ENSEIRB-MATMECA 1ère année informatique

DEVOIR MAISON - ALGORITHMIQUE DES GRAPHES

Modélisation de la propagation du Covid-19

Membres de l'équipe :

BAUDE Clément

PAU Marin

STAN Sophie

Encadrés par M. Nicolas HANUSSE

Résumé : Ce projet a pour but de modéliser numériquement la propagation d'une maladie contagieuse comme celle du Covid-19 dans une population donnée. Entre autres, il nous a permis d'étudier et de réaliser l'influence que peuvent avoir différents paramètres sur la propagation d'une maladie. Différentes courbes ont donc été tracées afin de représenter cette évolution, et ce à l'aide du langage de programmation `Python`.

Table des matières

1 Introduction

Cette partie présente les choix structurels faits, afin de décrire et implémenter au mieux la propagation d'une maladie dans une population. À cet effet, nous avons donc implémenté des classes en Python qui portent chacune une intelligence précise.

1.1 Présentation du sujet et choix de représentation

Pour simuler une propagation de maladie dans une population, il faut des individus fictifs et simuler des interactions entre ces individus. Pour ce faire, nous avons créé une classe de personne/individu, deux classes de graphes (l'une symbolisant les interactions **possibles** et l'autre les interactions **effectives** entre les individus), et enfin une classe représentant le monde fictif dans lequel les personnes évoluent. Cette classe plus abstraite est en quelque sorte l'horloge ou le maître du "jeu". Toutes ces classes possèdent des attributs (ou variables) et des méthodes (ou fonctions) en accord avec les hypothèses du modèle, hypothèses qui vont être précisées par la suite. Ainsi, nous avons créé quatre classes : la classe `Person`, la classe `Graph`, la classe `SubGraph` et la classe `World`.

1.2 La classe `Person`

Cette classe porte relativement peu d'intelligence et sert principalement à regrouper les informations d'une personne.

```
class Person:

    def __init__(self, ID, disease_duration, state='S', contamination_day=None, confinement_day=
    None):
        """
        Parameters
        ----------
        ID is a constant number which identifies a Person.
        disease_duration is the duration of the disease.
        state is a character: 'S' for Sain, 'M' for Malade, 'R' for Rémission, 'D' for Décédé
        contamination_day is a constant number, the date of contamination
        confinement_day is a constant number, the date of confinement
        """
        self.ID = ID
        self.state = state
        self.contamination_day = contamination_day
        self.confinement_day = confinement_day
        self.disease_duration = disease_duration
        self.daily_met_persons = []
```

Les attributs de cette classe sont les suivants :

- `ID` est un numéro d'identification qui permet de retrouver une personne rapidement dans la liste `population` ;
- `state` indique l'état de santé de la personne. (cf Docstring) ;
- `contamination_day` est la date de contamination de la personne ;
- `confinement_day` est la date de confinement de la personne ;
- `disease_duration` est la durée de la maladie (nécessaire pour déterminer la longueur maximale de la liste `daily_met_persons` ;
- `daily_met_persons` est une liste de listes propre à une personne. Chaque liste correspond à un jour. Chaque jour cette personne voit une liste de personnes, qui est rajoutée à la liste `daily_met_persons`.

En association avec cette classe `Person`, nous avons créé :

- une variable globale `POPULATION_SIZE` représentant la taille de l'échantillon, sa valeur par défaut est de 1000.
- une liste `population` de longueur `POPULATION_SIZE` qui répertorie toutes les personnes, avec correspondance entre l'`ID` et l'index de la liste.

La classe `Person` ne porte pas beaucoup d'intelligence. En effet, seules trois méthodes l'accompagnent, à savoir : une méthode `is_confined(self)` vérifiant si la personne est confinée ou non, une méthode `add_daily_met_person(self, visited, visited_by)` qui rajoute la liste des personnes rencontrées le jour dit, et enfin une méthode `construct_candidates_to_confinement(self)` qui construit une liste de candidats au confinement, après la mort de la personne (cf Scénario 2 et 3 en 2.2.1). La première méthode a une complexité en temps constante. La deuxième est linéaire en la taille de `visited` plus la taille de `visited_by`. La première taille est bornée par k, la deuxième par

n. La fonction s'exécute donc en $O(k+n)$. La troisième méthode, elle, concatène toutes les listes contenues dans `daily_met_persons`, puis supprime toutes les redondances de personnes, ce qui coûte cher mais est nécessaire.

1.3 La classe `Graph`

Cette classe modélise le réseau de contacts de la population. Nous avons choisi de le modéliser par un graphe **orienté** pour deux raisons.
Le graphe aléatoire de taille n nécessite un nombre naturel k de contacts pour sa construction. Le nombre total d'arêtes $k*n$ dans le cas d'un graphe non orienté sans redondance d'arêtes est majoré par $\sum_{i=0}^{n-1} i$ ie $k \leq \frac{n-1}{2}$. Il aurait donc fallu faire attention à cette valeur limite de k (à moins d'accepter d'avoir des arêtes redondantes, ce qui ne semble pas facile à implémenter et gérer).
Enfin, nous avons estimé qu'il n'y avait pas forcément équivalence dans les relations de rencontre. En effet, prenons par exemple une personne travaillant au supermarché. Dans le graphe de contacts, elle sera reliée à tous ses clients, et sa famille/amis. Le nombre de clients est potentiellement très élevé en comparaison avec le nombre de membres de sa famille et ses amis. Ainsi, lorsque cet employé devra choisir un certain nombre de personnes à voir parmi celles auxquelles il est relié, il risque de ne jamais voir sa famille et ses amis, et de les exclure. Par conséquent, des rencontres ne sont font plus, car le quota de personnes à voir du jour est épuisé. Ce problème n'est pas rencontré si on considère des arêtes orientées, ie des arcs. Tous les arcs pointent sur le vendeur de supermarché, et lui-même pointe sur les membres de sa famille/amis.

```python
class Graph:
    """ This class allows to construct an oriented Graph implemented by an adjacency list and a
    list of "parents".

    Vertices are Persons.
    Each Person has a group with a certain amounts of relationships.
    A graph can be random, circular, or mixed (union of both).
    """

    def __init__(self, population, num_relationships, circular=False, random=False):
        """
        Parameters
        ----------
        population is the list of all the Persons sorted out following their ID.
        num_relationships is the number of Persons, a Person is related to.
        circular is an option which allows to set up a circular graph.
        random is an option which allows to set up a random graph.
        """
        self.population = population
        self.population_size = len(population)
        self.num_relationships = num_relationships

        self.can_visit = [[] for k in range(self.population_size)]
        self.can_be_visited_by = [[] for k in range(self.population_size)]
        if circular:
            self.construct_circular_graph()
        if random:
            self.construct_random_graph()
```

La classe `Graph` possède plusieurs attributs :
- `population` est la liste décrite en 1.2 ;
- `population_size` est le nombre de sommets du graphe ;
- `num_relationships` est le nombre de personnes qu'une personne choisit d'être en contact avec ;
- `can_visit` est la liste d'adjacence du graphe. À l'indice `k` on retrouve la liste de personnes/contacts choisis par la personne d'`ID` égal à ***k***.
- `can_be_visited_by` est la liste de listes associée à la liste d'adjacence `can_visit` du graphe. Elle permet de garder une trace des personnes qui peuvent visiter une personne. Autrement dit, à l'indice ***k*** on retrouve la liste des personnes qui peuvent visiter la personne ayant l'ID ***k***. On a également : si personne ***j*** appartient à `can_visit[personne i]` alors personne ***j*** appartient à `can_be_visited_by[personne i]`.

De plus, la méthode `add_edges(self, person_1, person_2)` accompagne la classe. Elle permet de créer une arête orientée de `person_1` à `person_2` seulement si elle n'existe pas déjà, en temps constant. Enfin, la méthode `remove_vertex(self, dead_person)`, élimine toutes les arêtes reliées à une personne morte pour ne plus qu'elle soit visitée ou qu'elle visite quelqu'un. La complexité en temps de cette méthode est en $O(n + k)$.

1.4 La classe `SubGraph`

La classe `SubGraph` est responsable d'organiser le graphe quotidien des rencontres. Elle complète la classe `Graph` au sens où `Graph` représente les **possibles** rencontres entre les personnes (un réseau de contacts dense), et `SubGraph` est un sous-graphe de `Graph` qui établit la liste des rencontres quotidiennes que les personnes sont autorisées à effectuer (en fonction des confinements, des modes de visites, des personnes décédées...).

```
class SubGraph:
    """ This class gathers the lists of Persons, a Person will see on D-day.

    A SubGraph is the initial Graph of relationships if there is no visiting mode enabled (K =
    K_PRIME).
    Any other cases: each Person chooses num_persons_to_visit among the num_relationships they
    have.
    """

    def __init__(self, relationships_graph, num_persons_to_visit, visiting_mode="None",
    confinement_mode="None"):
        """
        Parameters
        ----------
        relationships_graph is the network of relationships/contacts between the Persons.
        num_persons_to_visit is the number of Persons a Person will visit daily.
        visiting_mode can be None, dynamic or static (see main.py).
        confinement_mode can be None, low or high (see main.py)
        """

        self.relationships_graph = relationships_graph
        self.population_size = relationships_graph.population_size
        self.num_persons_to_visit = num_persons_to_visit
        self.visiting_mode = visiting_mode
        self.confinement_mode = confinement_mode

        if self.visiting_mode == "None":
            # Visiting everyone everyday
            self.will_visit = copy.copy(relationships_graph.can_visit)
            self.will_be_visited_by = copy.copy(relationships_graph.can_be_visited_by)

        else:
            self.will_visit = [[] for k in range(self.population_size)]
            self.will_be_visited_by = [[] for k in range(self.population_size)]
            self.construct_sub_graph()
```

Dans cette classe on retrouve donc les attributs suivants :

- `relationships_graph` qui est le graphe de contacts initial (`Graph`) ;
- `population_size` qui est le nombre de sommets du graphe ;
- `num_persons_to_visit` qui est le nombre de visites quotidiennes autorisées. Ce nombre doit bien être inférieur au nombre de relations qu'une personne possède (`num_relationships`)
- `visiting_mode` est le mode de visite quotidien (statique, dynamique ou aucun) ;
- `confinement_mode` est le mode de confinement choisi en variable globale (fort, faible ou aucun).

Au vu de l'organisation épineuse des rencontres du jour, la classe `Subgraph` regroupe en tout six méthodes permettant de traiter les différents cas de personnes confinées, décédées et les différents modes de confinement et de visites. Ces différentes méthodes permettent de construire le sous-graphe, l'actualiser après chaque jour, de confiner/déconfiner des personnes précises.
La plupart de ces méthodes parcourent donc les listes d'adjacence et sont donc de complexité en temps quadratique.

1.5 La classe `World`

La classe `World` représente l'horloge de la simulation. C'est elle qui porte la plus grande partie de l'intelligence du programme.

```
class World:

    def __init__(self, population, disease_params, p_test, num_tested_persons, scenario,
    world_state, elapsed_days=1):
        """
        Parameters
        ----------
        population is the list of persons.
        disease_params is a structure (DEATH_RATE, SPREAD_RATE, DISEASE_DURATION).
        p_test is the test validity probability.
        num_tested_persons is the number of randomly tested persons everyday.
        scenario is a enum: the n-th scenario chosen.
        world_state is a dictionary in which person states are counted.
        elapsed_days (the counter of elapsed days since the beginning of the experiment).
        """
        self.alive_persons = copy.copy(population)

        self.death_rate = disease_params.DEATH_RATE
        self.spread_rate = disease_params.SPREAD_RATE
        self.disease_duration = disease_params.DISEASE_DURATION

        self.p_test = p_test
        self.num_tested_persons = num_tested_persons
        self.scenario = scenario
        self.world_state = world_state
        self.elapsed_days = elapsed_days
```

Les attributs de cette classe sont :

- `alive_persons` est la liste des personnes encore en vie ;
- `death_rate` représente le taux de mortalité du virus ;
- `spread_rate` représente le taux de propagation du virus ;
- `disease_duration` correspond à la durée de la maladie ;
- `p_test` correspond à la probabilité d'obtenir un test positif de la maladie sachant que la personne est malade ;
- `num_tested_persons` est le nombre de personnes testées par jour ;
- `scenario` est de type `enumerate` et permet d'indiquer en variable globale un certain scénario de confinements/tests des personnes ;
- `world_state` est un dictionnaire de la forme `{'S':` nb_S`,'R':` nb_R`,'D':` nb_D`,'M':` nb_M`,'C':` nb_C`}` où nb_i correspond au nombre de personnes dans l'état i. Il représente l'état courant du monde, c'est-à-dire, un compteur de l'état de santé des personnes ;
- `elapsed_days` est un compteur du nombre de jours passés depuis le début de l'expérience.

La classe `World` contient la méthode centrale `update_world(self, sub_graph, population)`, qui a de nombreuses missions telles que :

1. Mettre à jour l'état d'une personne, son confinement, sa liste de personnes vues au cours des r derniers jours ;
2. Prévenir la classe `SubGraph` des éventuels confinements, déconfinements, décès, de la fin d'une journée (et donc du renouvellement du graphe dans le mode dynamique). Ainsi, le `Subgraph` avise les nouvelles rencontres autorisées au jour suivant ;
3. Prévenir la classe `Graph` d'un décès ;
4. Mettre à jour la liste des personnes en vie, et l'état du monde `world_state` et le retourner pour tracer les courbes de l'expérience jour après jour.

La complexité de la méthode `update_world(self, sub_graph, population)` est donc beaucoup plus difficile à estimer étant donné qu'à l'intérieur même de sa boucle de parcours de la population, elle fait appel à tout un tas d'autres méthodes des autres classes.

1.6 Hypothèses du modèle

Le modèle de propagation du virus vérifie les règles de changement d'état suivantes. Si une personne saine X rentre en contact avec une personne malade Y, alors X devient malade par l'intermédiaire de Y avec la probabilité $q = \text{spread\_rate} = 3\%$ pour une durée de $r = \text{disease\_time} = 14$ jours. Si une personne est malade depuis r jours, alors soit elle décède avec la probabilité $p = \text{death\_rate} = 2\%$, soit elle devient immunisée avec la probabilité $(1 - p) = 98\%$. Deux règles pour les personnes immunisées ou décédées s'ajoutent : une personne immunisée ou décédée ne change jamais d'état ; une personne décédée ne peut contaminer une personne saine. Enfin, initialement on considère qu'une seule personne est malade et les $(n - 1)$ autres sont saines. Toutes les valeurs numériques introduites ici sont celles par défaut. Sauf mention contraire, ce sont celles utilisées pour produire nos tests et afficher nos figures. À noter également que nos simulations s'arrêtent lorsque la maladie est éradiquée (d'une manière ou d'une autre).

2 Résultats et interprétation

Cette partie présente les résultats des tests obtenus pour différents types de graphes, parmi trois types : **circulaire, aléatoire** et **mixte**. Dans la première partie, l'implémentation des différents graphes par des méthodes relatives à la classe `Graph` est expliquée, puis des figures illustrent les résultats d'évolution de l'état de santé de la population. Dans la deuxième partie, différents types de tests de dépistages sont mis en place et la mise en quarantaine (forte ou faible) est instaurée.

2.1 Une première modélisation

Dans cette première modélisation, on implémente les 3 types de graphes énoncés précédemment en ajoutant des méthodes à la classe `Graph`. Par ailleurs, on ajoute deux modèles différents à ces graphes : le modèle statique et le modèle dynamique, de paramètre $k' = 5$ (valeur par défaut estimée à partir du fait qu'une famille contient en moyenne quatre membres et qu'elle va faire des courses au supermarché). Dans le modèle statique une personne confinée ne peut fréquenter seulement k' personne sélectionnées au hasard et n'est pas autorisée à voir les autres personnes pendant le confinement. En revanche, dans le modèle du graphe dynamique, chaque personne est autorisée à voir k' personnes chaque jour, qu'elle sélectionne parmi ses $k = 50$ contacts. Elle a donc la possibilité de voir des personnes différentes d'un jour à l'autre.

2.1.1 Graphe circulaire

```
class World:
    [...]
    def construct_circular_graph(self):
        """ Constructs a circular sub_graph. """
        pop_size = self.population_size
        for k in range(pop_size):
            self.add_edge(self.population[k], self.population[(k + 1) % pop_size])
            self.add_edge(self.population[k], self.population[(k - 1) % pop_size])
```

La méthode `construct_circular_graph(self)`, de complexité en temps linéaire ($O(n)$) permet de créer le graphe circulaire. Ce dernier contient n personnes numérotées de 1 à n, et chaque personne numérotée i est reliée aux personnes $(i-1)$ et $(i+1)$ modulo n. Par conséquent, chaque sommet du graphe circulaire possède seulement 2 voisins. Cette modélisation ne reflète pas la réalité puisque chaque personne a nettement plus que 2 voisins dans la vie réelle. D'ailleurs, en testant, on a remarqué que le virus n'avait que peu de chance de se propager à cause du faible nombre de voisins. C'est la raison pour laquelle nous avons exceptionnellement augmenté le paramètre $q =$ `spread_rate` à 10%. Analysons les résultats obtenus à l'aide de ce graphe :

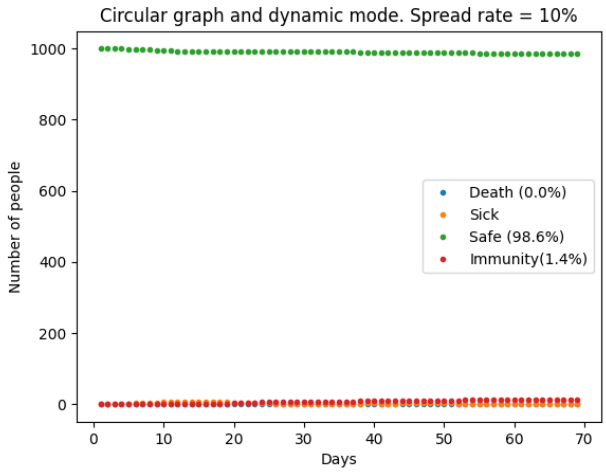

FIGURE 1 – Simulation sur un graphe circulaire avec le modèle dynamique

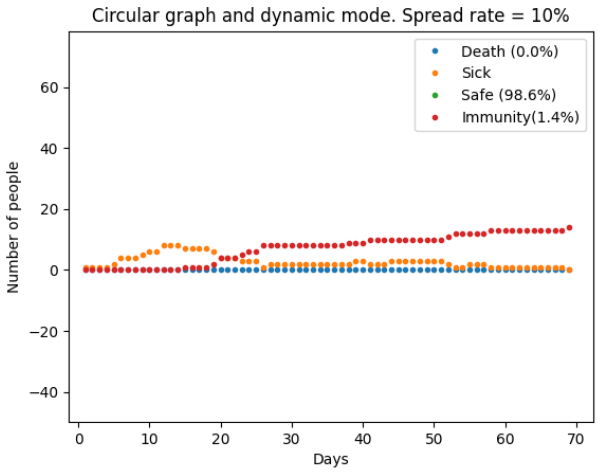

FIGURE 2 – Zoom des courbes obtenues

On peut tout de suite noter que le virus ne s'est pas propagé. En effet le pourcentage de personnes saines (donc jamais atteintes par le virus) à la fin de l'épidémie est de 98,6% sur la figure 1. De même, le pourcentage de personnes immunisées est de seulement 1,4%. Enfin, le virus n'a fait aucun mort pendant l'épidémie. Tout ceci n'est pas réaliste.
Passons maintenant aux résultats obtenus à l'aide des graphes aléatoires et mixtes, résultats qui sont beaucoup plus satisfaisants.

2.1.2 Graphe aléatoire

```
class World:
    [...]
     def construct_random_graph(self):
         """ Constructs a random sub_graph """
         pop_size = self.population_size
         for i in range(pop_size):
             potential_relationships_i = []
             for j in range(pop_size):
                 if i != j:
                     potential_relationships_i.append(self.population[j])
             relationships_i = rd.sample(potential_relationships_i, self.num_relationships)
             # num_relationships is obviously less than len(potential_relationships_i) = n - 1
             for person in relationships_i:
                 self.add_edge(self.population[i], person)
```

Le graphe aléatoire est créé grâce à la méthode `construct_random_graph(self))`. Cette dernière est de complexité en temps : $O(n^2)$. (À priori, dans le pire des cas, l'opérateur Python `rd.sample()` parcourt la liste `potential_relationships_i` dont la taille est égale à $k' < n$ une seule fois). Ce graphe est obtenu de la manière suivante. Chaque nœud choisit $k = num\_relationships$ nœuds choisis au hasard, à l'aide de la méthode `rd.sample(liste, taille_échantillon)` déjà existante dans le module `random` de `Python`. Le nombre total d'arêtes est donc de $k \times n$ et le degré moyen est $2k$.

2.1.3 Graphe mixte

Le graphe mixte est l'union des deux graphes précédents. Les graphes aléatoire et mixte sont assez proches et de ce que nous avons remarqué donnent des résultats semblables. C'est la raison pour laquelle à partir d'ici, nous n'analyserons seulement que les résultats obtenus avec un graphe mixte.
D'autre part, les graphes, seront aussi accompagnés d'une asymptote horizontale en rouge afin de localiser le pic de personnes malades, et une asymptote horizontale en noir, afin de situer le moment où la capacité des hôpitaux est dépassée. Nous avons estimé arbitrairement un nombre de lits à 1/5 de la population. Après recherche, ce nombre est vraiment surestimé mais notre modèle ne prend non plus en compte les porteurs sains. L'un sur l'autre, nous faisons donc l'hypothèse que cette barre noire fictive est adaptée.

Ainsi, observons les résultats obtenus sur le graphe mixte sur les figures 3 (modèle statique) et 4 (modèle dynamique).

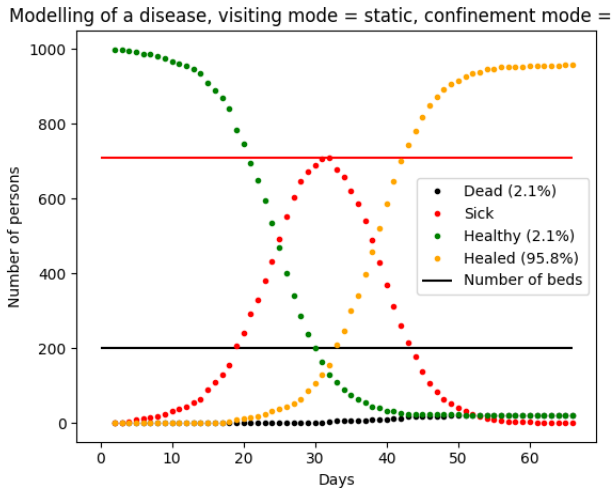

FIGURE 3 – Simulation sur un graphe mixte avec le modèle statique

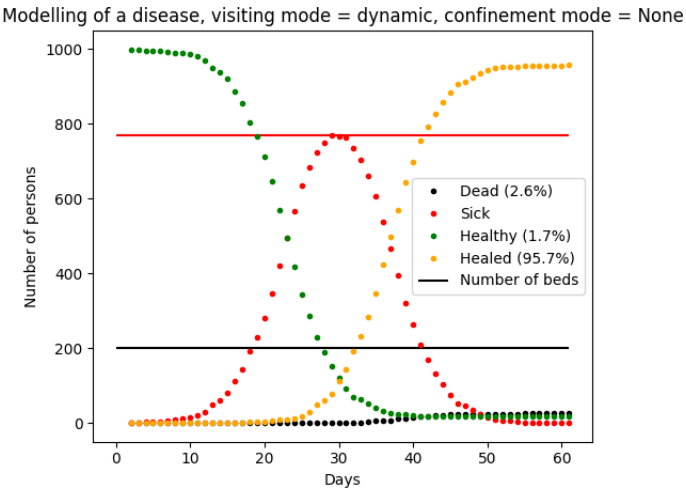

FIGURE 4 – Simulation sur un graphe mixte avec le modèle dynamique

On peut noter que les deux courbes sont assez similaires. Un pic de personnes contaminées arrive autour du 30ème jour. Lorsqu'environ 90% de la population est immunisée, l'immunité de groupe joue son rôle et la maladie disparaît d'elle même. On pourrait donc en conclure que les modèle dynamique et statique sont équivalents, mais il serait intéressant de comparer les deux modes sans le phénomène d'immunité de groupe. En effet il est préférable de trouver de moyens d'arrêter la maladie avant que 90% de la population ne soit malade. Pour cela nous allons reproduire la même expérience sur les figures 5 et 6 mais en diminuant le taux propagation de 2%.

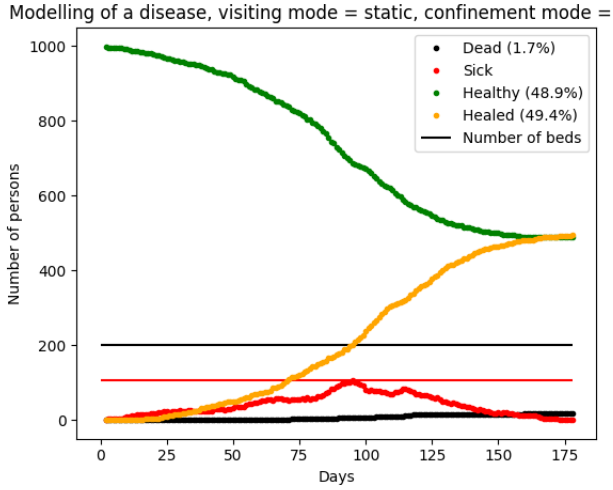

FIGURE 5 – Simulation sur un graphe mixte avec le modèle statique et un taux de propagation de 1%

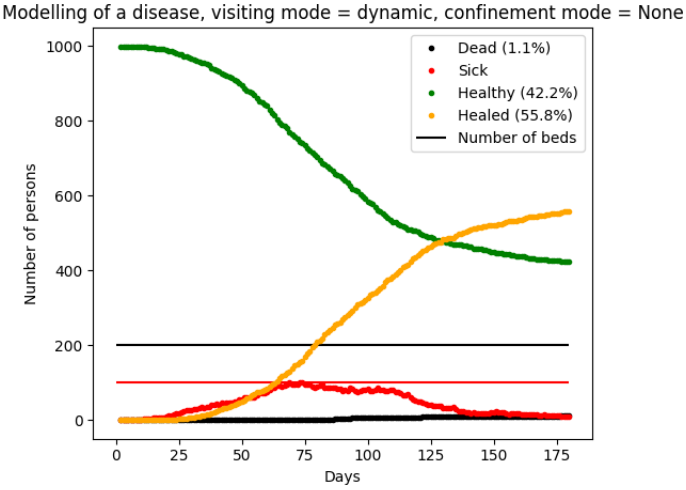

FIGURE 6 – Simulation sur un graphe mixte avec le modèle dynamique et un taux de propagation de 1%

On observe deux courbes légèrement différentes. Le modèle dynamique semble contaminer plus de personnes. (42% de personnes encore saine, contre 49% pour le modèle statique). Le modèle statique est donc préférable pour contenir la maladie (sans prendre en compte les considérations économiques et sociales). Toutefois, notre modèle ne reflète pas vraiment la réalité : on ne confine pas tout le monde dès qu'une maladie est détectée.

Nous allons donc un peu améliorer le modèle avec un **'mode évolutif'** : dès que 5% de la population est malade,

le modèle dynamique est activé (il pourrait correspondre à la mise en place de mesures de distanciations sociales) puis quand 15% de la population est touchée, le modèle statique est activé (mise en place d'un confinement). Ce nouveau modèle ne correspond pas tout à fait à la réalité non plus puisque, à moins de faire des tests massifs, le nombre de malades n'est pas connu mais il se rapproche plus de la réalité. Lorsqu'aucun des deux modèles n'est actif, une personne visite donc les k personnes avec qui elle peut avoir des contacts. Étant donné que maintenant k représente les contacts effectifs et plus les contacts potentiels, nous allons diminuer la valeur par défaut de k à 10 (ce qui correspond plus au nombre moyen de personnes rencontrées par jour). Voici le résultat de la simulation en **'mode évolutif'** :

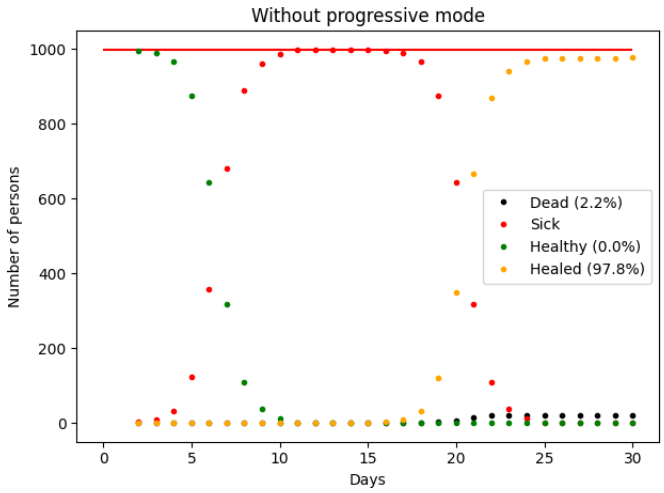

FIGURE 7 – Simulation sur un graphe mixte sans le mode évolutif

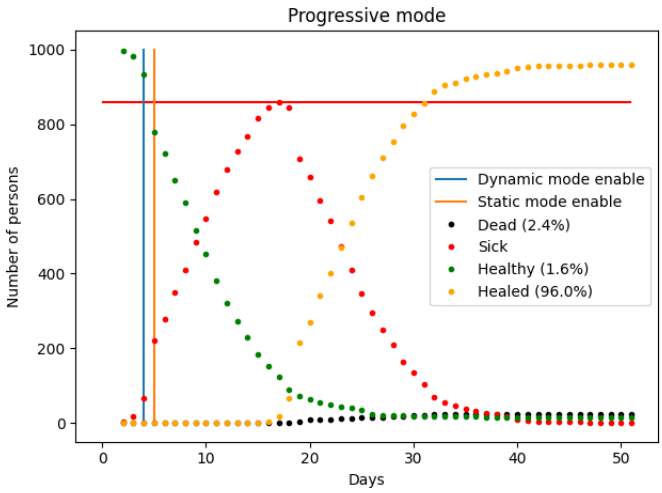

FIGURE 8 – Simulation sur un graphe mixte avec le mode évolutif

Plusieurs choses peuvent être notées sur les figures 7 and 8 :

— Lorsqu'aucun modèle n'est actif (début de l'épidémie) la courbe explose, en seulement quelques jours le cap des 15% de la population infectée est atteint et le modèle statique est activé.
— Le pic de l'épidémie est atteint bien plus tôt que précédemment dans le graphe circulaire (20 jours contre 80)
— L'épidémie s'arrête ici aussi lorsque 80% de la population est immunisée.
— La proportion de personnes décédées et saines est un peu près similaire à précédemment
— On observe une chute continue du nombre de malades une quinzaine de jours après la mise en place du modèle statique. Le pic observé en rouge n'est plus une *cloche* mais un *pic*. Ce modèle est donc efficace pour contenir le plus possible la maladie.

2.1.4 Conclusion

Nous avons vu que le graphe **mixte** est celui qui représente le mieux la vie de tous les jours nous allons donc le conserver pour la suite. Nous allons par ailleurs garder aussi un mode **dynamique**, qui selon nous représente le mieux la manière qu'ont les personnes d'interagir entre eux.
Nous pouvons toutefois émettre quelques critiques sur les modèles utilisés :

— Le modèle considère qu'une personne guérie est immunisée pour toujours et ne peut donc plus tomber malade or nous n'avons aucune certitude sur cette affirmation.
— Le modèle statique correspond à un confinement avec sa famille de k' personnes. Ceci est critiquable suivant le fait que toutes les familles n'ont pas le même nombre de personnes,et que beaucoup de personnes sont isolées seules dans leur appartement. De plus, il manque les sorties occasionnelles, comme pour faire les courses par exemple, qui regroupent des personnes de diverses familles.
— Le modèle dynamique est intéressant puisqu'effectivement on ne rencontre pas tous les jours les mêmes personnes. Toutefois il pourrait être amélioré en utilisant différent degrés (et donc des poids sur les arcs) de relation entre les personnes : les personnes très proches (famille) que l'on voit tous les jours, les personnes proches (amis, collègues de bureau) que l'on voit plusieurs fois par semaine, les personnes moyennement

proches (dans le même établissement scolaire, bâtiment de travail) qu'on peut croiser 'aléatoirement' tous les jours (à la cantine, dans les couloirs), les personnes lointaines (même ville) qu'on peut voir aléatoirement quelques fois par semaine (dans la rue, en faisant ses courses).

— La maladie ne peut se terminer qu'au bout de r jours, ce qui pourrait être amélioré, par exemple en faisant suivre à la durée de la maladie une loi gaussienne d'espérance r et d'écart type à déterminer.

— Le taux ne mortalité ne prend pas en compte la saturation des hôpitaux et le temps dont nous disposons pour nous préparer à l'épidémie. Il est évident que tout le monde n'a pas forcément besoin d'une hospitalisation, comme c'est le cas des porteurs sains. Or le modèle ne différencie pas les porteurs sains de la maladie (qui sont malades sans pronostic engagé) et les malades dont les chances de survie dépendent d'une hospitalisation. Évidemment, le taux de décès devrait être plus important dans le cas de la figure 8 que de la figure 3 car le pic de malades arrive plus tôt et est plus important dans le premier cas.

2.2 Tests et mise en quarantaine

Nous introduisons dans cette partie deux nouveaux phénomènes : les tests et la mise en quarantaine. La mise en quarantaine est appelée confinement dans le sujet pour éviter les confusions avec le modèle statique qui correspond au confinement de **toute** la population. Nous appellerons ce phénomène mise en quarantaine (qui est en fait le confinement ciblé d'une seule personne). La mise en place des tests dépend des scénarios proposés dans les parties suivantes. Lorsqu'une personne est malade est testée, la probabilité que le test soit positif est p_test. Si une personne est détectée positive elle est mise en quarantaine. Il existe deux types de mise en quarantaine, la mise en quarantaine faible : visites autorisées qu'avec les k' personnes visitées le jour précédent et la mise en quarantaine forte : aucun contact possible. Une mise en quarantaine dure $r + 1$ jours

2.2.1 Sans test mais avec mise en quarantaine

Dans ce scénario, aucun test n'est effectué mais lorsqu'une personne décède, toutes les personnes avec qui elle a été en contact les $r' \leq r$ derniers jours sont mises en quarantaine. Voyons les résultats pour les mises en quarantaine faible et forte.

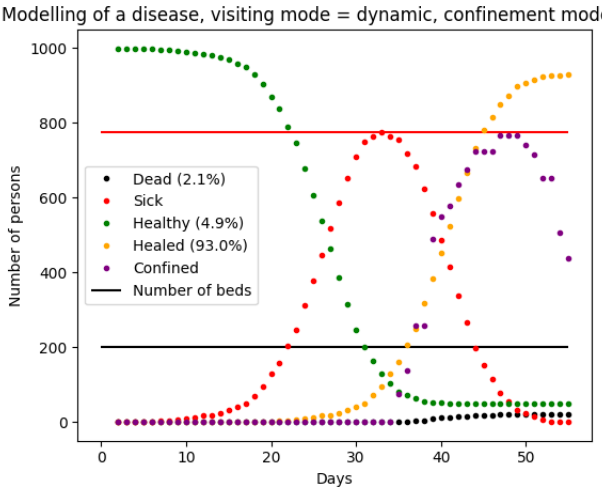

FIGURE 9 – Simulation sur un graphe mixte avec le mode évolutif et la mise en quarantaine faible

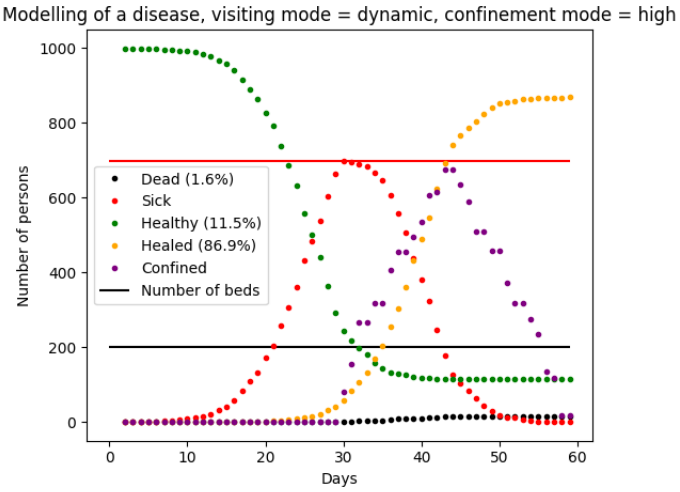

FIGURE 10 – Simulation sur un graphe mixte avec le mode évolutif et la mise en quarantaine forte

On peut remarquer sur les figures 9 et 10 que les mises en quarantaine n'ont pas vraiment été efficaces (cf les Figures 5 et 6). En effet le pic de personnes malades n'est pas abaissé. Cela est dû au fait qu'attendre qu'une personne décède pour confiner ses proches ne permet pas de stopper la transmission à temps. La maladie a le temps de se propager chez les proches mais aussi leurs connaissances.

Cependant, on peut quand même observer qu'au moment de la disparition de la maladie, il reste tout de même environ 5 à 10% de personnes non infectées par la maladie (courbe verte). Il s'agit des personnes de l'entourage du défunt qui n'étaient pas malades et que nous avons confiné par précaution.
On observe ici aussi que la seule manière d'arrêter l'épidémie est l'immunité de groupe. On comprend donc la nécessité d'introduire les tests, ils permettront d'isoler plus rapidement les personnes contaminées. Le taux de décès est identique à ce qu'on pouvait observer dans la partie **Graphe mixte** ; ceci n'est pas surprenant puisqu'à la fin de l'épidémie autant de personnes ont été malades (environ 90%), encore une fois on prend pas en compte ici la saturation des hôpitaux. En réalité on ne peut pas vraiment comparer le taux de décès, comme il y a approximativement le même nombre de malade dans toute nos expériences le nombre de décès dépend de la 'chance' ou 'malchance' des personnes.

2.2.2 Avec test sur proches des défunts

Dans ce nouveau scénario, des tests sont mis en place sur les personnes ayant contacté le défunt dans les r' derniers jours. Si la personne est malade ('M'), elle est détectée positive avec une probabilité $p\_test = 80\%$ (valeur par défaut), et une personnes positive est mise en quarantaine. Analysons les résultats obtenus pour ces tests.
Normalement, on s'attend à des résultats similaires au 2.2.1, mais moins bons. En effet, on ne confine plus tout l'entourage par précaution, mais seulement ceux qui sont déjà infectés, permettant aux personnes saines de se contaminer par l'intermédiaire de quelqu'un d'autre.

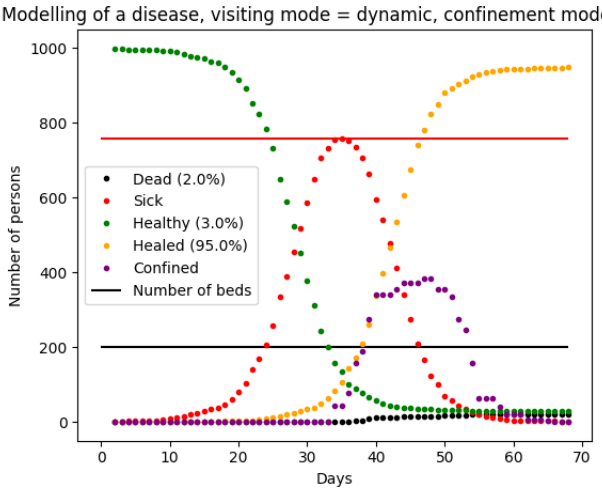

FIGURE 11 – Simulation sur un graphe mixte avec les tests sur les proches des défunts et la mise en quarantaine faible

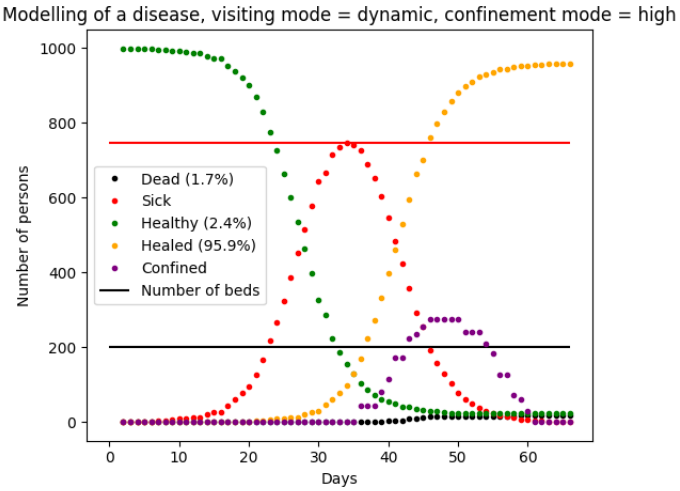

FIGURE 12 – Simulation sur un graphe mixte avec les tests sur les proches des défunts et la mise en quarantaine forte

On observe bien sur les figures 11 et 12 que moins de personnes sont confinées en comparaison avec le scénario précédent (cf 2.2.1 courbe violette). De même, nous avons la confirmation que les personnes en contact avec le défunt mais qui n'était pas malades n'ont pas été confinées sont donc tombées malade par une autre intermédiaire (fin de la courbe verte) : il ne reste que très peu de gens non infectés à la fin de la maladie.
On voit également que l'immunité de groupe fait effet ici pour arrêter l'épidémie.

2.2.3 Tests aléatoires et mise en quarantaine

Dans la partie tests aléatoires, on passe enfin à un scénario de prévention plus concret. On confine des personnes **avant** qu'une personne ne décède.
On teste donc un échantillon aléatoire de n' $< n$ personnes chaque jour (valeur par défaut : $n' = \frac{n}{5}$). Si on détecte une personne malade, alors elle est mise en quarantaine. Examinons les résultats obtenus par ces tests aléatoires.

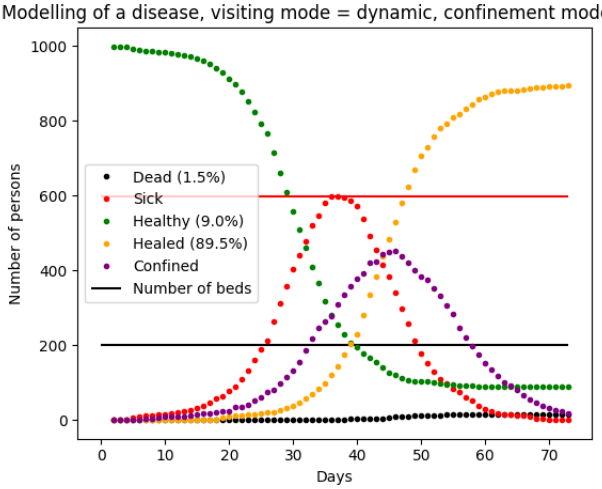

FIGURE 13 – Simulation sur un graphe mixte avec tests aléatoires et la mise en quarantaine faible

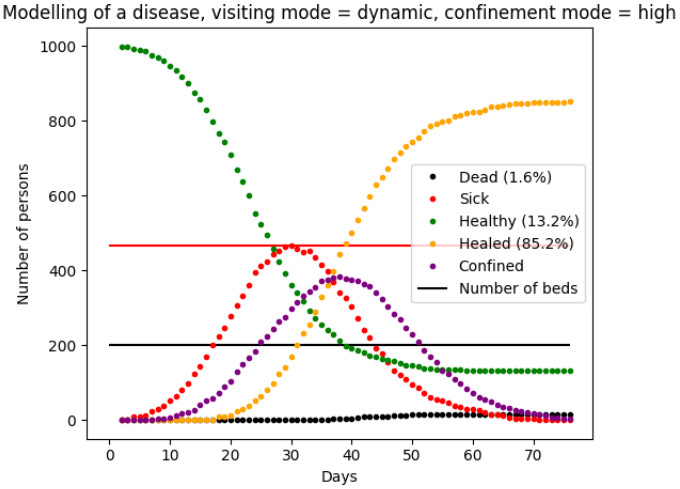

FIGURE 14 – Simulation sur un graphe mixte avec tests aléatoires et la mise en quarantaine forte

On note tout d'abord que les tests aléatoires permettent une mise en quarantaine efficace des personnes testées, comme en témoignent les figures 13 et 14. Le pic de personnes malades (courbe rouge) a très nettement été aplati. Pour les deux figures, la quarantaine s'étend du premier jour jusqu'au 70^{eme}, peu de temps avant la fin de l'épidémie. De plus, malgré un pic d'épidémie toujours élevé pour les deux modes de quarantaine (entre 400 et 600 personnes infectées), le nombre de personnes infectées diminue rapidement dès le 30^{eme} jour. Enfin, on pourrait noter que le taux de décès a diminué, mais comme expliqué précédemment on ne peut pas vraiment comparer le taux de décès. Pour avoir un taux de décès vraiment représentatif il faudrait augmenter la population.
Par ailleurs, on remarque parfois que cette technique de dépistage aléatoire génère parfois des graphiques où la maladie ne se propage plus du tout et est stoppée net dès le début, ce qui la rend efficace. (cf Fig 15 ci-dessous)

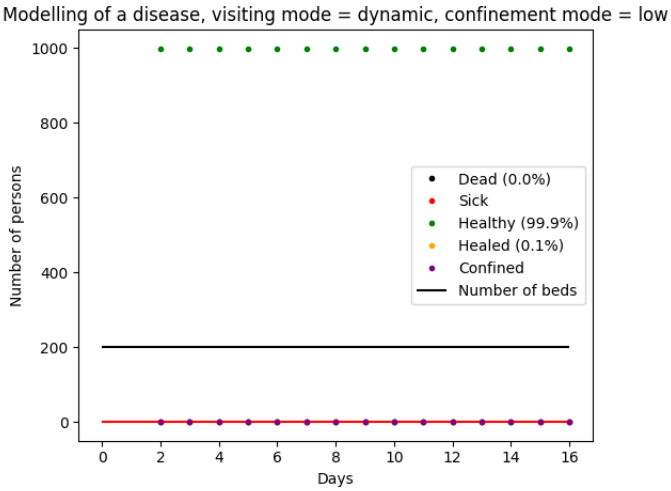

FIGURE 15 – Simulation sur un graphe mixte avec tests aléatoires et la mise en quarantaine faible

2.2.4 Tests massifs

Enfin, le dernier scénario correspond à des tests massifs où l'on combine les deux stratégies de dépistage précédentes : tests sur défunts et tests aléatoires (cf 2.2.2 et 2.2.3). Cela permet de mettre en quarantaine le maximum de personnes de façon à mieux contrôler le virus. Étudions les résultats obtenus pour ces tests finaux.

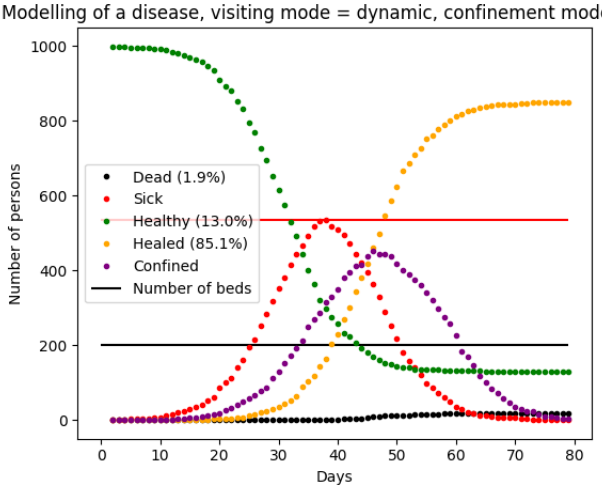

FIGURE 16 – Simulation sur un graphe mixte avec tests massifs et la mise en quarantaine faible

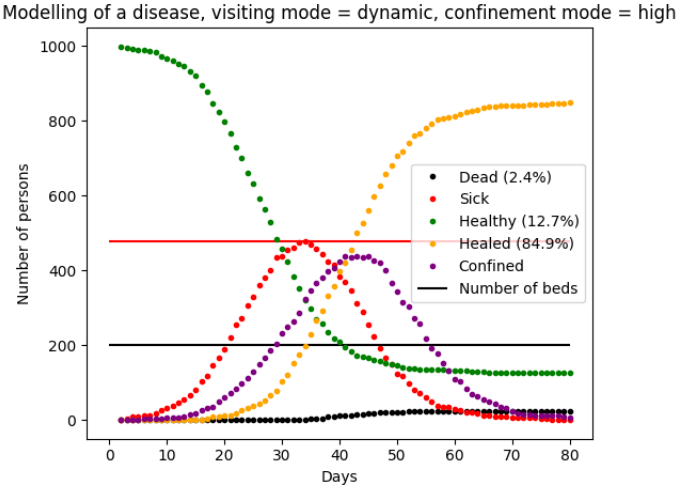

FIGURE 17 – Simulation sur un graphe mixte avec tests massifs et la mise en quarantaine forte

On peut remarquer que les 2 figures obtenues (cf 16 et 17 ne sont pas très éloignées des 2 précédentes (tests aléatoires 2.2.3). En effet, on avait mis en évidence que les tests sur les défunts provoquaient peu de mise en quarantaine. Ainsi, pour ces tests, on observe un nombre important de personnes en quarantaine, s'élevant à 300 vers le 30^{eme} jour. Finalement, l'analyse des résultats est à peu près la même que celle des tests aléatoires.

2.2.5 Conclusion

Par conséquent, il en ressort que chaque scénario mis en place a un impact différent sur la propagation de l'épidémie. Sans modèle évolutif, l'épidémie dure plus de 160 jours et le pourcentage de personnes infectées par le virus atteint 30% le 80^{eme} jour. Avec le modèle évolutif et sans mise en quarantaine, le pourcentage de personnes infectées par le virus atteint rapidement 40% de la population vers le 20^{eme} jour et l'épidémie ne dure plus que 80 jours. Enfin, avec les tests aléatoires, massifs et mise en quarantaine, le taux de décès diminue légèrement et la mise en quarantaine permet de mieux contrôler la propagation du virus.

2.3 Répartition des tâches

2.3.1 Partie code

Implémentation des différentes classes : Marin et Sophie (*)
Implémentation des graphes circulaire, aléatoire et mixte : Marin
Mise en place des quarantaines faible et forte : Marin
Mise en place des tests : Clément
(*) Sophie : refonte et factorisation profonde du code pour faciliter sa lecture et optimiser sa complexité ainsi que faciliter par la suite l'ajout de scénarios/paramètres/options
–> centralisation de l'intelligence dans la classe `World` et modification de la classe `Person` (afin d'en limiter l'usage à la récupération/définition d'informations) ;
–> orientation du graphe et sous-graphe avec une liste d'adjacence de "pères" ;
–> ajout de fonctionnalités permettant de modéliser le décès d'une personne ;
–> refonte de la spécification des paramètres de lancement (paramètres de la maladie, modes de confinement, tests, etc).

2.3.2 Partie rapport

Partie 1 (Introduction) : Clément et Sophie
Parties 2.1.1 et 2.1.2 (Graphe aléatoire et circulaire) : Clément et Sophie
Parties 2.1.3 à 2.2.1 5 (Graphe mixte, conclusion et sans test mais avec mise en quarantaine) : Marin et Sophie
Parties 2.2.2 à 2.2.5 (Tests sur proches des défunts, tests aléatoires, tests massifs et conclusion) : Clément et Sophie