices to be detected. Conversely, $2k$ vertices can detect at most k arc-disjoint paths.*

Proof. Using Lemma 1.1, this is trivial.
Suppose k arc-disjoint paths need $2k-1$ vertices to be detected. This means that there is one of these paths that need less than 2 vertices to be detected, which is impossible according to Lemma 1.1.
Suppose $2k$ vertices can detect $k+1$ arc-disjoint paths. This means that there is one of these paths that need less than 2 vertices to be detected, which is again impossible. □

Theorem 2.2. *For any graph G,*

$$n_G \quad \geq \quad \max_{v \in V} \delta(v)$$

Proof. Let $w \in V$ such that $\delta(w) = \max_{v \in V} \delta(v) :=$ D. We want to show that $n_G \geq D$. We are looking for the number of vertices needed to detect all edges adjacent to w.

- If D is even, there are exactly $\frac{D}{2}$ edge-disjoint paths going through w in G. Therefore, by Lemma 2.1, we need at least $2 \times \frac{D}{2} = D$ vertices to detect these $\frac{D}{2}$ paths. Hence, $n_G \geq D$.

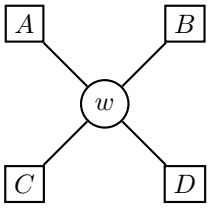


Figure 7: In this example, w is connected to four subgraphs of G: A, B, C, D. If we were to list all the edge-disjoint paths in this graph, AwD and BwC would work. We can see that we thus need at least 4 vertices to detect the edges of the paths.

- If D is odd, there are exactly $\frac{D-1}{2}$ edge-disjoint paths going through w in G, and one wz-path for some $z \in V$ that is arc-disjoint from all previous paths. Therefore, by the same lemma, we need at least D-1 vertices to detect the $\frac{D-1}{2}$ paths. Since according to previous lemma, the D-1 vertices can detect at most $\frac{D-1}{2}$ arc-disjoint paths, the wz-path is not detected. Hence, we also need at least one additional vertex to detect this path. Finally, it gives us $n_G \geq D - 1 + 1 = D$.

□

Remark. *The bound is tight for some graphs. For example, for paths we have* $n_G = \max_{v \in V} \delta(v) = 2$. *For star graphs, we have* $n_G = \max_{v \in V} \delta(v) = -1$.

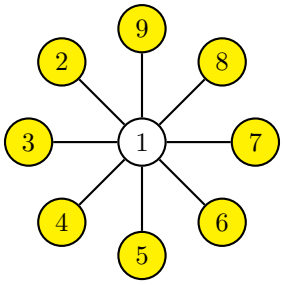
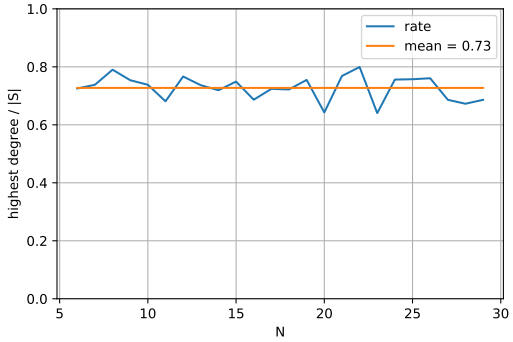


Figure 8: Example of a 9-vertices star. $S_G = \{\{2, 3, 4, 5, 6, 7, 8, 9\}\}$ *and* $n_G = 8 = \delta(1) = n - 1$.

In order to have an idea of the *"quality"* of the bound, we decided to plot the coefficient $\frac{D}{n_G}$ (D being the highest degree) with respect to the total number of vertices. We generated 500 graphs having between 6 and 30 vertices, with a random number of edges, and computed a mean for all values of N. Here is the result:



We can observe that whatever the number of vertices, the rate between the highest degree of G (lower bound) and n_G is around 0.7, which is quite high. We can therefore conclude that for any general graph, our bound has good quality. However, for a really specific graph, it is possible to make the bound terrible (for example, in a succession of triangles connected by one vertex, the highest degree is 4, while n_G grows to infinity).

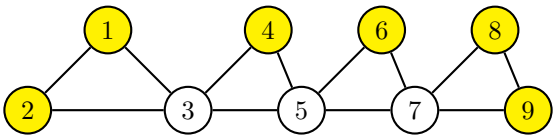
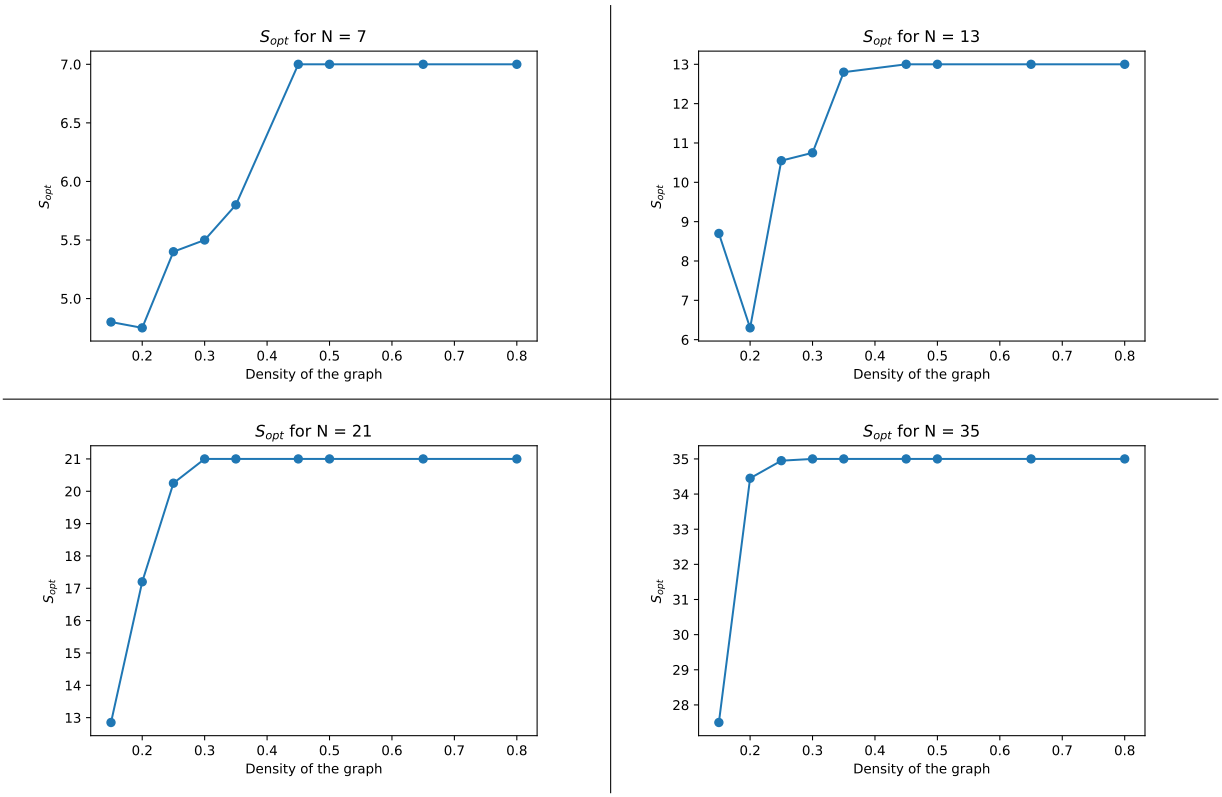


Figure 9: 4 triangles connected by one vertex each. We can see that although the highest degree is 4, $n_G = 8$ and grows to infinity if we keep on adding triangles.

2.2.2 Upper bound

Using the Integer Program presented in the next section, we managed to compute n_G for some random graphs, and display it with respect to their edge density.

Here are the results, for different values of N (note that all values are means over 50 random graphs):



We can observe that for any graph G, whatever its number of vertices, n_G goes quickly to n. That makes an upper bound really hard to find, and not that "useful" as it would always be close to n.

3 Algorithm to find S

3.1 Global idea

Finding the minimum number of vertices needed in S is easy if we can, for any given graph, determine a set S that works (note that several different S work for some graphs, for example for cycles). That's why a part of our research consisted in finding an algorithm or a model that could build a set S for any graph.

Our work was based on one key observation: if a couple of vertices $(i, i') \in S$ allows us to detect an edge j, it means j is in the shortest path from i to i': $SP_{i,i'}$.

3.2 Shortest Paths

To find shortest paths between two vertices of the graph, we used the Dijkstra Algorithm. Indeed, because the graphs we consider are not oriented and can contain cycles, we can adapt the Dijkstra algorithm by considering every edge $j \in E$ has an orientation from $i \in V$ to $i' \in V$ and also from i' to i; (i, i') with a weight equal to 1. In that case, a Shortest Path is simply a path of a certain size (the number of edges forming it).

Nevertheless, we have to keep in mind the Dijkstra Algorithm returns one possible Shortest Path. That's why the first thing to do is to find out if for a given set of vertices (i, i'), there is a **unique** Shortest Path. To do so, we can compute the powers of the adjacency matrix A (from A to A^n) and use the fact that $\forall k \in \{1, .., n\}, \forall (i, i') \in V^2; A^k_{i,i'}$ returns the number of paths of size k from i to i'.

In the method *compute_taille_PCC* from the file *Fonctions.py*, we identify if there is a unique shortest path between all the vertices of the graph and the size of it and if there is one, the size of it. We then compute the Dijkstra Algorithm in the method *compute_PCC_revPCC* to determine the set of edges that forms a Shortest Path from i to i'. It also returns the "reversed Shortest Path",i.e. the couple of vertices for which a given edge is implicated in the Shortest Path. In terms of complexity, *compute_taille_PCC* is in $O(n^3)$. A classic Dijkstra runs in $O(m + n\log(n))$. Here, we execute Dijkstra on each vertex of the graph. Thus, our Dijsktra runs in $O(nm + n^2\log(n))$. These complexities are not so great but we didn't manage to find anything more efficient.

Having this data in mind, we can model the problem with the Integer Program described next.

3.3 Integer Program

Data:

- $d^{SP}_{i,i'}, \quad \forall (i, i') \in \{1, .., n\}^2, i' \neq i$ is the distance of the Shortest Path between i and i'
- $SP_{i,i'}, \quad \forall (i, i') \in \{1, .., n\}^2, i' \neq i$ is the set of edges that forms the Shortest Path from i to i'
- $SP(j), \quad \forall j \in \{1, ..m\}$ gives all the couples of vertices (i, i') for which the edge j belongs to a Shortest Path

Variables:

- $x_i = \begin{cases} 1 \text{ if } i \in S \\ 0 \text{ otherwise} \end{cases} \qquad \forall i \in \{1, .., n\}$
- $y_j = \begin{cases} 1 \text{ if edge } j \text{ is detected} \\ 0 \text{ otherwise} \end{cases} \qquad \forall j \in \{1, .., m\}$

- $z_{i,i'} = x_i \times x_{i'}, \quad \forall (i,i') \in \{1,..,n\}^2, i' \neq i$ is a binary variable introduced to linearize the program.

Integer Program $(IP1)$**:**

$$\min \quad \sum_{i=1}^{n} x_i$$

$$\text{s.t.} \quad z_{i,i'} \times d^{SP}_{i,i'} \leq \sum_{j \in SP_{i,i'}} y_j \qquad \forall (i,i') \in \{1,..,n\}^2,\ i' \neq i \qquad (1)$$

$$\sum_{(i,i') \in SP(j)} z_{i,i'} \geq y_j \qquad \forall j \in \{1,..,m\} \qquad (2)$$

$$\sum_{j=1}^{m} y_j = m \qquad (3)$$

$$z_{i,i'} \leq x_i \quad and \quad z_{i,i'} \leq x_{i'} \quad \forall (i,i') \in \{1,..,n\}^2 \qquad (4)$$

$$z_{i,i'} \geq x_i + x_{i'} - 1 \qquad \forall (i,i') \in \{1,..,n\}^2, i' \neq i \qquad (5)$$

The constraint (1) indicates that if i and i' are chosen, the edges on the SP from i to i' are detected. (2) says that for an edge j to be detected, we have to take at least one couple (i,i') that represents the extremity of a SP in which j is implicated. (3) guarantees that all edges are detected. (4) and (5) are the constraints used to linearize $(IP1)$. Note that in the Python code, the program wasn't linearised (constraints (4) and (5) are not here, and neither is $z_{i,i'}$) as it still solves the problem.

3.4 NP-completeness

As our problem consists of deciding whether we take one vertex or not, it is modelled with binary variables. Enumerating all the possibilities would be exponential. Furthermore, we have not found a more efficient way of modelling the problem than with an Integer Program, which is known to be NP-complete. For those reasons, we tend to believe that this problem is NP-complete, although we haven't managed to prove it.

3.5 Other ideas

Since our problem could be seen as a combinatorial problem, we thought about modeling it using a Constraint Programming approach, but the fact that all variables are integers makes the integer program really great. Hence, the integer programming approach seemed more efficient.

We also though about building a more general algorithm that would build a possible S for any given graph G. Our idea was to split this graph into K subgraphs for which we have results (cycles, complete graphs, trees, paths, non-connected components etc.) and then find $S_{subgraph_i}$ for each of those subgraphs. We would then consider $S = \bigcup_{i \in \{1,..,K\}} S_{subgraph_i}$ and lastly, delete vertices that are not necessary anymore now that the subgraphs have been mixed together.
Sadly, we didn't manage to give a complete algorithm that would work for any graph. Nevertheless, we managed in the process to find some useful rules to build S_G, that are presented next.

4 Additional rules that we found to build a minimal S

Some rules presented in this part have been proven. Others are only conjectures, as we didn't find a proof for it.

Lemma 4.1. *For any graph $G = (V, E)$, let $S \subseteq V$ such that S detects edge-failure. If a vertex has degree 1, it will always be in S.*

Proof. Let $G = (V, E)$ such that $\exists v \in V$ such that $\delta(v) = 1$. Let $w \in V$ be the only neighbor of v. Suppose $v \notin S$, then $vw \notin G[S]$, the subgraph of G induced by S. Hence, the edge vw is not detected. Therefore, by contradiction, $v \in S$. □

Lemma 4.2. *Let $G = (V, E)$. Let $w_1, w_2 \in V$ such that w_1 is the vertex of G of highest degree, and w_2 is the vertex of second highest degree after w_1. That is $\delta(w_1) = \max_{v \in V} \delta(v)$ and $\delta(w_2) = \max_{v \in V \setminus \{w_1\}} \delta(v)$. If $\delta(w_1) \geq 2\delta(w_2)$, then there exists an optimal solution of our problem $S \subseteq V$ such that $w_1 \notin S$.*

Proof. We didn't find a proof. □

Lemma 4.3. *Let $G = (V, E)$ s.t. G is connected and there is a path $G_p = (V_p, E_p)$ s.t. $\exists G_1 = (V_1, E_1), G_2 = (V_2, E_2)$, two connected graphs, s.t. $E = E_1 \sqcup E_2 \sqcup E_p$ and $|V_1|, |V_2| > 1$. Then $n_{G_1} + n_{G_2} \geq n_G$ and $\forall (S_1, S_2) \in G_1 \times G_2$, $S_1 \sqcup S_2 - V_p$ detects edge-failure.*

Proof. Let such $G = (V, E), G_p, G_1, G_2$, $e \in E$, $(S_1, S_2) \in G_1 \times G_2$ be.
If $e \in E_1$, then $\exists u, v \in S_1$ s.t. the distance between u and v changes if e gets cut, when limited to E_1. However, since there is only one path between V_1 and any vertex outside V_1 and $u, v \in S_1$, e is detected if cut in E.
Likewise with $e \in E_2$.
Otherwise, $e \in E_p$. Let $(u_1, u_2) \in S_1 \times S_2$. There is a unique path between u_1 and u_2 and it contains every $e_p \in E_p$, thus e. So if e gets cut, there is no more path between u_1 and u_2, hence the edge-failure is detected.
Thus, $S_1 \sqcup S_2$ detects edge-failure in G. Hence, we have $n_G \leq n_{G_1} + n_{G_2}$.
Let $u \in S_1$. If $u \in V_p$, then $\exists e \in E_1, v \in S_1 - u$ s.t. if e gets cut the distance between u and v changes. $\exists w \in S_2 - V_p$, because $n_{G_2} \geq 2$ since $|V_2| \geq 2$. Since there is only one path between V_1 and V_2, since u is on that path, $v \in V_1$ and $w \in V_2$ the distance between v and w is equal to the sum of the distance between v and u and the distance between u and w. As such, v and w detect the edge failure if e gets removed. Hence u can be removed from S_1, while keeping S_1's ability to detect edge failures.
Likewise for S_2. Thus $\forall (S_1, S_2) \in G_1 \times G_2$, $S_1 \sqcup S_2 - V_p$ detects edge failure. □

5 Additional work that was done but didn't lead to results

5.1 On the bounds of n_G

We also tried to find a lower bound and an upper bound of n_G by looking at the chromatic number χ_G. Nevertheless, we didn't find any relevant result. Indeed, as explained in the Upper Bound section, it looked hard to find a better upper bound than n. On the lower bound, it seemed that we always had $n_G \geq \chi_G$. But we found some cases in which $\max_{v \in V} \delta(v) \geq \chi_G$, which meant that the Lower Bound found on the maximum degree of the graph was a better lower bound than anything we could find with the chromatic number.

We looked at the clique number too. Because it is known that $\forall G, \chi_G \geq \omega_G$, the clique number would not prove to be a tight lower bound should a result be proven with the chromatic number. We thus tried to find a direct relation of equality between S and some data about the graph and results on the maximal clique of a graph (notably, n, m, the number of similar cliques, the clique number, the

number of vertices shared between the similar maximal cliques) but we didn't find anything.

Once we found a correlation between S_G and the density of the graph, we also tried to find a direct formula that would give a bound for n_G using the fact that the density D is defined as such for an undirected simple graph: $D = \frac{2 \times m}{n \times (n-1)}$. However, nothing came out of this.

5.2 On interval and planar graphs

For interval graphs, our aim was to find a relation between the vertices forming S_G and the sets (vertices) that would intersect themselves between each others. We also looked for a formula that would give n_G as a function of the number of non-empty intersecting sets (vertices) but didn't find anything.

Planar graphs didn't look so different as any graph in our research, which is why we didn't search for anything particular on this subject.

5.3 Possible algorithm

The Lemma 4.3 offers the beginning of an algorithm generating a set in S_G. By removing the edges composing every cycle in G (which can be done by depth search), we obtain trees linking more complex graphs. These graphs are mixed between each others. Unfortunately we didn't find any good result on such graphs and as a result didn't develop the idea further. But here is the skeleton of what would be algorithm;

- 1. Mark all the edges that compose a cycle in G.
- 2. For all the marked connected component, find the sets that detect edge failures.
- 3. For all these sets, find the one of minimal size by removing the vertices which are reached by an unmarked edge.
- 4. Mark the vertices of such set.
- 5. Using Lemma 4.3, the marked vertices make for a minimal subset of G.

We could add some rules to this algorithm. For example, we could take in foresight vertices of degree equal to 1.