EURIA
Euro-Institut d'Actuariat

Optimisation de positionnement territorial sous contraintes

MASTER 1 Actuariat 2017-2018

Rapport de Bureau d'Étude

Sujet proposé par Optimind Winter

Auteurs :
M. Alexandre Sanzey
M. Alexis Perraud

Encadrants :
M. Nicolas Lorin
M. André Grondin
M. Anthony Nahelou
M. Pierre Aillot

17 mai 2018

Remerciements

Nous tenons à remercier toutes les personnes ayant contribué à notre bureau d'étude.

Nous remercions particulièrement Messieurs Nicolas LORIN et André GRONDIN (Actuaires chez Optimind Winter) qui nous ont accompagnés et ont su être disponibles et à notre écoute tout au long du projet.

Également, nous souhaitons remercier nos tuteurs universitaires Messieurs Anthony NAHELOU (Actuaire indépendant chez Sterenn Actuariat), Pierre AILLOT et Franck VERMET (Enseignants) pour leur suivi, leurs conseils et leur aide qui ont été précieux quant à l'avancée du bureau d'étude.

Table des matières

Table des figures

Introduction

La recherche opérationnelle en matière de localisation d'activités a été un domaine de recherche majeur lors du siècle dernier. La logistique, le transport, la grande distribution, les services bancaires, la poste ou encore l'assurance sont autant de secteurs qui nécessitent une recherche approfondie en matière de localisation. En effet, les arrêts de transport en commun, les agences bancaires comme les bureaux de poste ont besoin d'optimiser l'emplacement de leur réseau d'activités afin d'avoir une couverture maximale sur un territoire donné.

Plus précisément, il est dans leur intérêt de réfléchir à l'emplacement de leurs services, en prenant compte de certains paramètres comme la localisation précise des consommateurs ou la demande, afin d'optimiser leur rentabilité et satisfaire un maximum de clients.

Notre sujet concerne donc l'optimisation de positionnement territorial sous contraintes. Il s'agit, parmi tous les domaines d'applications, de positionner géographiquement des établissements pour les rendre accessibles à un public cible sous contrainte de durée de trajet, de pourcentage de la population touchée, de densité de population (répondant éventuellement à des contraintes relatives à leurs caractéristiques).

L'enjeu est, à partir d'une population et d'un nombre d'unités commerciales donnés, de trouver la meilleure position pour chaque établissement permettant une couverture du territoire et une rentabilité optimale.

Par ailleurs, suite à cette première problématique, une nouvelle nous a été posée : optimiser la couverture territoriale d'un réseau de garages automobiles partenaires d'une compagnie d'assurance dans le cadre de l'assistance automobile.

Il s'agit donc de choisir quels sont les meilleurs garages permettant de constituer un réseau de garages « partenaires » sur un territoire donné, de manière à ce qu'il couvre de façon optimale les pannes de véhicules sur ledit territoire tout en minimisant le temps d'attente des assurés, et de permettre ainsi à chaque garage du réseau de réaliser un nombre minimum d'interventions.

Chapitre 1

Implantation de nouvelles unités commerciales

1.1 Problème p-médian

1.1.1 Un problème d'optimisation territorial sous contraintes

Initialement, le problème d'optimisation de positionnement territorial sous contraintes nous était inconnu. Un travail de recherche a donc été nécessaire afin de comprendre les enjeux du sujet. Nous nous sommes renseignés sur ce qu'était l'optimisation de positionnement territoriale sous contraintes, et une fois l'idée comprise, notre travail fut de rechercher des méthodes de résolution. Très rapidement, le problème p-médian s'est trouvé omniprésent dans nos recherches et nous a semblé répondre en tout point à notre problème.
Le modèle p-médian consiste à trouver les localisations pour un nombre p d'activités devant fournir n clients de telle manière que la somme de l'ensemble des distances séparant chaque activité aux clients les plus proches soit minimale. Ce problème a trouvé son origine au début du XXème siècle grâce aux travaux de l'économiste et sociologue Alfred Weber.

1.1.2 Formulation mathématique du problème p-médian

Il est indispensable de formuler ce problème mathématiquement avant de se lancer dans une quelconque programmation.
Dans un premier temps, nous considérons un réseau constitué de noeuds reliés entre eux par un coût de déplacement d_{ij} (distance entre le noeud i et le noeud j). L'objectif est de trouver un emplacement optimal pour p activités au sein du réseau en prenant soin de minimiser la distance totale entre les p activités et les clients qui leur sont associés.
Il est possible d'ajouter comme dernier paramètre une pondération en tenant compte la

demande a_i au noeud i avec pour objectif de minimiser la somme totale des distances pondérées par la demande.

Le problème p-médian s'écrit donc comme suit :

Minimiser la fonction objective $\sum_i \sum_j a_i d_{ij} x_{ij}$ avec :

$\sum_i x_{ij} = 1 \ \forall i$ qui assure que tous les clients sont assignés à une seule activité,

$x_{ij} \leq y_j \ \forall i, j$ qui empêche d'assigner un client à une activité si elle n'est pas ouverte,

$\sum_j y_j = p$ qui signifie que le nombre total d'activités est p,

$x_{ij}, y_j \in \{0; 1\} \ \forall i, j$ qui indique que x_{ij} et y_j sont des variables binaires,

où :

- a_i est la demande au noeud i (égale à 1 dans le cas non-pondéré),
- d_{ij} est la distance du noeud i au noeud j,
- p est le nombre d'activités à localiser,
- $x_{ij} = 1$ si le noeud i est assigné à l'activité j et 0 autrement,
- $y_j = 1$ si l'activité j est ouverte et 0 autrement.

1.1.3 Complexité algorithmique de la résolution du problème p-médian

La résolution algorithmique du problème p-médian reste limitée. Faisant parti de la classe des problèmes NP-complets, c'est à dire que le nombre de solutions à étudier augmente de façon exponentielle lorsque le nombre de variables augmente, il est nécessaire de s'appuyer sur des méthodes annexes afin de trouver une solution au problème.
Ainsi, le nombre de solutions à étudier est $\frac{n!}{p!(n-p)!}$ où n désigne le nombre de noeuds et p le nombre d'activités à placer.
Par exemple, si l'on voulait placer 10 agences au sein d'un réseau de 100 clients, il y aurait $\frac{100!}{10!90!} = 17310309456440$ de possibilités, soit plus de 17310 milliards de solutions à examiner.
Si l'on dispose d'un ordinateur capable de réaliser 1 million d'opérations à la seconde, il faudrait plus de 17 millions de secondes, c'est à dire environ 200 jours, pour trouver la solution optimale.

Voici un tableau récapitulant les temps nécessaires, avec un ordinateur capable d'effectuer 1 million d'opérations par secondes (ordinateur classique), pour trouver la solution optimale du problème p-médian :

n (noeuds)	p (activités à placer)	temps nécessaire pour trouver la solution
5	2	10^{-5} secondes
10	4	$2,1 \times 10^{-4}$ secondes
20	8	$1,3 \times 10^{-1}$ secondes
30	10	30 secondes
40	10	14 minutes et 7 secondes
50	10	2 heures et 51 minutes
70	10	plus de 4 jours
100	10	plus de 200 jours
100	15	plus de 8000 ans

1.1.4 Problème 1-médian

Théorie

La résolution du problème 1-médian est le cas de figure le plus simple. Il s'agit de placer une unique activité au sein d'un réseau constitué de n noeuds.
Nous rappelons que a_i est la demande au noeud i. La résolution du problème 1-médian consiste à dire que si la demande en un noeud du réseau est supérieure à la moitié de la demande totale, soit $\sum_i a_i$, alors ce noeud est la solution optimale. De plus, celle-ci est unique.
Pour ce, selon les travaux de Goldman, il suffit de choisir le noeud ayant la plus faible demande parmi tous les noeuds situés à l'extrémité du réseau. Si ce noeud concentre moins de la moitié de la demande totale alors il est supprimé et sa demande est reportée sur le noeud suivant. Ce processus est réalisé jusqu'à obtenir un noeud concentrant plus de la moitié de la demande qui sera la solution optimale.

Exemple

Dans ce paragraphe, nous allons détailler, à travers un exemple, le processus décrit ci-dessus.
Le réseau est composé de 5 noeuds. À chaque noeud i est associé une demande h_i. La demande totale est égale à $\sum_i h_i = 48$.

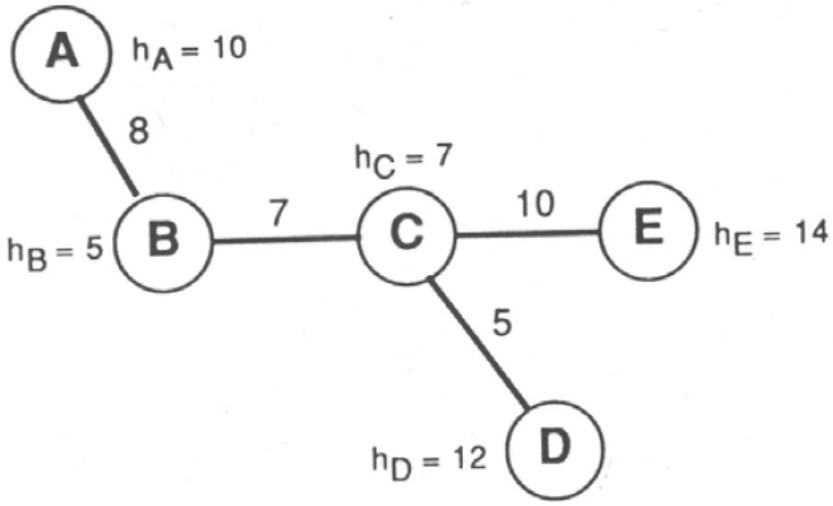


FIGURE 1.1 – Résolution du problème 1-médian : étape 1

Ici, aucun des noeuds ne concentre plus de la moitié de la demande (égale à 24). On choisit donc le noeud ayant la plus faible demande parmi les noeuds situés à l'extrémité du réseau : le noeud A. Sa demande étant égale à 10, on le supprime et on la reporte sur le noeud suivant, à savoir le noeud B qui a désormais une demande égale à 15.

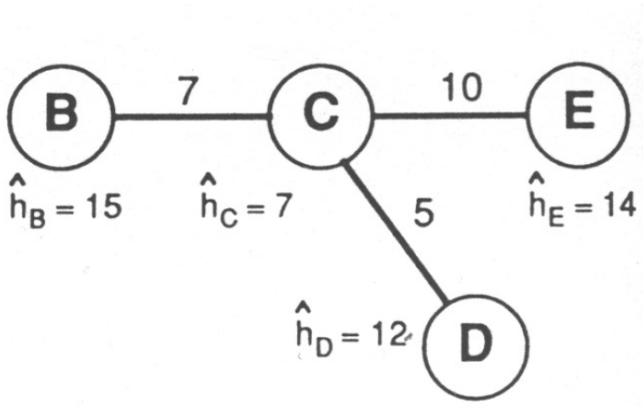


FIGURE 1.2 – Résolution du problème 1-médian : étape 2

Aucun noeud ne concentre plus de la moitié de la demande. On choisit donc le noeud D qui concentre la plus faible demande (égale à 12). Enfin, on réédite le processus : le point D est supprimé et sa demande est reportée au noeud C. Le point C a désormais une demande égale à 19.

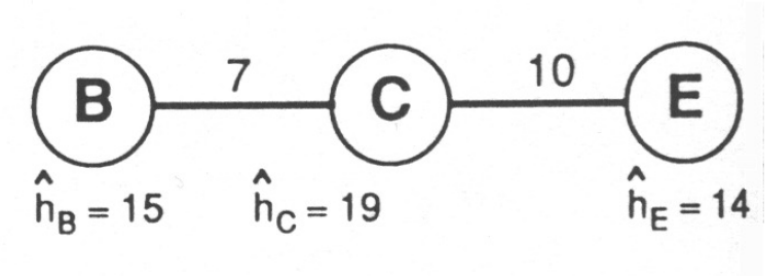


FIGURE 1.3 – Résolution du problème 1-médian : étape 3

Comme il n'y a toujours pas de noeud concentrant plus de la moitié de la demande, on supprime le point E. Sa demande est reportée au noeud C.

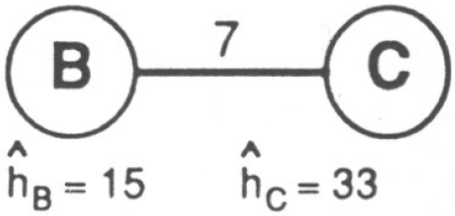


FIGURE 1.4 – Résolution du problème 1-médian : étape 4

La demande au noeud C devient supérieure à la moitié de la demande totale, il est donc le point optimal. L'activité doit être placée au noeud C.

Programmation

Nous avons voulu mettre en oeuvre le problème 1-médian à l'aide du logiciel R. Dans ce paragraphe, il sera présenté les différentes étapes et les programmes que nous avons réalisés.

Premièrement, nous avons créé une carte dans laquelle nous avons placé aléatoirement des points. Chacun des points représente un noeud du réseau. Comme la demande n'est pas la même en chaque point du réseau, nous leur avons ajouté aléatoirement un poids. Ainsi, plus le poids est élevé, plus la demande est forte.

Ci-dessous, une simulation d'un réseau de 100 noeuds :

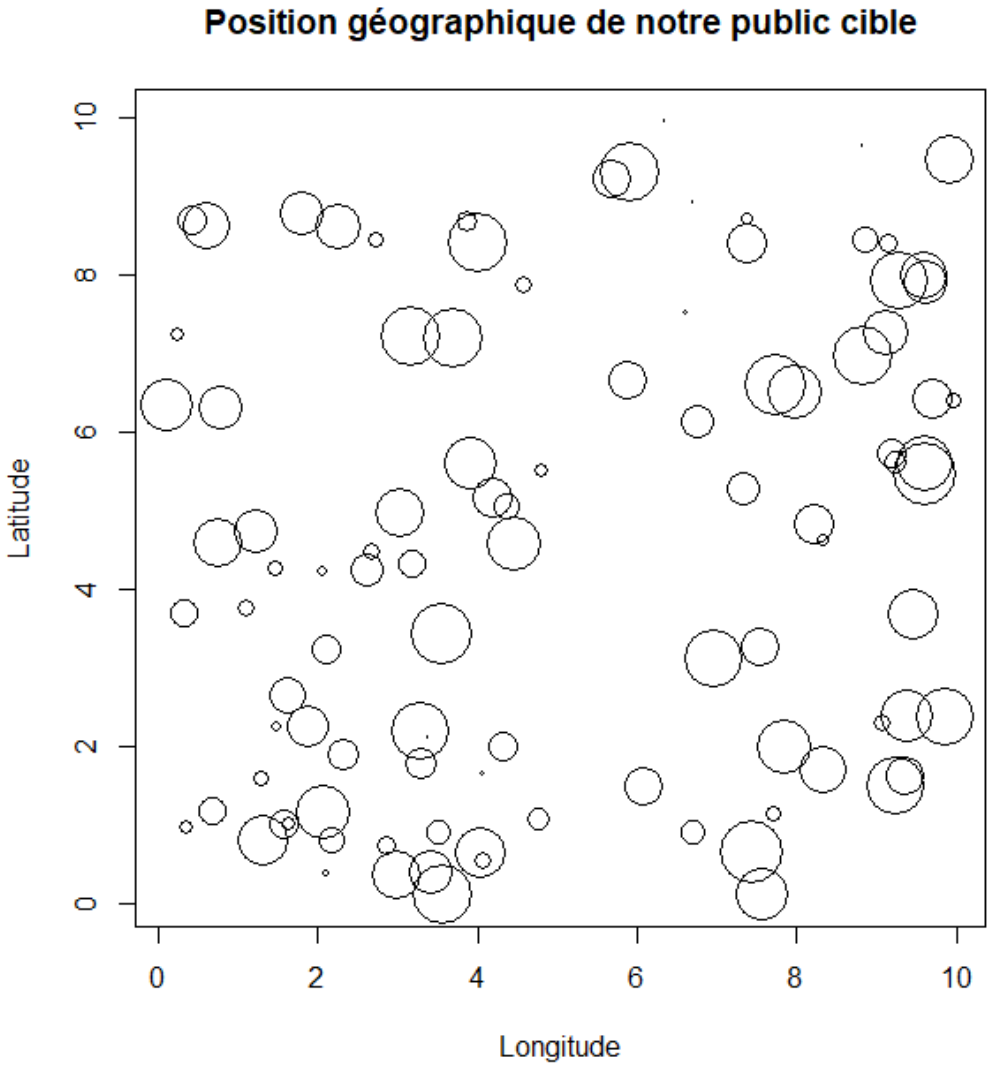


FIGURE 1.5 – Simulation d'un réseau de 100 noeuds

Ensuite, nous avons réalisé un premier algorithme. Nous partons d'une carte comportant, par exemple, 100 noeuds. Nous décidons que les extrémités du réseau sont les points situés aux extrémités des abscisses et des ordonnées : 4 points sont donc considérés comme étant à l'extrémité du réseau. Si aucun des noeuds ne possède une demande supérieure ou égale à la moitié de la demande totale, les 4 points sont successivement supprimés. Leurs poids sont reportés sur les deuxièmes points les plus à l'extrémité. Plus précisément, le noeud à l'abscisse la plus faible est supprimé et sa demande est reportée sur le point ayant la deuxième plus petite abscisse. Il se passe le même procédé pour les trois autres noeuds (tout en vérifiant si la demande d'un point n'a pas dépassé la moitié de la demande totale). Ce processus est réalisé tant que la demande en chaque point est inférieure à la moitié de la demande totale. La position optimale est la position du premier noeud dont sa demande atteint la moitié de la demande totale.

L'exemple, ci-dessous, montre l'évolution entre chaque tour de boucle :

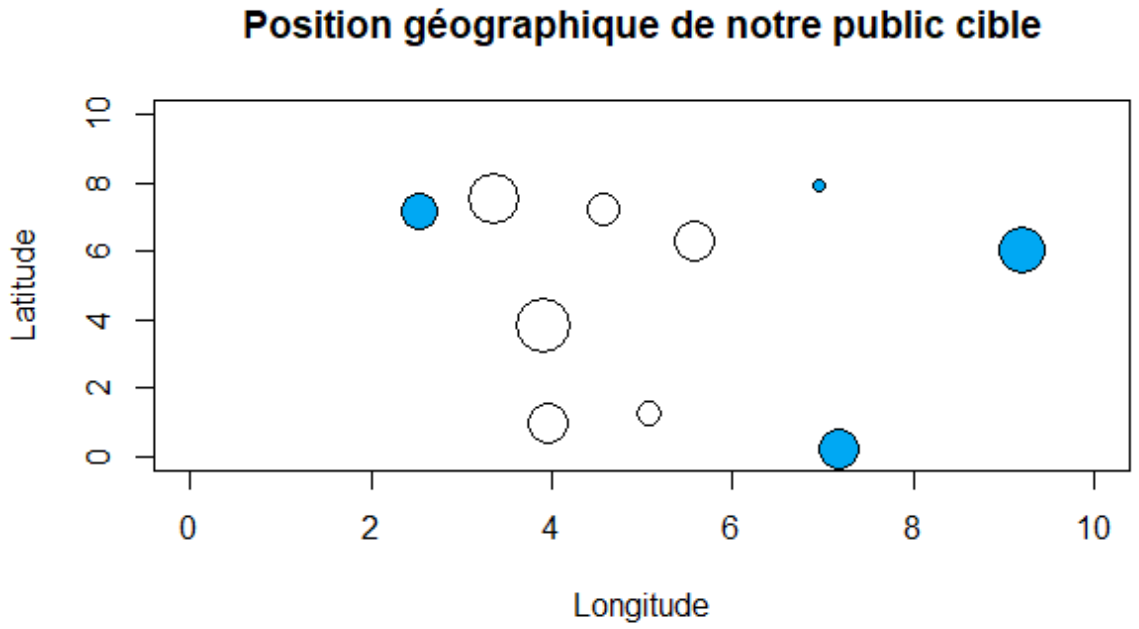


FIGURE 1.6 – Position géographique de notre public cible

Lors du premier tour de boucle, les 4 points situés aux extrémités (points bleus (fig. 1.6)) sont supprimés et les poids sont reportés sur les deuxièmes noeuds les plus à l'extrémité. (fig. 1.7).

Désormais, aucun des noeuds ne concentre plus de la moitié de la demande totale, un deuxième tour de boucle est exécuté. Lors de celui-ci, un seul point (point bleu (fig. 1.7)) et non 4 ont été supprimés car l'un des noeuds a atteint une demande supérieure à la moitié de la demande totale. (fig. 1.8).

Enfin, le point dont le poids a dépassé la moitié de la demande totale (point rouge) est la position optimale trouvée. (fig. 1.9).

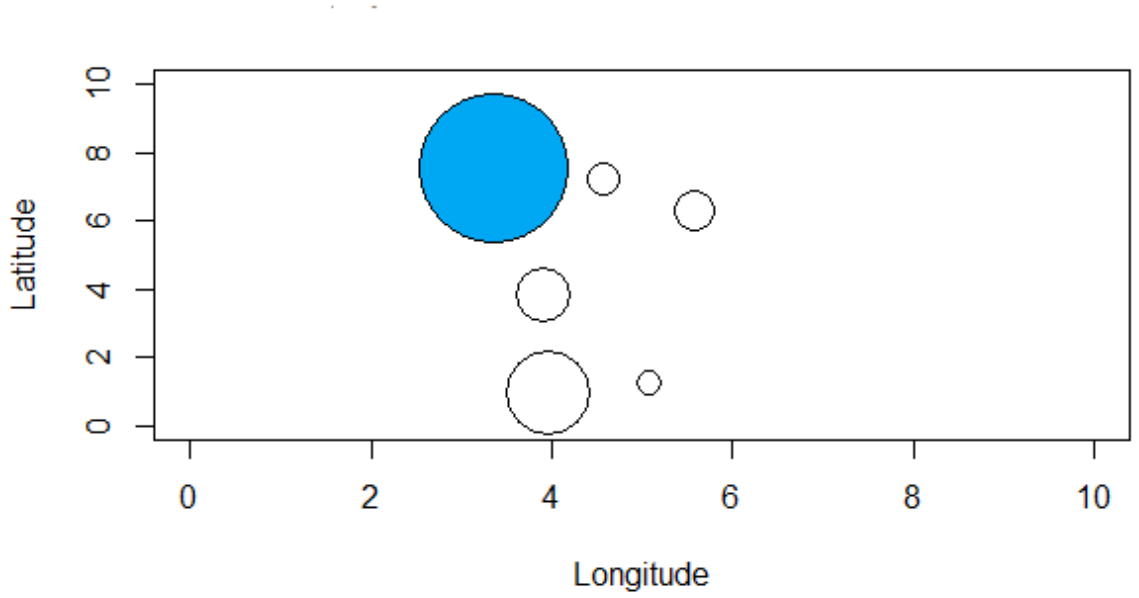


FIGURE 1.7 – Évolution après le premier tour de boucle

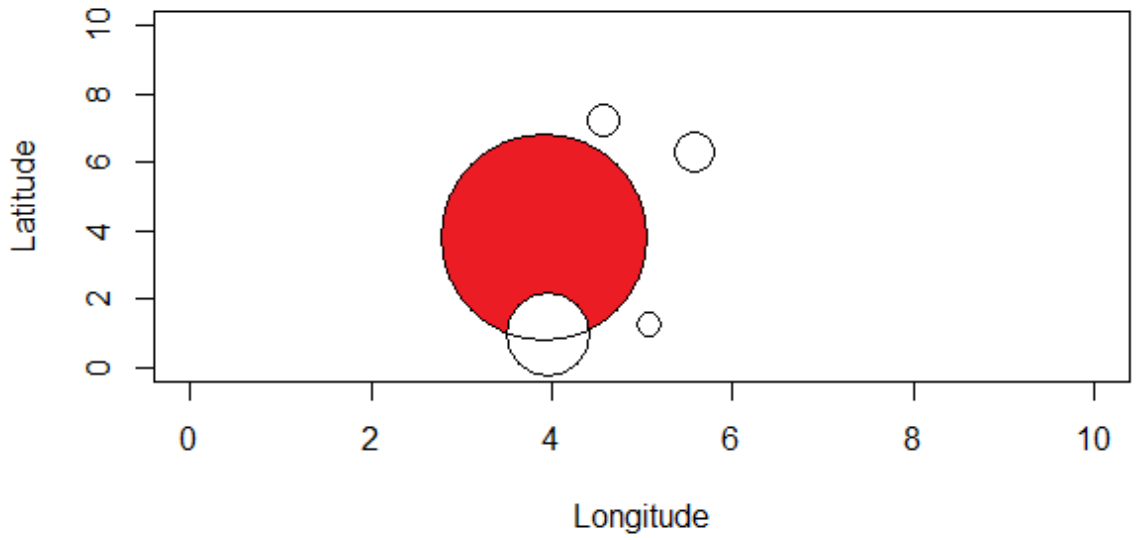


FIGURE 1.8 – Évolution après le deuxième tour de boucle

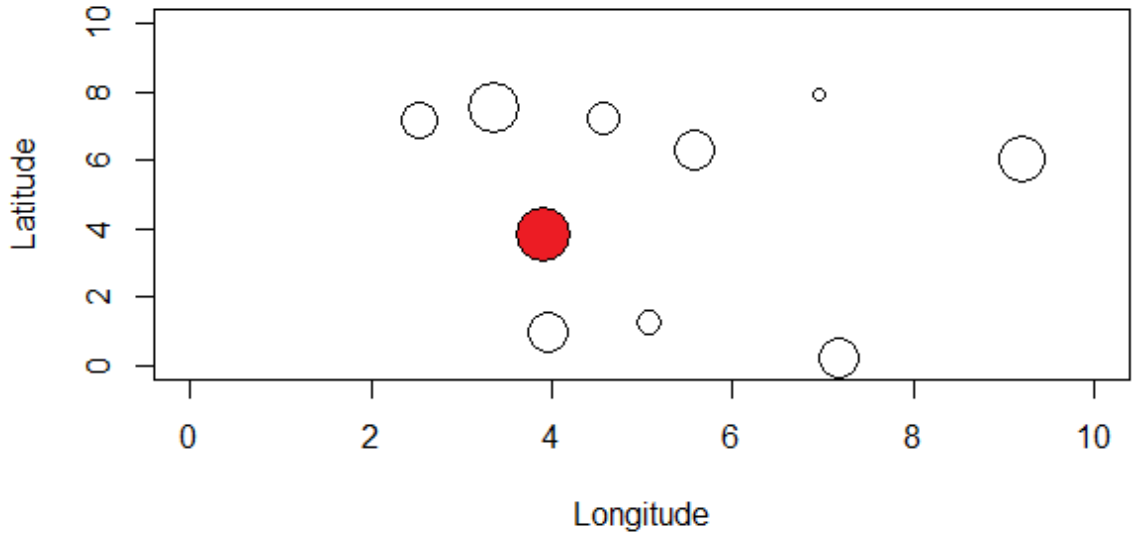


FIGURE 1.9 – Position optimale

Cependant, dans ce premier algorithme, nous traitons les 4 points à l'extrémité du réseau dans un même tour de boucle. Le poids de ces 4 noeuds n'est donc pas pris en compte. C'est pourquoi nous avons modifié notre fonction afin que, lors de chaque tour de boucle, seul le point ayant la plus faible demande soit traité.

L'exemple, ci-dessous, montre l'évolution entre chaque tour de boucle :

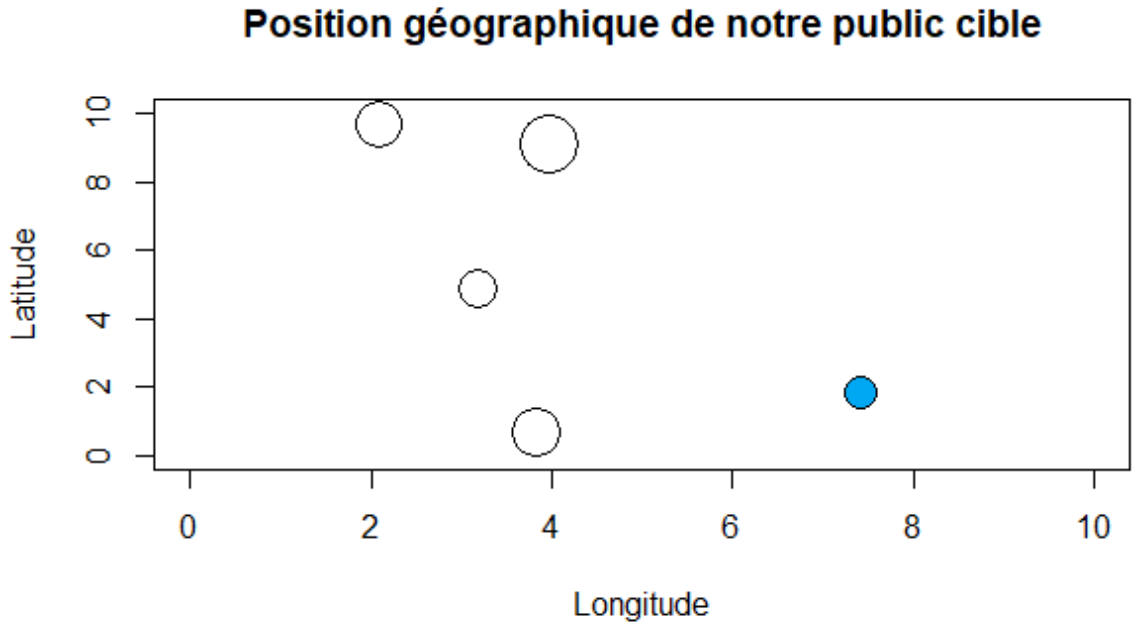


FIGURE 1.10 – Position géographique de notre public cible

Lors du premier tour de boucle, on remarque que seul le point, parmi les points situés aux extrémités, ayant la plus faible demande est traité (point bleu (fig. 1.10)).

Tant qu'il n'y a pas de noeud concentrant plus de la moitié de la demande, le processus reste le même. ((fig. 1.11), (fig. 1.12)).

Le point concentrant plus de la moitié de la demande (fig. 1.13) est la position optimale trouvée. (fig. 1.14).

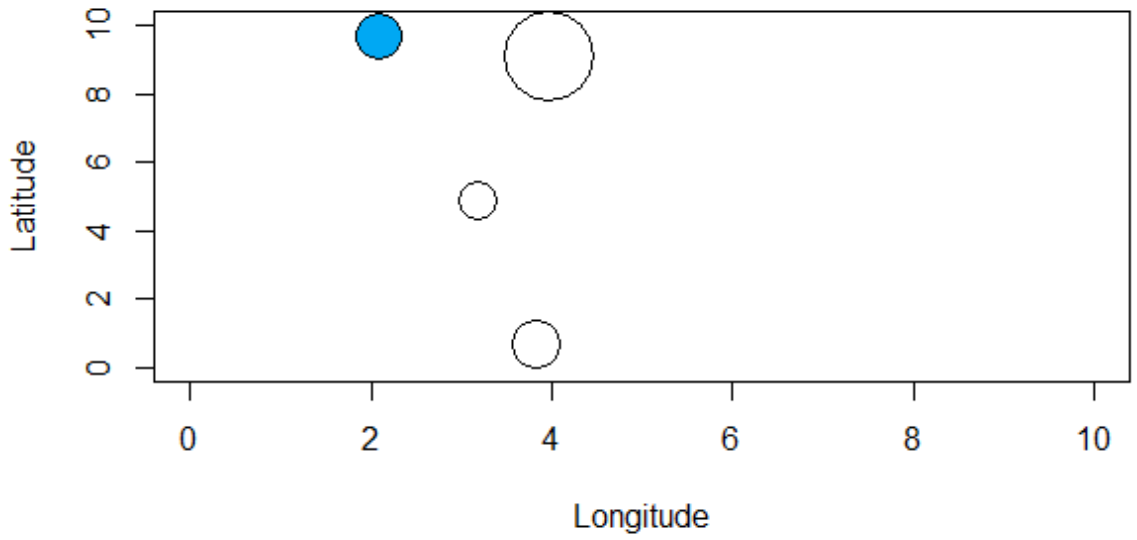


FIGURE 1.11 – Évolution après le premier tour de boucle

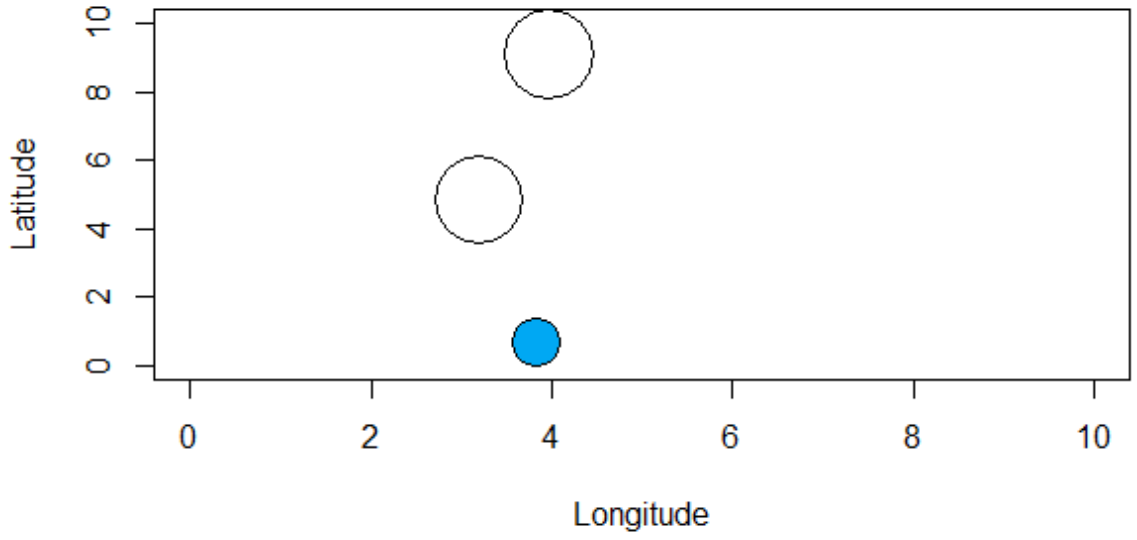


FIGURE 1.12 – Évolution après le deuxième tour de boucle

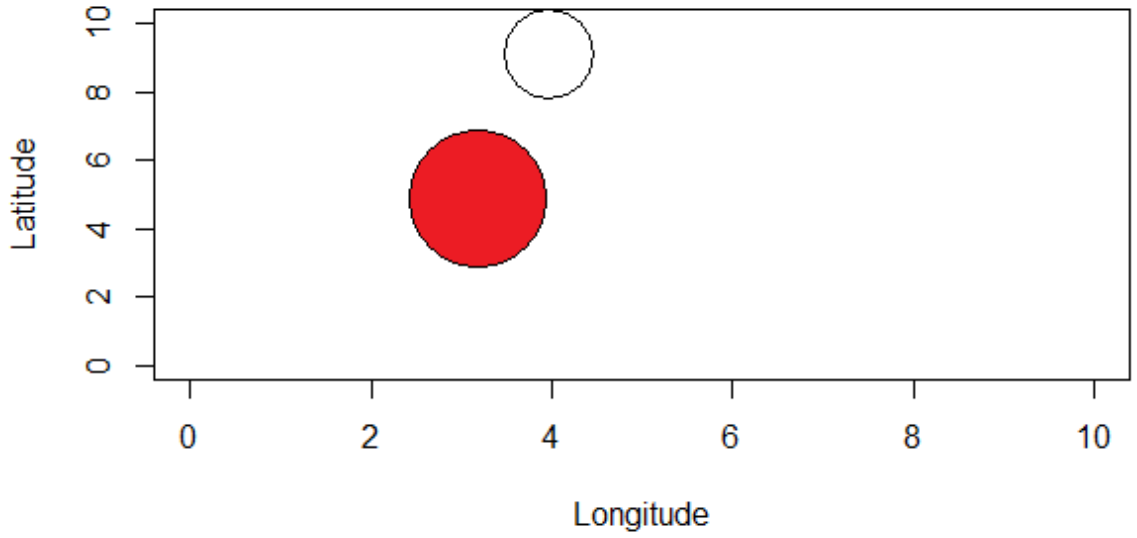


FIGURE 1.13 – Évolution après le troisième tour de boucle

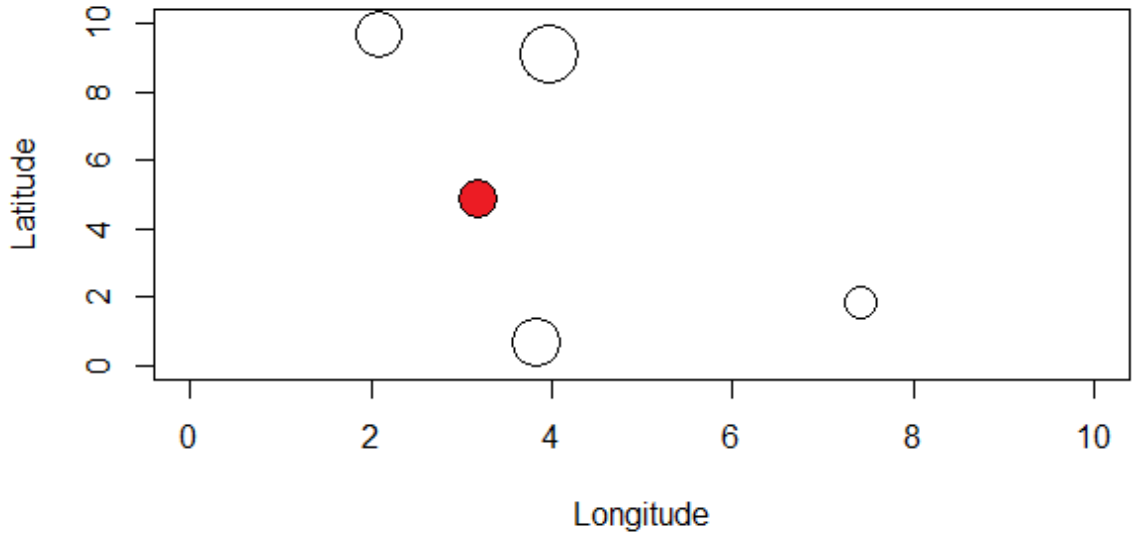


FIGURE 1.14 – Position optimale

Nous avons cependant noté un nouveau problème dans ce second algorithme : quand un point est supprimé, sa demande n'est pas obligatoirement reportée sur le point le plus proche.
Pour pallier cela, nous avons appelé la fonction dist (distance) sur R. Elle permet de calculer les distances entre tous les points et elle les répertorie dans une matrice. Ainsi, quand un point est supprimé, son poids est désormais reporté sur le point le plus proche.

Ci-dessous, un exemple montrant la différence entre la nouvelle programmation et l'ancienne :

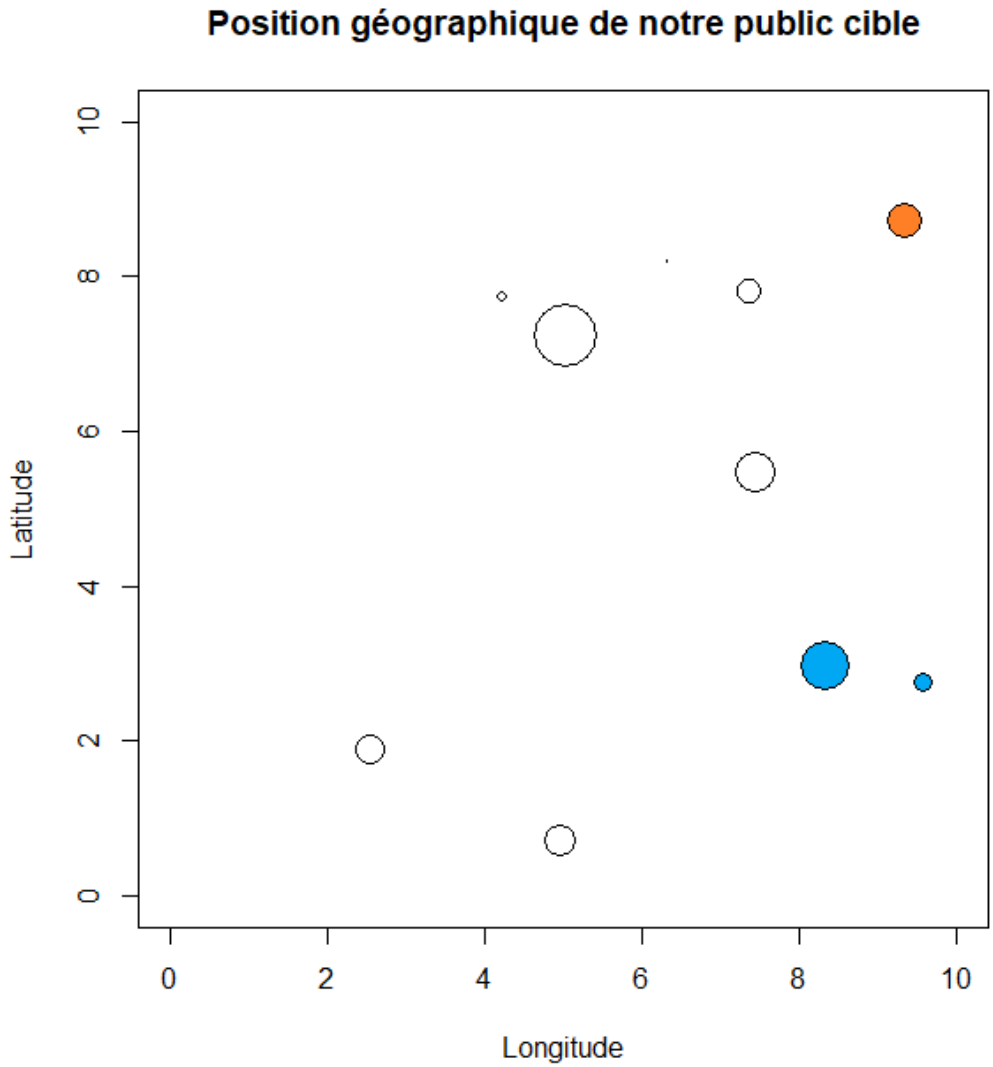


FIGURE 1.15 – Position géographique de notre public cible

Dans un premier temps, le noeud du réseau traité est celui qui concentre la plus petite demande parmi les points situés aux extrémités. Le point supprimé voit donc sa demande reportée sur le point le plus proche (points bleus) et non sur le deuxième le plus à l'extrémité (point orange). (fig. 1.15).

On obtient donc cette carte après le premier tour de boucle :

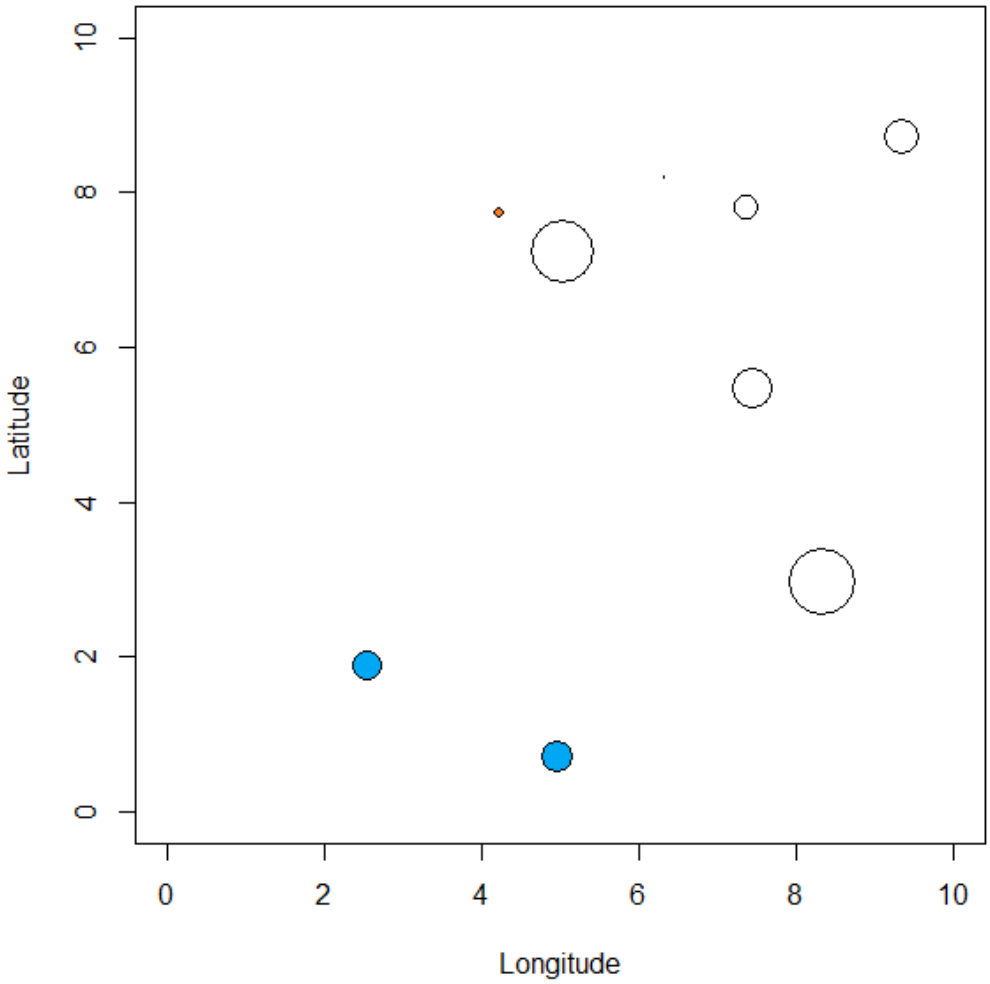


FIGURE 1.16 – Évolution après le premier tour de boucle

Lors du second tour de boucle, une situation similaire se présente. Le noeud (bleu) à l'extrémité gauche est supprimé et sa demande est reportée sur le point le plus proche (second point bleu) et non sur le noeud orange. (fig. 1.16).

Ci-dessous, la carte obtenue après le second tour de boucle :

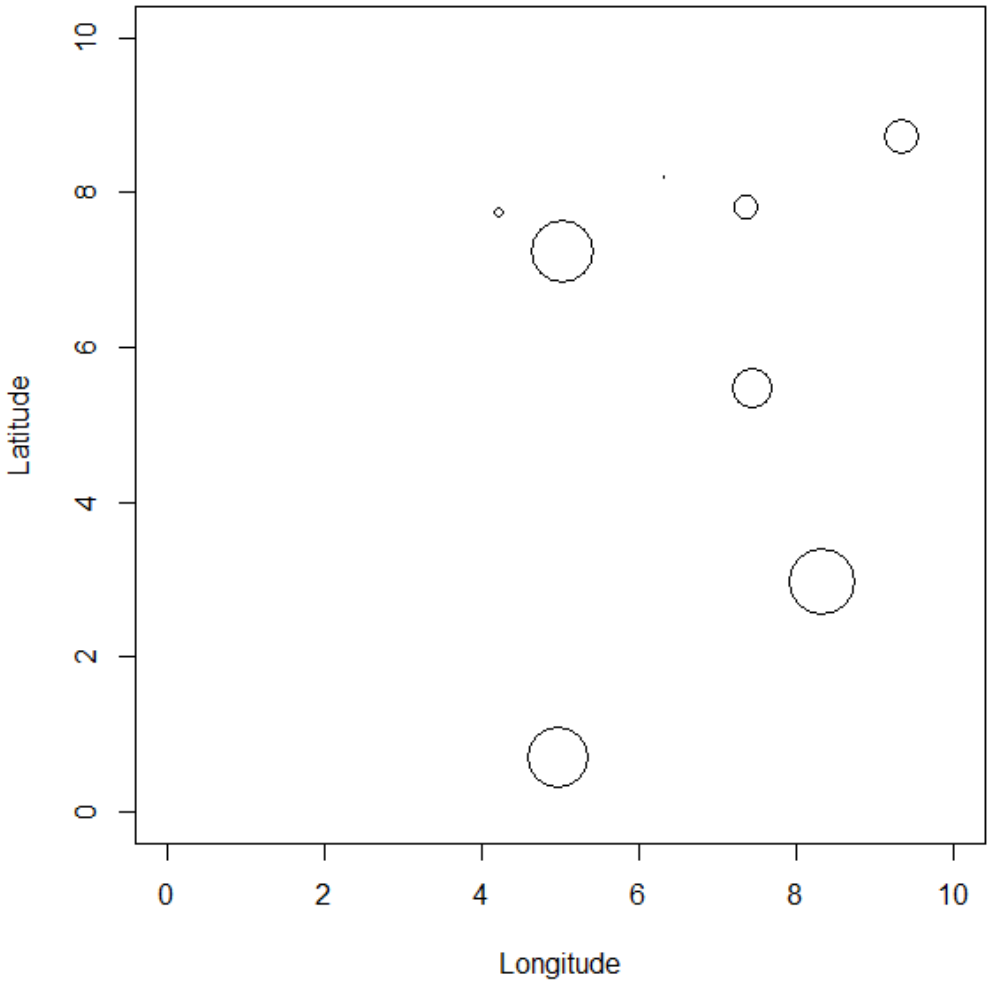


FIGURE 1.17 – Évolution après le second tour de boucle

Finalement, après plusieurs tour de boucle, la position optimale trouvée est :

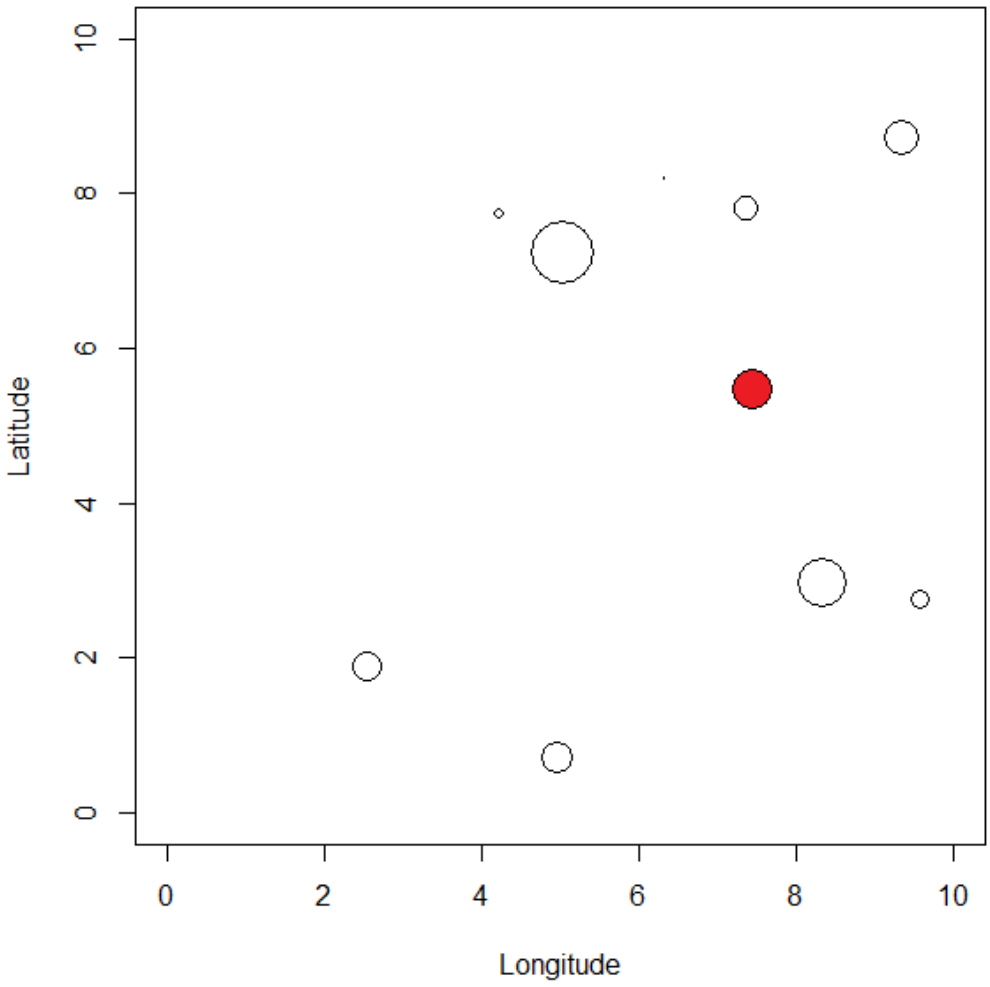


FIGURE 1.18 – Position optimale

Ci-dessous, la position optimale trouvée, sur une simulation d'un réseau de 100 noeuds, à l'aide de notre algorithme 1-médian final :

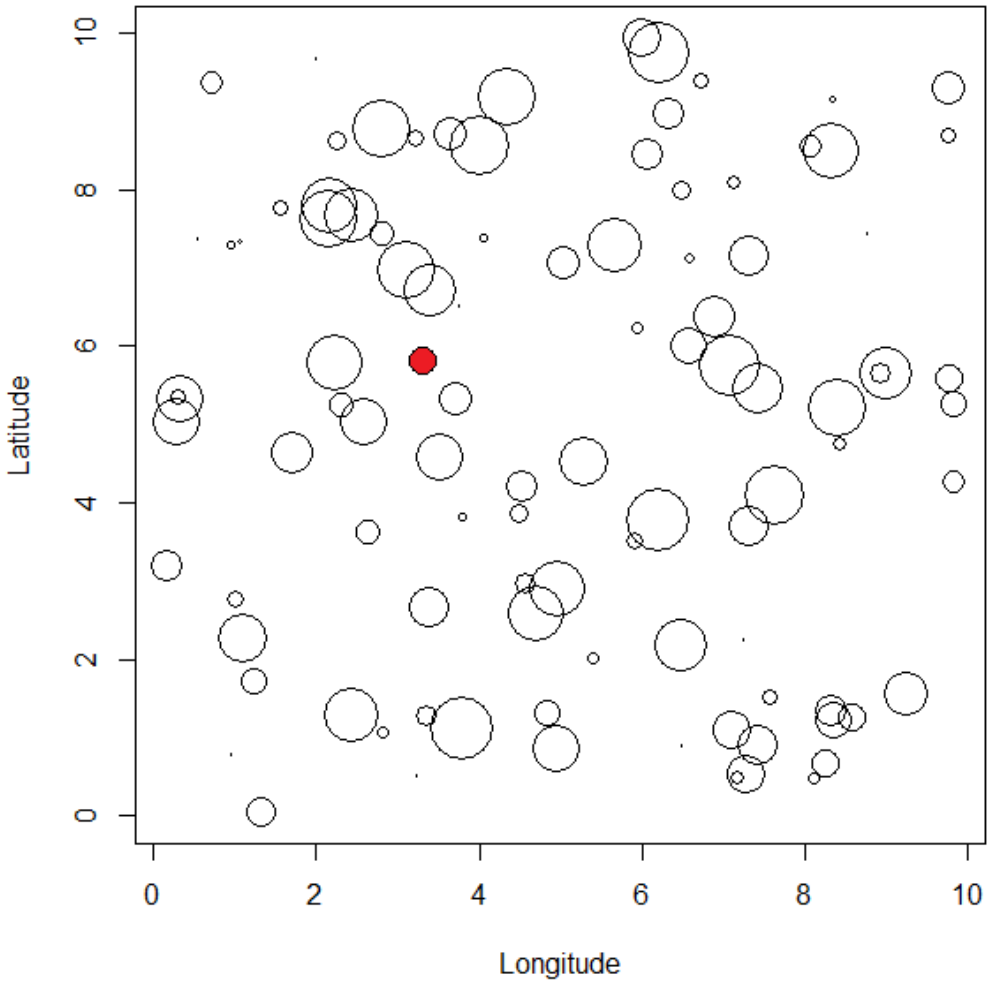


FIGURE 1.19 – Résolution du problème 1-médian sur une simulation d'un réseau de 100 noeuds

1.2 Application à des données réelles

Nous avons voulu mettre en application les différents programmes algorithmiques décrits jusqu'à maintenant sur des données réelles.

1.2.1 Données

Nous avons donc utilisé une base de données trouvée sur Internet en Open Data nommée villes_france.csv contenant 36701 lignes et 6 colonnes. Chaque ligne représente une commune de France. Chaque colonne correspond à une variable décrivant les villes de France :

— un numéro entre 1 et 36701 (identifiant)
— le numéro de département
— le nom de la commune
— la population
— la latitude
— la longitude

Après avoir nettoyé ces données afin de les rendre exploitables, nous avons voulu dans un premier temps nous restreindre à des sous groupes de cette base de données pour des raisons de complexité algorithmique.

1.2.2 Résolution algorithmique

Rhône

Nous avons tout d'abord restreint notre étude au département du Rhône. Suite à l'utilisation de la fonction "unmedian4" sur le dataframe "Rhone", nous avons été capable de trouver une solution optimale : Écully, une commune proche de Lyon. (fig. 1.20).

Île-de-France

Dans un second temps, nous avons voulu travailler sur un territoire plus grand qu'un département. Nous avons donc choisi une région : l'Île-de-France. Par le même procédé que pour la résolution de ce problème sur le département du Rhône, le résultat obtenu est la ville de Vanves, au sud de Paris. (fig. 1.21).

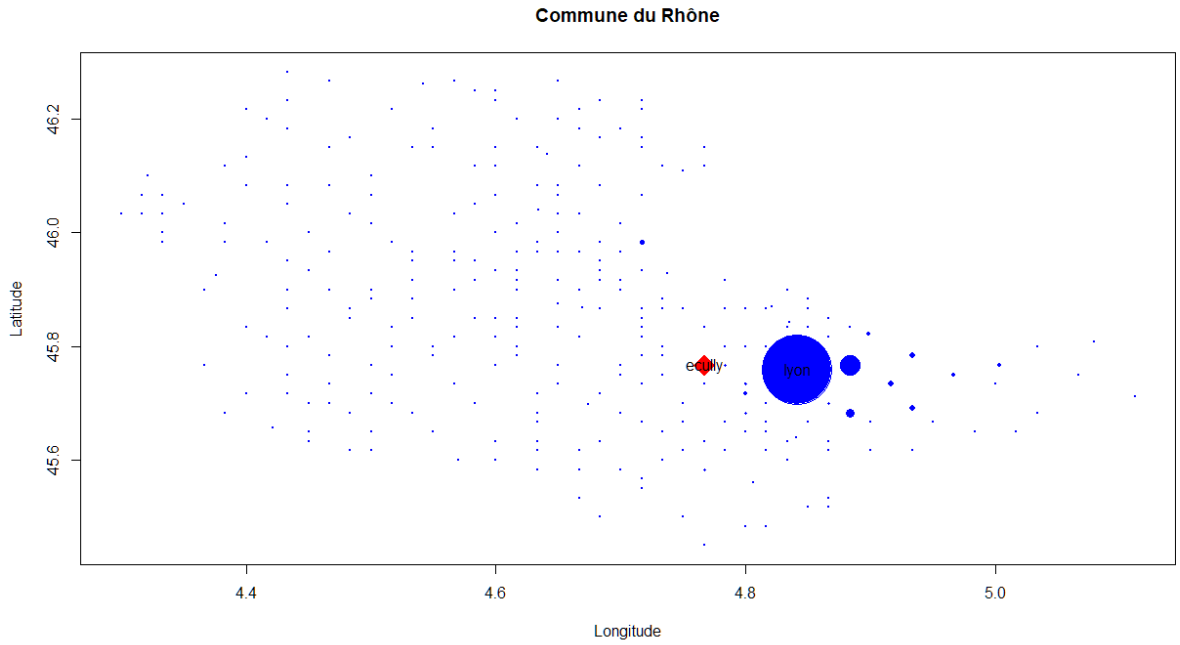


FIGURE 1.20 – Résolution du problème 1-médian sur le département du Rhône

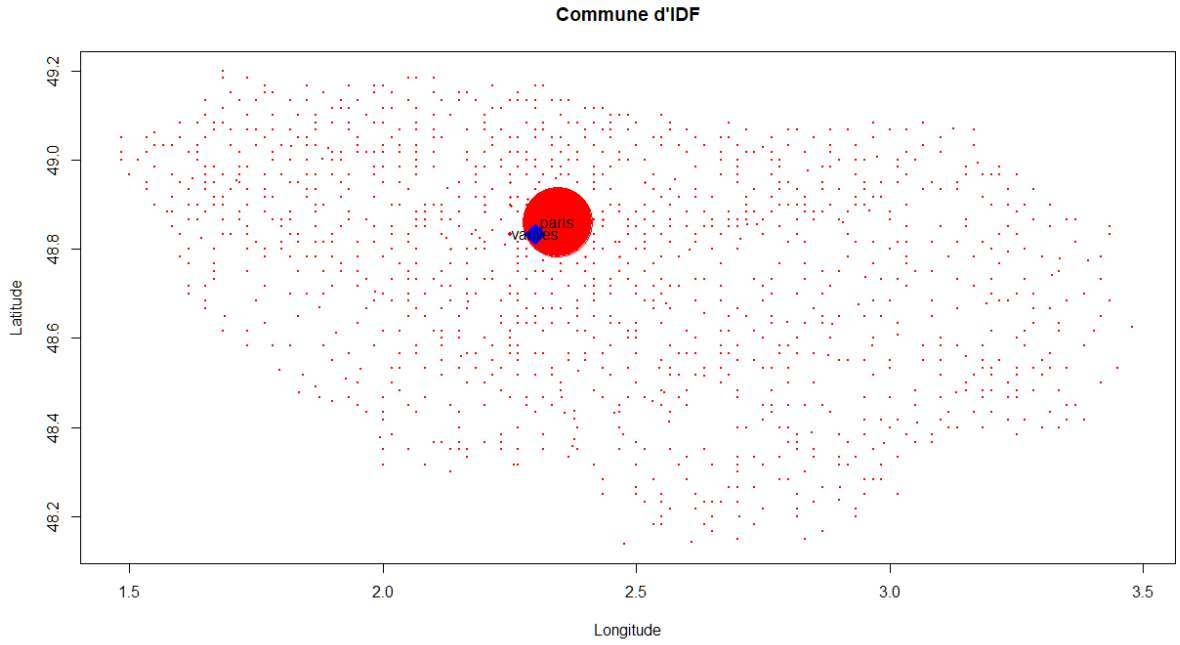


FIGURE 1.21 – Résolution du problème 1-médian sur la région Île-de-France

France

Pour des raisons de complexité algorithmique, il n'est pas possible de résoudre ce problème sur la totalité de la France de la même manière que dans les deux cas précédents. Nous avons donc décidé de partitionner la France en différentes zones à l'aide de l'algorithme du K-means, puis d'appliquer l'algorithme 1-médian à chacune de ces zones. Ce processus est intéressant car dans l'énoncé inital de notre problème, nous cherchions à avoir un nombre entier "p" d'unités commerciales à implanter. Ainsi, il suffit de faire coïncider le nombre d'unités commerciales "p" recherchées avec le nombre "k" de l'algorithme du K-means. De cette manière, nous allons obtenir "p" zones dans un premier temps avec l'algorithme du K-means ; puis, nous obtiendrons une unité commerciale pour chacune de ces zones après l'application de l'algorithme "unmedian4", ce qui fait un total de "p" unités commerciales.

Dans les deux exemples, ci-dessous, ce procédé a été utilisé pour k = 6 (fig. 1.22) puis k = 50 (fig. 1.23). Les différentes zones sont affichées sous différentes couleurs pour pouvoir les différencier et la localisation des unités commerciales est marquée d'un point jaune.

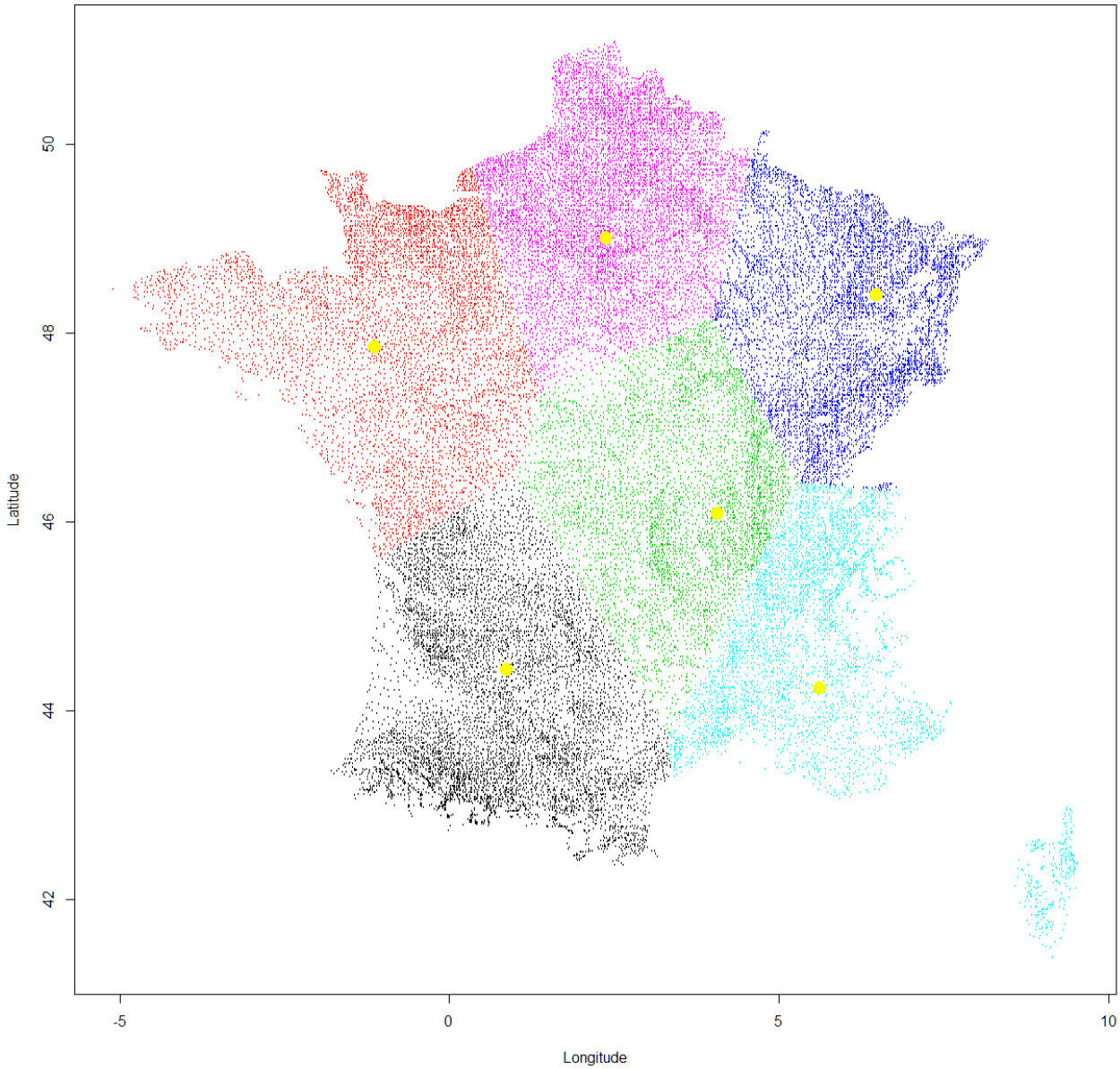


FIGURE 1.22 – Résolution sur la France avec k = 6

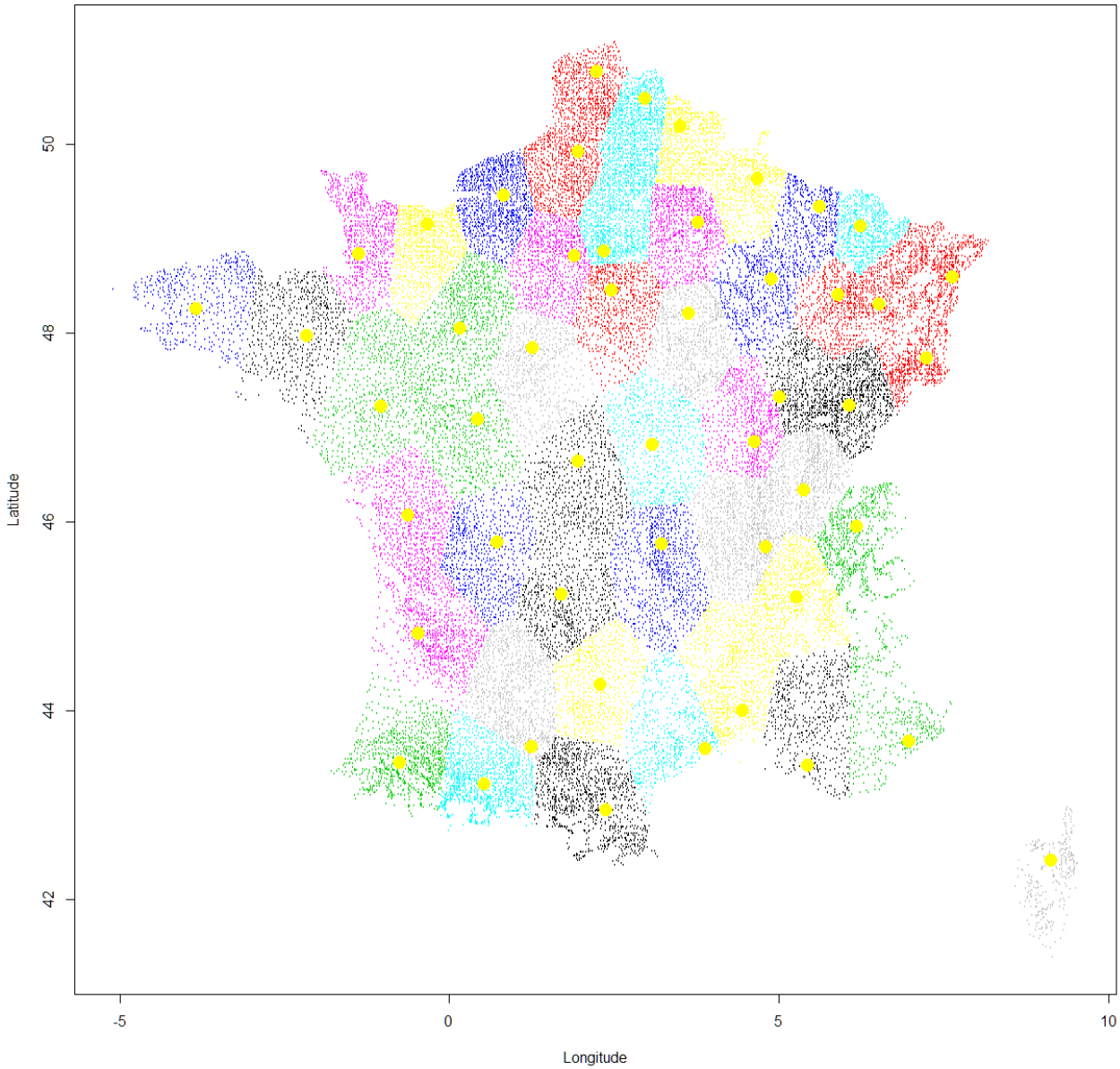


FIGURE 1.23 – Résolution sur la France avec k = 50

1.3 Limites

Pour des raisons de complexité algorithmique, nous avons uniquement utilisé la résolution du problème 1-médian, et non celle du problème p-médian.

Ce programme pourrait encore être amélioré, par exemple en utilisant les temps de trajet routiers plutôt que la distance à vol d'oiseau (distance euclidienne).

L'algorithme du k-means utilisé pour faire les zones ne prend pas en considération la population des villes, ce qui peut amener à avoir des zones hétérogènes.

L'utilisation de zones permet d'avoir une résolution de ce problème, mais de manière indépendante entre chaque zone, et ne prend donc pas en compte les zones voisines.

Chapitre 2

Optimisation d'un réseau de garages partenaires

Pour la gestion de ses contrats d'assurance automobile, plus particulièrement pour l'assistance automobile, une compagnie d'assurance travaille avec des garages automobiles partenaires. Elle souhaite optimiser sa couverture territoriale, et donc ses partenariats avec ces garages. Elle peut recruter de nouveaux garages partenaires, et elle peut en supprimer certains. Ainsi, l'objectif est qu'elle pourra dépanner plus vite ses clients si son réseau de partenaires est correctement implanté sur le territoire, ce qui correspond à un gain financier pour la compagnie d'assurance, et à un gain de satisfaction pour l'assuré.

Mais qu'est-ce que c'est qu'un garage automobile partenaire ? Qu'a-t-il de particulier ?

Il s'agit d'un garage qui a passé un accord avec une compagnie d'assurance pour offrir à ses assurés quelques avantages pour la prise en charge de leur véhicule en cas de sinistre :

— pas d'avance des frais de réparation ;
— prise en charge plus rapide du véhicule ;
— disponibilité pouvant aller jusqu'à 24h/24 et 7j/7 ;
— gestion des questions d'assistance ;
— compétences techniques nécessaires pour la réparation sur place ;
— tarifs beaucoup plus intéressants ;
— démarches administratives simplifiées.

Quel intérêt a ce garage à passer ce type d'accord avec une compagnie d'assurance ?

En conséquence du nombre important d'assurés que possède la compagnie d'assurance, le garage, par cet accord, gagne en quantité de travail. En effet, par cet accord, un nombre plus ou moins important de pannes automobiles lui seront attribuées, ce qui lui permet d'avoir une quantité de travail plus importante, et donc de compléter son agenda, ses

heures de travail et ses revenus par rapport à un garage ne passant pas ce type d'accord et qui se repose uniquement sur sa clientèle habituelle. Il peut également bénéficier d'un accompagnement avec des formations techniques et des applications spécifiquement dédiées au dépannage sur place.

Ainsi, pour que la mise en place de ces partenariats garde son intérêt pour les garages, nous avons considéré une contrainte qui consiste à garantir un nombre minimum d'interventions pour chaque garage. Ainsi, un garage partenaire à qui l'on ne peut pas garantir ceci, sera retiré du réseau, et un garage susceptible de devenir partenaire devra être localisé de telle manière à ce que l'on puisse lui garantir ceci afin qu'il intègre le réseau.

Notre objectif est de scorer les garages en fonction de l'historique des lieux de pannes afin de savoir quels garages sont à retirer de notre réseau de garages partenaires, et lesquels sont à recruter.

2.1 Données

2.1.1 Garages

Nous avons pour cela utilisé une base de données au format CSV que l'on a trouvée en libre accès sur internet sur le site www.data.gouv.fr . Cette base de données recense 44 garages automobiles décrits par différentes variables, dont une est primordiale pour notre travail ; les coordonnées GPS de chaque garage.

Après avoir extrait les informations utiles de cette base, c'est-à-dire la latitude et la longitude de chaque garage, nous avons voulu partitionner cette base en deux parties : certains de ces garages sont considérés comme des garages partenaires de notre réseau, quant aux autres, ils sont considérés comme susceptibles de devenir partenaires. Ce procédé permet d'être au plus proche de la réalité, en considérant qu'un assureur possède déjà un réseau contenant des garages partenaires, et que, dans le cadre de l'amélioration de son réseau, il possède une liste de garages susceptibles de devenir partenaires.

2.1.2 Pannes automobiles

N'ayant pas eu de données réelles à exploiter, nous avons simulé aléatoirement des lieux de pannes dans le secteur considéré. Nous considérons que ces lieux de pannes simulés correspondent à un historique des lieux de pannes du passé, connu par la compagnie d'assurance. Ces données ne sont pas représentatives de la réalité car les lieux de pannes ne se trouvent donc pas forcément sur des axes routiers.

Ces lieux de pannes sont représentés par des points verts.

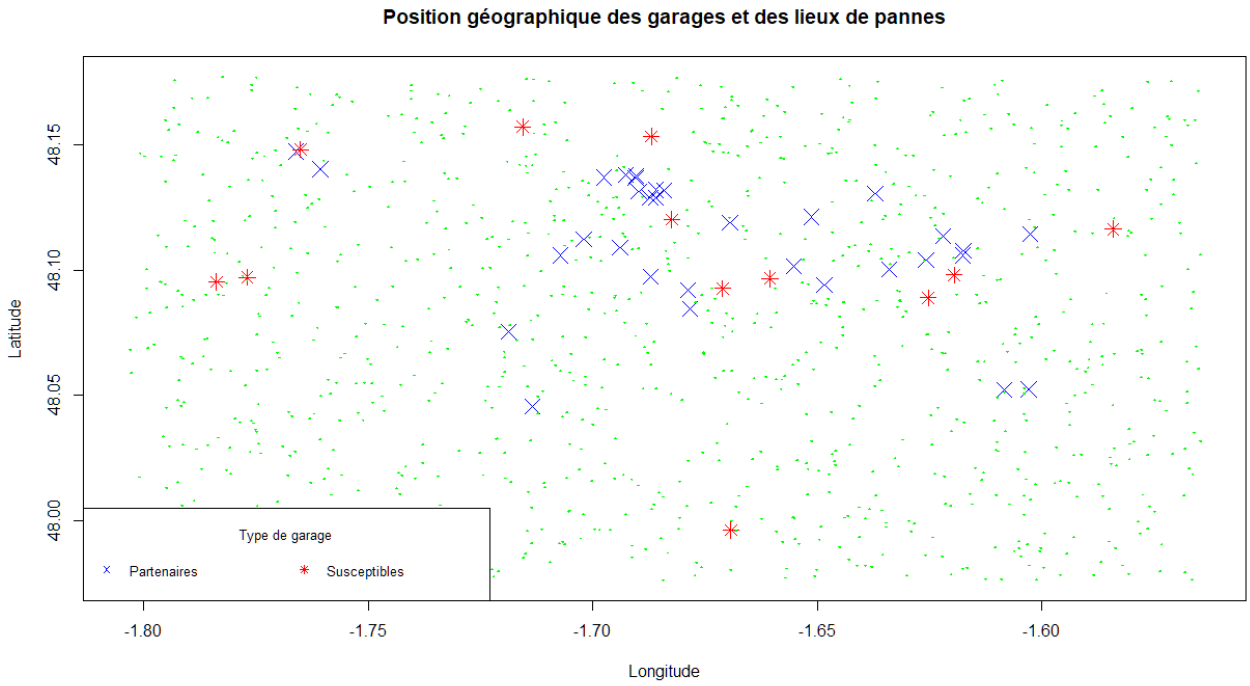


FIGURE 2.1 – Visualisation des données utilisées

2.2 Score des garages

2.2.1 Fonction de score

Pour attribuer un score à chaque garage, nous avons voulu que celui-ci dépende du nombre de pannes dans le secteur du garage, mais également de la distance garage - lieu de panne. Pour cela, nous avons utilisé une fonction de score qui attribue pour chaque panne, un score entre 0 et 1 selon la distance garage - lieux de panne. Cette fonction est bien évidemment décroissante ; un score proche de 1 sera attribué à une panne proche du garage, et un score proche de 0 sera attribué à une panne éloignée du garage. Ceci est cohérent avec le coût du dépannage qui dépend en partie de la distance de trajet. Cependant, la partie la plus délicate est de choisir la forme de cette fonction. Après réflexion, et selon les cas d'applications, nous avons retenu deux fonctions. La première fonction (fig. 2.2) nous a été utile lors des deux premières méthodes de scoring, alors que la deuxième fonction (fig. 2.4) nous a été utile lors de la dernière méthode de scoring. Nous reviendrons plus en détail sur ces fonctions lors de leurs utilisations dans les différentes méthodes de scoring.

2.2.2 1re méthode de scoring

Dans un premier temps, nous avons voulu regarder si chaque garage se trouvait dans une zone à risque (zone où un nombre important de pannes sont survenues par le passé), de manière indépendante des autres garages, c'est-à-dire sans prendre en considération la concurrence entre les garages. Ceci permet de voir quels garages sont amenés à effectuer un nombre suffisant d'interventions, de manière à ce que l'on respecte notre engagement en tant que compagnie d'assurance, c'est-à-dire de garantir un minimum d'interventions au garage.

Nous avons commencé par fixer le rayon d'action des garages (disque centré sur la localisation du garage et de rayon R). Pour scorer chaque garage, nous avons sélectionné les lieux de pannes qui se trouvent dans le rayon d'action du garage considéré. Nous avons ensuite compté le nombre pannes dans ce rayon d'action, ce qui constitue un premier indicateur, qui nous renseigne sur le risque de la zone d'action du garage. Ensuite, nous avons voulu prendre en considération la distance entre le garage et les lieux de panne, c'est pourquoi nous avons décidé d'attribuer un score à chaque panne, selon sa proximité avec le garage. Pour attribuer ce score, nous avons utilisé la fonction suivante (fig. 2.2) que l'on a créé de la manière suivante : nous nous sommes inspirés de la fonction sigmoïde, de laquelle nous avons pris le symétrique par rapport à l'axe $y = 0.5$, puis translaté et contracté entre 0 et R (le rayon d'action). Cette fonction permet d'accentuer, par rapport à une courbe linéaire, la satisfaction de la compagnie d'assurance (faible coût) et du client (temps d'attente faible) pour des pannes proches du garage, et à l'inverse, de pénaliser les pannes éloignées.

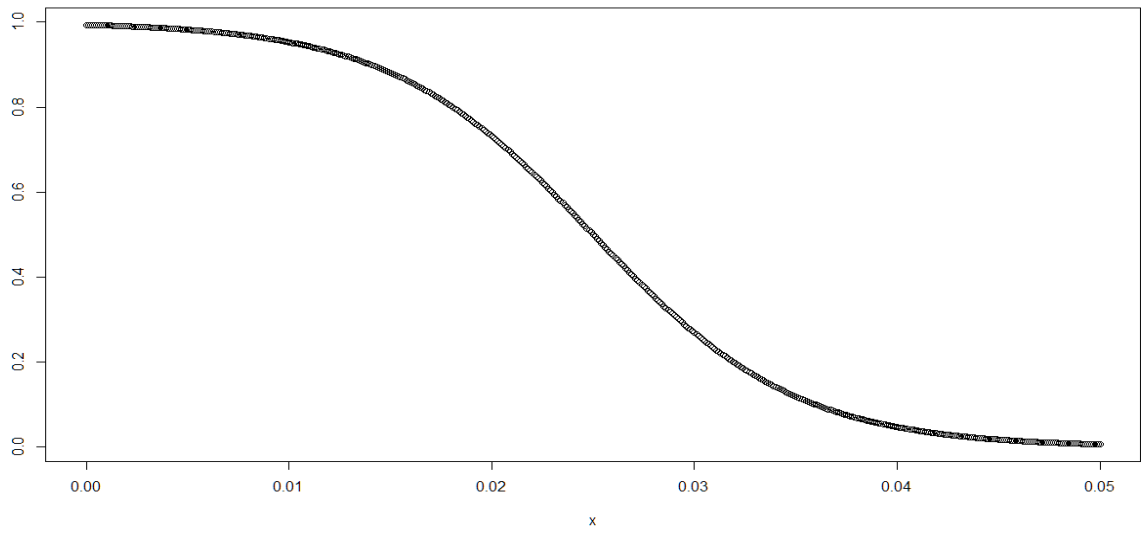


FIGURE 2.2 – 1re fonction de score utilisée

2.2.3 2e méthode de scoring

La première méthode de scoring est facile à mettre en oeuvre, mais elle présente des limites. En effet, si une panne est dans le rayon d'action de plusieurs garages, elle sera comptabilisée pour chacun de ces garages, alors qu'en réalité, lorsque la panne survient, un seul de ces garages sera missionné pour effectuer le dépannage. Généralement, c'est le garage le plus proche du lieu de panne qui sera missionné. C'est pourquoi, dans cette méthode, nous allons regarder, pour chaque panne, quel garage sera en charge du dossier, c'est-à-dire quel garage est le plus proche du lieu de panne (en terme de distance à vol d'oiseau : distance euclidienne) Pour cela, on reprend le travail initial de la 1re méthode de scoring, que l'on ajuste et améliore. On va, contrairement à la 1re méthode, commencer par scorer les garages partenaires, puis on s'occupera des garages susceptibles de devenir partenaires.

On regarde donc, pour chaque garage partenaire, les pannes localisées dans son rayon d'action R. Puis, pour chacune de ces pannes, on va regarder si elle se trouve dans le rayon d'action de plusieurs garages. Si ce n'est pas le cas, alors le dossier de cette panne est attribuée au garage considéré. Mais si c'est le cas, et qu'il y a plusieurs garages qui peuvent effectuer ce dépannage, c'est le garage le plus proche qui aura le dossier. On va donc maintenant regarder si le garage en question est le plus proche. Si c'est le cas, on lui attribue cette panne, et si ce n'est pas le cas, cette panne sera donc attribuée à un des autres garages du secteur. Et par le même procédé que précédemment, nous scorons ces garages, mais cette fois-ci en se basant uniquement sur les pannes qui seraient réellement attribuées à chaque garage.

Pour scorer les garages susceptibles de devenir partenaires, nous allons, pour chacun de ces garages, les rajouter un par un au réseau, et voir l'impact qu'aurait son recrutement dans le réseau. Ainsi, on part de notre réseau de garages initial, auquel on rajoute un garage susceptible de devenir partenaire, et on le score de la même manière que l'on a scoré les garages partenaires avec la 2e méthode : on regarde les pannes qu'il pourrait récupérer dans son secteur s'il devenait partenaire. On ajoute uniquement un garage susceptible de devenir partenaire au réseau à la fois, afin de le scorer, puis on l'enlève, on repart du réseau initial, et on recommence avec le garage suivant.

2.2.4 3e méthode de scoring

La deuxième méthode de scoring est beaucoup plus réaliste que la première, car elle va évaluer réellement ce qui se passerait, en regardant précisément à quel garage serait affecté un dossier de panne automobile. Cependant, elle présente elle aussi des limites : regarder pour chaque garage les pannes qui lui sont proches, pour ensuite déterminer, pour chacune de ces pannes, si elles sont plus proche de ce garage ou d'un autre est très coûteux en temps de calcul. Ceci est problématique pour traiter le même problème avec un réseau plus grand

et un historique de panne plus grand.

C'est pourquoi nous avons développé une 3e méthode, qui est à la fois plus rapide en temps de calcul, mais également plus réaliste. Grâce au package osrm, nous avons été capable de créer une matrice des temps de trajet entre les garages et les lieux de pannes. Ainsi, l'élément (i,j) de cette matrice contient le temps de trajet réel (en minutes) entre le lieu de panne i et le garage j. Le package osrm assure le lien entre R et l'API d'OSRM. OSRM est un service de routage basé sur les données OpenStreetMap. Ce package permet d'obtenir le temps de trajet, la distance et le plus court chemin entre deux points via le réseau routier d'OpenStreetMap. Ce package et ces fonctions présentent quand même des inconvénients, il y a un problème de stabilité : sans modifications, le même code R peut ne pas fonctionner lors de sa première exécution, et très bien fonctionner lors d'une deuxième exécution. Ceci est problématique pour le calcul de cette matrice des distances, mais une fois que l'on a réussi à l'obtenir, le problème est résolu car nous n'aurons plus besoin de calculer des distances par la suite. Nous n'aurons plus qu'à exploiter les données de cette matrice.

Désormais, en utilisant la fonction which.min par lignes sur cette matrice (où l'on regarde uniquement les colonnes des garages partenaires), on obtient, pour chaque lieu de panne, le garage partenaire le plus proche en temps de trajet. Et en appliquant la fonction min, on obtient les temps de trajet entre chaque lieu de panne et le garage chargé du dossier de la panne en question (fig. 2.3). Par la suite, nous allons scorer les garages en fonction des pannes qui leurs sont attribuées, et de leurs distances au garage. Pour cela, il est nécessaire d'avoir une fonction de score adaptée à ce cas, c'est pourquoi nous avons tout d'abord voulu exploiter l'histogramme des valeurs des temps de trajet "Garage - Lieu de panne" (fig. 2.3).

À partir de cet histogramme, nous avons pu créer une fonction de score adaptée (fig. 2.4). Étant donné que la majeure partie des temps de trajet est comprise entre 0 et 15 minutes, nous avons voulu créer une fonction dont l'allure avait une forte décroissance entre 0 et 15 minutes, afin de différencier au mieux le score des différentes pannes. En fonction des valeurs observées des distances garages - lieux de pannes, nous avons décidé d'utiliser la fonction $e^{-\frac{x}{10}}$ où x représente la distance garage - lieu de panne. La première fonction de score utilisée n'est pas adaptée à ce cas, car nous aurions obtenu des scores très proches pour des pannes étant à un temps de trajet proche de 0 minute, et des pannes étant à un temps de trajet proche de 10 minutes, ce qui ne nous intéresse pas car nous cherchons à différencier les garages afin qu'ils soient au plus proche des lieux de panne.

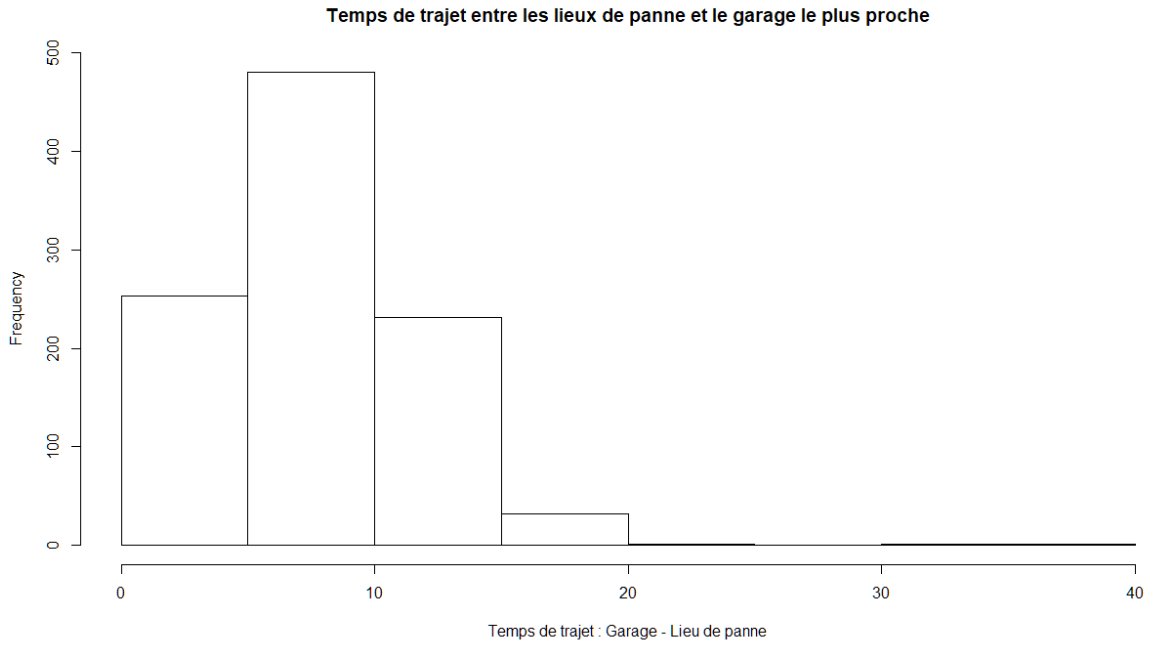


FIGURE 2.3 – Histogramme des temps de trajet "Garage - Lieu de panne"

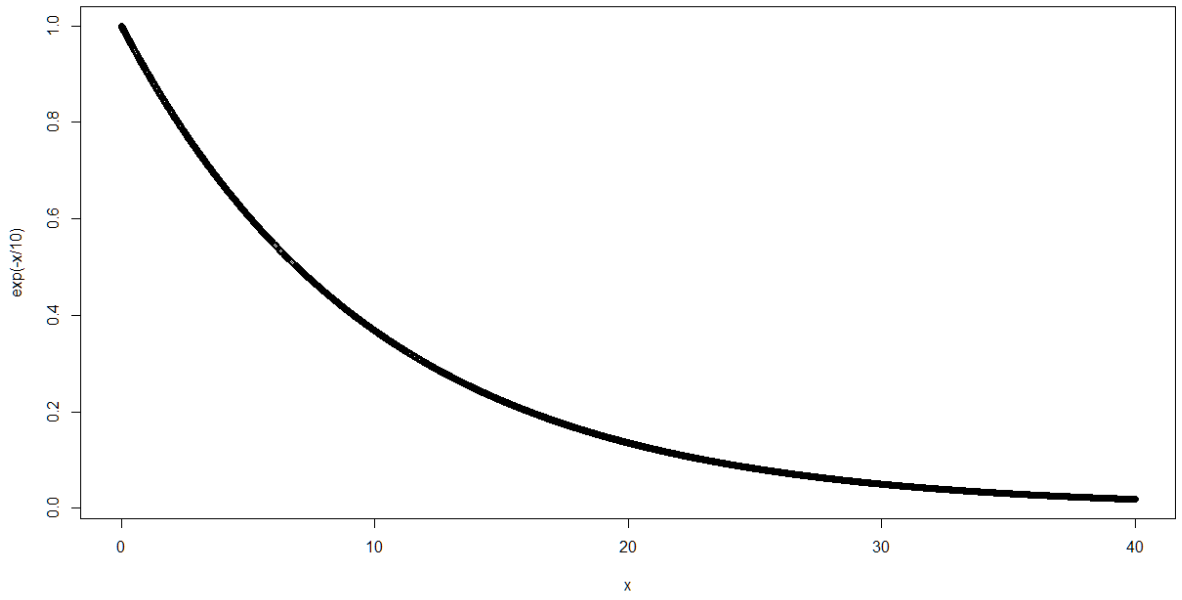


FIGURE 2.4 – 2e fonction de score utilisée

Ayant désormais la matrice des temps de trajet et la fonction de score, nous avons procédé de la même manière que pour la 2e méthode de scoring, c'est-à-dire que l'on a dans un premier temps scoré les garages partenaires, puis nous avons scoré les garages susceptibles de devenir partenaires en les ajoutant un à un au réseau. Une différence existe néanmoins, nous avons abandonné la contrainte du rayon d'action R de chaque garage, car on considère que chaque panne doit être prise en compte, peu importe où elle se trouve, car nous n'allons pas laisser un assuré en panne en bord de route sans lui porter assistance.

Cette approche présente un réel intérêt car l'utilisation des temps de trajet en minutes est beaucoup plus proche de la réalité, et coïncide avec le temps d'attente de l'assuré avant d'être dépanné.

2.2.5 Méthode d'optimisation

Une fois que les garages sont scorés, il nous reste à exploiter ces scores, c'est-à-dire à faire des choix, prendre des décisions quant à retirer des garages partenaires du réseau et à recruter des garages susceptibles de devenir partenaires.

Nous avons tout d'abord voulu réutiliser nos travaux sur le problème p-médian évoqué précédemment et l'adapter à cette situation. Cependant, dans l'application du problème p-médian, le nombre entier « p » est fixé dès le départ et cela ne nous convenait pas car nous voulions plus de flexibilité. En effet, nous considérons que le nombre final de garages dans le réseau doit plutôt être déterminé par le besoin réel en garages, et doit davantage dépendre de la sinistralité, des scores des garages et de nos critères qui déterminent si l'on retire ou recrute des garages. C'est pourquoi nous avons décidé de ne pas travailler avec le problème p-médian.

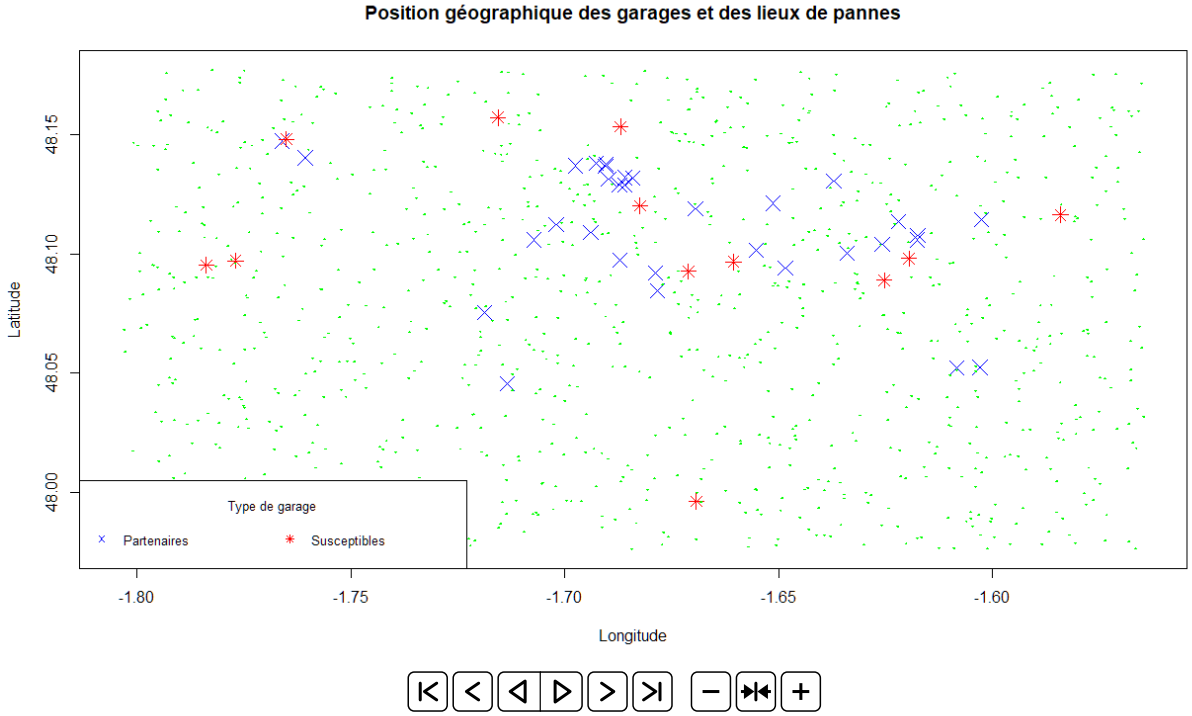
Nous avons donc utilisé une boucle while (tant que), qui s'arrête lorsqu'il n'y a plus d'évolution entre les états des garages, à savoir s'ils sont "Partenaires" ou "Susceptibles". À chaque tour de boucle, nous calculons les scores des garages, peu importe leurs états. On fixe une limite qui correspond à un nombre minimal d'interventions que chaque garage doit faire pour rester partenaire (car la compagnie d'assurance doit garantir ce minimum à ses garages). Les garages "Partenaires" qui ont un score inférieur à la limite passent à l'état "Susceptibles", c'est-à-dire qu'ils sont retirés du réseau à cette étape, mais qu'ils ne sont pas définitivement supprimés. En effet, après plusieurs tours de boucle, et les changements qui en découlent, ils seront peut-être amenés à être réintégrés dans le réseau. Ensuite, on ajoute un des garages "Susceptibles" au réseau si son score est supérieur à la limite fixée (toujours pour satisfaire la garantie d'un nombre minimal d'interventions promises par la compagnie d'assurance). Il est important de remarquer que l'on en ajoute qu'un seul à la fois, car les scores des garages "Susceptibles" ont été calculés en simulant l'ajout d'un unique garage au réseau. Nous ne connaissons pas l'influence qu'aurait l'ajout de plusieurs garages à la fois. Bien évidemment, l'unique garage "Susceptible" que l'on ajoute (s'il

existe, car son score doit être supérieur à la limite fixée) est celui qui possède le score le plus élevé.

Par ce procédé, à chaque tour de boucle, nous retirons du réseau tous les garages "Partenaires" qui ont un score insuffisant, et on ajoute le garage "Susceptible" ayant le meilleur score, si celui-ci est supérieur à la limite que l'on s'est fixée.

L'animation suivante vous permet de voir l'évolution de l'état des différents garages entre "Partenaires" et "Susceptibles", à chaque itération de la méthode d'optimisation.

Animation des étapes de la méthode d'optimisation



Remarque : Il est fort probable que le lecteur PDF par défaut de votre navigateur ne gère pas les boutons d'action et animations (Adobe Acrobat Reader DC le fait)

2.2.6 Limites

Nous pourrions rajouter des hypothèses et des contraintes à notre modèle pour le complexifier. Nous pourrions considérer que, sur la durée pour laquelle nous possédons l'historique des pannes utilisé, un garage ne puisse avoir qu'un nombre limité de dossiers pour des questions de gérabilités (par exemple : 50 pannes par mois). Nous pourrions également optimiser d'une manière différente, en cherchant à maximiser le pourcentage de pannes dépannées par un garage se trouvant à une distance ou à un temps de trajet inférieur à une borne que l'on fixerait.

Conclusion

Lors de ce projet, nous avons pu nous rendre compte de la force du problème p-médian sur un sujet d'optimisation de positionnement territorial sous contraintes. La limite principale, qui est la complexité du problème, oblige à s'appuyer sur des méthodes annexes afin de trouver une solution optimale acceptable. L'utilisation du problème 1-médian nous a permis, dans un premier temps, de trouver des solutions restreintes à des départements et à la localisation d'une seule activité. Par la suite, afin de résoudre le problème pour plusieurs activités, nous avons décidé de nous appuyer sur une méthode annexe : le partitionnement de données et l'algorithme du K-means. Cependant, cette méthode présente quelques limites et peut encore être améliorée. Principalement, les zones créées par l'algorithme du K-means peuvent être hétérogènes et l'utilisation du problème 1-médian sur chaque zone se fait de manière indépendante des autres zones.

Ensuite, nous nous sommes penchés sur un sujet plus proche de la réalité : l'optimisation d'un réseau de garages partenaires. La difficulté principale fut la méthode de scoring. Á l'instar du problème 1-médian, nous avons amélioré à plusieurs reprises notre méthode de scoring jusqu'à utiliser une fonction score s'appuyant sur les temps de trajet réels. Par la suite, tout notre travail s'est concentré sur la méthode d'optimisation, à savoir comment procédé pour obtenir le réseau de garage le plus optimal possible. Celui-ci aurait encore pu être d'avantage complexifié en limitant le nombre de dossiers gérés par un garage par exemple.

Finalement, ce projet, composé de deux sujets quelques peu différents, montre la diversification et la complexité de l'optimisation territoriale, aussi bien utile pour un compagnie d'assurance voulant offrir un prise en charge rapide à leurs assurés que pour une entreprise voulant s'implanter de façon optimale au sein d'un territoire.

Annexe A

Code informatique R - Chapitre 1

```
### Programme 1-médian ###



unmedian1 = function(z){

  colpoids = which(colnames(z)=='poids')
  colx = which(colnames(z)=='x')
  coly = which(colnames(z)=='y')
  a = sum(z$poids)
  i1 = 0; i2 = 0; i3 = 0; i4 = 0
  i = max(i1,i2,i3,i4)

  while (i < a/2){

    linexmin = which.min(z$x)
    pxmin = z[linexmin, colpoids]
    if (pxmin < a/2){
      z = z[-linexmin,]
      linexmin2 = which.min(z$x)
      z[linexmin2, colpoids] = z[linexmin2, colpoids] + pxmin
      i1 = z[linexmin2, colpoids]
      coord1 = c(z[linexmin2, colx], z[linexmin2, coly])}
```

```
  lineymin = which.min(z$y)
  pymin = z[lineymin, colpoids]
  if (pymin < a/2 & i1 < a/2){
    z = z[-lineymin,]
    lineymin2 = which.min(z$y)
    z[lineymin2, colpoids] = z[lineymin2, colpoids] + pymin
    i2 = z[lineymin2, colpoids]
    coord2 = c(z[lineymin2, colx],z[lineymin2, coly])}

  linexmax = which.max(z$x)
  pxmax = z[linexmax, colpoids]
  if (pxmax < a/2 & i1 < a/2 & i2 < a/2){
    z = z[-linexmax,]
    linexmax2 = which.max(z$x)
    z[linexmax2, colpoids] = z[linexmax2, colpoids] + pxmax
    i3 = z[linexmax2, colpoids]
    coord3 = c(z[linexmax2, colx], z[linexmax2, coly])}

  lineymax = which.max(z$y)
  pymax = z[lineymax, colpoids]
  if (pymax < a/2 & i1 < a/2 & i2 < a/2 & i3 < a/2){
    z = z[-lineymax,]
    lineymax2 = which.max(z$y)
    z[lineymax2, colpoids] = z[lineymax2, colpoids] + pymax
    i4 = z[lineymax2, colpoids]
    coord4 = c(z[lineymax2, colx], z[lineymax2, coly])}

  i = max(i1,i2,i3,i4)}

if (i1 >= a/2){
  return(coord1)}
if (i2 >= a/2){
  return(coord2)}
if (i3 >= a/2){
  return(coord3)}
if (i4 >= a/2){
  return(coord4)}}
```

```
unmedian2 = function(z){

  colpoids = which(colnames(z)=='poids')
  colx = which(colnames(z)=='x')
  coly = which(colnames(z)=='y')
  a = sum(z$poids)
  i = 0
  k = 0

  while (i<a/2){

    linexmin = which.min(z$x)
    pxmin = z[linexmin,colpoids]
    lineymin = which.min(z$y)
    pymin = z[lineymin,colpoids]
    linexmax = which.max(z$x)
    pxmax = z[linexmax,colpoids]
    lineymax = which.max(z$y)
    pymax = z[lineymax,colpoids]

    extrem = unique(matrix(c(linexmin, pxmin, lineymin, pymin, linexmax,
    pxmax, lineymax, pymax), ncol = 2, byrow = T))

    poidsmin = min(extrem[,2])
    linemin = extrem[extrem[,2] == poidsmin ,1]

    if (poidsmin < a/2){

      if (poidsmin==pxmin){
        z=z[-linemin,]
        linemin2=which.min(z$x)
        z[linemin2,colpoids]=z[linemin2,colpoids]+pxmin
        i=z[linemin2,colpoids]
        coord=c(z[linemin2,colx],z[linemin2,coly])
      }
```

```
    else {
      if (poidsmin==pymin){
        z=z[-linemin,]
        linemin2=which.min(z$y)
        z[linemin2,colpoids]=z[linemin2,colpoids]+pymin
        i=z[linemin2,colpoids]
        coord=c(z[linemin2,colx],z[linemin2,coly])
      }

      else {
        if (poidsmin==pxmax){
          z=z[-linemin,]
          linemin2=which.max(z$x)
          z[linemin2,colpoids]=z[linemin2,colpoids]+pxmax
          i=z[linemin2,colpoids]
          coord=c(z[linemin2,colx],z[linemin2,coly])
        }

        else {
          if (poidsmin==pymax){
            z=z[-linemin,]
            linemin2=which.max(z$y)
            z[linemin2,colpoids]=z[linemin2,colpoids]+pymax
            i=z[linemin2,colpoids]
            coord=c(z[linemin2,colx],z[linemin2,coly])
          }
        }
      }
    }
  }

  i=max(pxmin,pymin,pxmax,pymax,i)
  k = k+1
}

return (coord)}
```

```
unmedian3 = function(z){

  colpoids = which(colnames(z)=='poids')
  colx = which(colnames(z)=='x')
  coly = which(colnames(z)=='y')
  a = sum(z$poids)
  poidsmin = 0
  k = 0

  while (dim(z)[1] > 1){

    linexmin = which.min(z$x)
    pxmin = z[linexmin,colpoids]
    lineymin = which.min(z$y)
    pymin = z[lineymin,colpoids]
    linexmax = which.max(z$x)
    pxmax = z[linexmax,colpoids]
    lineymax = which.max(z$y)
    pymax = z[lineymax,colpoids]

    extrem = unique(matrix(c(linexmin, pxmin, lineymin, pymin, linexmax,
    pxmax, lineymax, pymax), ncol = 2, byrow = T))

    poidsmin = min(extrem[,2])
    linemin = extrem[extrem[,2] == poidsmin ,1]

    d = as.matrix(dist(z[,c(colx, coly)]))
    dlinemin = d[,linemin]
    dlinemin[linemin] = max(dlinemin) + 1

    linemin2 = which.min(dlinemin)
    z[linemin2, colpoids] = z[linemin2, colpoids] + poidsmin
    coord = c(z[linemin2, colx],z[linemin2, coly])
    z = z[-linemin,]}

  return(coord)}
```

```
distance = function(z, ind){
  colx = which(colnames(z)=='x')
  coly = which(colnames(z)=='y')
  d = sqrt( ( z[ind,colx] - z[,colx] )^2 + ( z[ind, coly] - z[,coly] )^2 )
  d[ind] = max(d) + 1
  return(which.min(d))}

unmedian4 = function(z){
  h = z
  colpoids = which(colnames(z)=='poids')
  colx = which(colnames(z)=='x')
  coly = which(colnames(z)=='y')
  a = sum(z$poids)
  poidsmin = 0
  k = 0

  while (dim(z)[1] > 1){

    linexmin = which.min(z$x)
    pxmin = z[linexmin,colpoids]
    lineymin = which.min(z$y)
    pymin = z[lineymin,colpoids]
    linexmax = which.max(z$x)
    pxmax = z[linexmax,colpoids]
    lineymax = which.max(z$y)
    pymax = z[lineymax,colpoids]

    extrem = unique(matrix(c(linexmin, pxmin, lineymin, pymin, linexmax,
    pxmax, lineymax, pymax), ncol = 2, byrow = T))

    poidsmin = min(extrem[,2])
    linemin = extrem[extrem[,2] == poidsmin ,1]

    linemin2 = distance(z, linemin)
    z[linemin2, colpoids] = z[linemin2, colpoids] + poidsmin
    coord = c(z[linemin2, colx],z[linemin2, coly])
    z = z[-linemin,]    }

  return(coord)}
```

```
# Simulation sur données fictives

n = 5000
x = runif(n, 0, 10)
y = runif(n, 0, 10)
poids = runif(n, 0, 5)
z = data.frame(x, y, poids)

plot(z[,1], z[,2], cex = z[,3], type = "p",
main = "Position géographique de notre public cible",
xlab = "Longitude", ylab = 'Latitude',
col = "black", xlim = c(0, 10), ylim = c(0, 10))

unmedian4(z)



# Application à des données réelles : la France



# Importation et traitement des données

vf = read.csv("Données/villes_france.csv", header = FALSE,
sep = " ", as.is = TRUE)
vf = vf[-c(36660, 36661),]
vf[,4] = as.numeric(vf[,4])
colnames(vf) = c("Numéro ville", "Département", "Ville",
"Population", "Longitude", "Latitude")
View(vf)

colnames(vf) = c("Numéro ville", "Département", "Ville",
"poids", "x", "y")

plot(vf[,5], vf[,6], xlim = c(-6, 11), ylim = c(41, 52),
xlab = "Longitude", ylab = "Latitude")
```

```
# Restriction au département du Rhone

Rhone = vf[vf$Département == 69,]
plot(Rhone[ ,5], Rhone[ ,6], cex = 10 * (Rhone[,4] / max(Rhone[ ,4]) ),
xlab = "Longitude", ylab = "Latitude", pch = 19, col = 'blue',
main = "Commune du Rhône")
text(Rhone[Rhone$poids == max(Rhone$poids),5],
Rhone[Rhone$poids == max(Rhone$poids) ,6],
labels = Rhone[Rhone$poids == max(Rhone$poids) ,3])

# Résolution 1-médian dans le Rhone

loc4 = unmedian4(Rhone)
Rhone[( Rhone[,5] == loc4[1] & Rhone[,6] == loc4[2] ),]
points(loc4[1], loc4[2], col = 'red', pch = 18, cex = 3)
text(loc4[1], loc4[2],
labels = Rhone[( Rhone[,5] == loc4[1] & Rhone[,6] == loc4[2] ),3] )




# Restriction à la région Ile-de-France

IDF = vf[vf$Département %in% c(75, 77, 78, 91, 92, 93, 94, 95),]
plot(IDF[ ,5], IDF[ ,6], cex = 10 * (IDF[,4] / max(IDF[ ,4]) ),
xlab = "Longitude", ylab = "Latitude", pch = 19, col = 'red',
main = "Commune d'IDF")
text(IDF[IDF$poids == max(IDF$poids),5], IDF[IDF$poids == max(IDF$poids) ,6],
labels = IDF[IDF$poids == max(IDF$poids) ,3])

# Résolution 1-médian en IDF

loc = unmedian4(IDF)
IDF[( IDF[,5] == loc[1] & IDF[,6] == loc[2] ),]
points(loc[1], loc[2], col = 'blue', pch = 18, cex = 3)
text(loc[1], loc[2],
labels = IDF[( IDF[,5] == loc[1] & IDF[,6] == loc[2] ),3] )
```

```
# Travail sur la France

k = 50

France$Classe = as.numeric(kmeans(France[,c(5,6)], k)$cluster)
plot(France[,5], France[,6], cex = France[,4] / max(France[,4]),
xlab = "Longitude", ylab = "Latitude", col = France[,7])

for(i in 1:k){
  unmedian4(France[France$Classe == i,])}
```

Annexe B

Code informatique R - Chapitre 2

```
###### Importation des données ######

library(readr)
garages <- read_delim("Données/garages-partenaires-du-reseau-star.csv",
";", escape_double = FALSE, trim_ws = TRUE)
View(garages)

garages = garages[-which(is.na(garages$Coordonnees)),]
n = dim(garages)[1]
garages$Identifiant = 1:n


###### Traitement des données : Exploitation des coordonnées GPS ######

library(stringr)

garages$Latitude = as.numeric(str_split_fixed(garages$Coordonnees, ", ", 2)[,1])
garages$Longitude = as.numeric(str_split_fixed(garages$Coordonnees, ", ", 2)[,2])

xlim = c( min(as.numeric(garages$Longitude)) - 0.02 ,
max(as.numeric(garages$Longitude)) + 0.02 )
ylim = c( min(as.numeric(garages$Latitude)) - 0.02 ,
max(as.numeric(garages$Latitude)) + 0.02 )

plot(garages$Longitude, garages$Latitude, col = 'blue',
xlim = xlim, ylim = ylim)    # Position des garages
```

```
###### Découpage de la base de données en deux parties :
garages partenaires et garages susceptibles de devenir partenaires ######

gge_partenaires = sort(sample(garages$Identifiant, 32))
gge_susceptibles = sort(setdiff(garages$Identifiant, gge_partenaires))

Etat = NULL
Etat[gge_partenaires] = "Partenaires"
Etat[gge_susceptibles] = "Susceptibles"
garages$Etat = Etat

plot(garages$Longitude[garages$Etat == "Partenaires"],
garages$Latitude[garages$Etat == "Partenaires"],
col = 'blue', xlim = xlim, ylim = ylim, pch = 4, cex = 2,
xlab = "Longitude", ylab = "Latitude")
points(garages$Longitude[garages$Etat == "Susceptibles"],
garages$Latitude[garages$Etat == "Susceptibles"], col = 'red',
xlim = xlim, ylim = ylim, pch = 8, cex = 1.5)




###### Simuler des lieux de pannes automobiles ######

N = 1000

Lieux_pannes = cbind( runif(N, xlim[1], xlim[2]) , runif(N, ylim[1], ylim[2]) )

points(Lieux_pannes[,1], Lieux_pannes[,2], col = 'green', pch = 18, cex = 0.4)
```

```
# Fonctions annexes #

# Vérifie si les points P sont dans le cercle de centre C et de rayon R

test_disque = function(P, C, R){
  a = ((P[,1] - C[1])^2 + (P[,2] - C[2])^2) < R^2
  return(a)}

# Calcul les distances des points P au point C

distance = function(P, C){
  d = sqrt( (C[1] - P[,1])^2 + (C[2] - P[,2])^2 )
  return(d)}

# Donne un score au garage C qui gère les pannes aux lieux P
selon la distance garage- lieu de panne

score = function(P, C){
  s = sum( 1/ ( 1 + exp( - ((abs( R - distance(P, C) )*(1/R))*10 -5) ) ) )
  return(s)}



###### Score des garages ######


R = 0.05
Nb_pannes = NULL
Score = NULL

for(k in 1:n){
  Loc_gge = c(garages$Longitude[k], garages$Latitude[k])
  Pannes_proches = Lieux_pannes[test_disque(Lieux_pannes, Loc_gge, R) ,]
  Nb_pannes[k] = dim(Pannes_proches)[1]
  Score[k] = score(Pannes_proches, Loc_gge) }

garages$Nb_pannes_ind = Nb_pannes
garages$Score_ind = Score
```

```
R = 0.05
Nb_pannes = rep(NA, dim(garages)[1])
Score = rep(NA, dim(garages)[1])
adresse_gge_part = garages[garages$Etat == 'Partenaires', c(9,8)]

for(k in gge_partenaires){

  Loc_gge = c(garages$Longitude[k], garages$Latitude[k])
  Pannes_proches = Lieux_pannes[test_disque(Lieux_pannes, Loc_gge, R) ,]
  Lieux_pannes_reelles = data.frame()
  m = 0

  for(i in 1:(dim(Pannes_proches)[1])){

    ind = which(test_disque(adresse_gge_part, Pannes_proches[i,], R))

    if(length(ind) == 1){
      gge_gagnant = Loc_gge

    }else{

      d = cbind(ind, distance(adresse_gge_part[ind,], Pannes_proches[i,]))
      ind_gge_gagnant = d[which.min(d[,2]),1]
      gge_gagnant = adresse_gge_part[ind_gge_gagnant,] }

    if( (Loc_gge[1] == gge_gagnant[1]) &  (Loc_gge[2] == gge_gagnant[2])){
      m = m + 1
      Lieux_pannes_reelles[m,1] = Pannes_proches[i,1]
      Lieux_pannes_reelles[m,2] = Pannes_proches[i,2]   } }

  if(dim(Lieux_pannes_reelles)[1] > 0){
    Nb_pannes[k] = dim(Lieux_pannes_reelles)[1]
    Score[k] = score(Lieux_pannes_reelles, Loc_gge)
  }else{
    Nb_pannes[k] = 0
    Score[k] = 0 }}

garages$Nb_pannes_dep = Nb_pannes
garages$Score_dep = Score
```

```
R = 0.05
Nb_pannes = rep(NA, dim(garages)[1])
Score = rep(NA, dim(garages)[1])

for(k in gge_susceptibles){

  Loc_gge = c(garages$Longitude[k], garages$Latitude[k])
  Pannes_proches = Lieux_pannes[test_disque(Lieux_pannes, Loc_gge, R) ,]
  adresse_gge_part = garages[garages$Etat == 'Partenaires', c(8,9)]
  adresse_gge_part[dim(adresse_gge_part)[1]+1,] = Loc_gge
  Lieux_pannes_reelles = data.frame()
  m = 0

  for(i in 1:(dim(Pannes_proches)[1])){

    ind = which(test_disque(adresse_gge_part, Pannes_proches[i,], R))

    if(length(ind) == 1){
      gge_gagnant = Loc_gge

    }else{
      d = cbind(ind, distance(adresse_gge_part[ind,], Pannes_proches[i,]))
      ind_gge_gagnant = d[which.min(d[,2]),1]
      gge_gagnant = adresse_gge_part[ind_gge_gagnant,] }

    if( (Loc_gge[1] == gge_gagnant[1]) &  (Loc_gge[2] == gge_gagnant[2])  ){
      m = m + 1
      Lieux_pannes_reelles[m,1] = Pannes_proches[i,1]
      Lieux_pannes_reelles[m,2] = Pannes_proches[i,2]      } }

  if(dim(Lieux_pannes_reelles)[1] > 0){
    Nb_pannes[k] = dim(Lieux_pannes_reelles)[1]
    Score[k] = score(Lieux_pannes_reelles, Loc_gge)
  }else{
    Nb_pannes[k] = 0
    Score[k] = 0 }}

garages$Nb_pannes_dep[gge_susceptibles] = Nb_pannes[gge_susceptibles]
garages$Score_dep[gge_susceptibles] = Score[gge_susceptibles]
```

```
###### Matrice des temps de trajet Garages <-> Pannes ######

# Package OSRM
# https://rgeomatic.hypotheses.org/710

library(osrm)

Localisation_pannes = as.data.frame(cbind(id = 1:N,
lon = Lieux_pannes[,1], lat = Lieux_pannes[,2]))
Source_garage = as.data.frame(cbind(id = 1:n,
lon = garages$Longitude, lat = garages$Latitude))

osrmTable(src = Source_garage[1, c("id", "lon", "lat")],
dst = Localisation_pannes[1:200, c("id", "lon", "lat")])


Distance = matrix(0, nrow = N, ncol = n)
colnames(Distance) = 1:n
for(k in 1:n){
  for(j in 1:(N/200)){
    ind_ligne = (200*(j-1)+1):(200*j)
    Distance[ind_ligne,k    ] = osrmTable(src = Source_garage[k, c("id", "lon", "lat"
    dst = Localisation_pannes[ind_ligne, c("id", "lon", "lat")])$durations }}

apply(Distance, 1, which.min)




hist(apply(Distance[,gge_partenaires], 1, min),
main = "Temps de trajet entre les lieux de panne et le garage le plus proche",
xlab = "Temps de trajet : Garage - Lieu de panne")
x = seq(0, 40, by = 0.01)
plot(x, exp(-x/10))
```

```
Score = NULL
Nb_pannes = NULL

for(k in 1:length(gge_partenaires)){
  table = Distance[,gge_partenaires]
  x = table[which(apply(table, 1, which.min) == k), k]
  Nb_pannes[gge_partenaires[k]] = length(x)
  Score[gge_partenaires[k]] = sum(exp(-x/10)) }


for(k in 1:length(gge_susceptibles)){
  table = Distance[,c(gge_susceptibles[k], gge_partenaires)]
  x = table[which(apply(table, 1, which.min) == 1), 1]
  Nb_pannes[gge_susceptibles[k]] = length(x)
  Score[gge_susceptibles[k]] = sum(exp(-x/10)) }

garages$Nb_pannes = Nb_pannes
garages$Score = Score


# Ici, la condition du rayon R est abandonné.
# On considère que chaque panne doit être prise en compte




# Choix des garages partenaires et susceptibles après évaluation des scores #

limite = 10

ind_suppr = (garages$Etat == 'Partenaires' & garages$Score < limite)
ifelse(length(which(ind_suppr)) == 0, 0,
(garages[ind_suppr,]$Etat = rep('Susceptibles', length(which(ind_suppr))) ))

id_aj = ifelse((max(garages[garages$Etat == 'Susceptibles',]$Score) > limite),
as.numeric(garages[garages$Etat == 'Susceptibles',]
[which.max(garages[garages$Etat == 'Susceptibles',]$Score), 1]), NA)
ifelse( is.na(id_aj), 0,
(garages[garages$Identifiant == id_aj,]$Etat = 'Partenaires'))
```

```
####### Méthode d'optimisation ######

plot(garages$Longitude[garages$Etat == "Partenaires"],
garages$Latitude[garages$Etat == "Partenaires"],
main = "Position géographique des garages et des lieux de pannes",
xlab = "Longitude", ylab = 'Latitude', col = 'blue',
xlim = xlim, ylim = ylim, pch = 4, cex = 2)
points(garages$Longitude[garages$Etat == "Susceptibles"],
garages$Latitude[garages$Etat == "Susceptibles"], col = 'red',
xlim = xlim, ylim = ylim, pch = 8, cex = 1.5)
points(Lieux_pannes[,1], Lieux_pannes[,2],
col = 'green', pch = 18, cex = 0.4)
legend("bottomleft", title="Type de garage",
c("Partenaires","Susceptibles"), pch = c(4, 8),
col = c("blue", "red"), bg = "white", horiz=TRUE, cex=0.8)

gge_partenaires_avant = gge_susceptibles
gge_susceptibles_avant = gge_partenaires
it = 1
limite = 10

while((sum(gge_partenaires == gge_partenaires_avant) +
sum(gge_susceptibles == gge_susceptibles_avant) != n) & (it < 50)){

  gge_partenaires_avant = gge_partenaires
  gge_susceptibles_avant = gge_susceptibles
  plot(garages$Longitude[garages$Etat == "Partenaires"],
  garages$Latitude[garages$Etat == "Partenaires"],
  main = "Position géographique des garages et des lieux de pannes",
  xlab = "Longitude", ylab = 'Latitude', col = 'blue',
  xlim = xlim, ylim = ylim, pch = 4, cex = 2)
  points(garages$Longitude[garages$Etat == "Susceptibles"],
  garages$Latitude[garages$Etat == "Susceptibles"], col = 'red',
  xlim = xlim, ylim = ylim, pch = 8, cex = 1.5)
  points(Lieux_pannes[,1], Lieux_pannes[,2],
  col = 'green', pch = 18, cex = 0.4)
  legend("bottomleft", title="Type de garage",
  c("Partenaires","Susceptibles"), pch = c(4, 8),
```

```
 col = c("blue", "red"), bg = "white", horiz=TRUE, cex=0.8)

 Score = NULL
 Nb_pannes = NULL

 for(k in 1:length(gge_partenaires)){
   table = Distance[,gge_partenaires]
   x = table[which(apply(table, 1, which.min) == k), k]
   Nb_pannes[gge_partenaires[k]] = length(x)
   Score[gge_partenaires[k]] = sum(exp(-x/10)) }

ind_suppr = (garages$Etat == 'Partenaires' & garages$Score < limite)
 ifelse(length(which(ind_suppr)) == 0, 0,
 (garages[ind_suppr,]$Etat = rep('Susceptibles', length(which(ind_suppr)))))

 gge_partenaires = which(garages$Etat == "Partenaires")
 gge_susceptibles = which(garages$Etat == "Susceptibles")

 plot(garages$Longitude[garages$Etat == "Partenaires"],
 garages$Latitude[garages$Etat == "Partenaires"],
 main = "Position géographique des garages et des lieux de pannes",
 xlab = "Longitude", ylab = 'Latitude', col = 'blue',
 xlim = xlim, ylim = ylim, pch = 4, cex = 2)
 points(garages$Longitude[garages$Etat == "Susceptibles"],
 garages$Latitude[garages$Etat == "Susceptibles"], col = 'red',
 xlim = xlim, ylim = ylim, pch = 8, cex = 1.5)
 points(Lieux_pannes[,1], Lieux_pannes[,2],
 col = 'green', pch = 18, cex = 0.4)
 legend("bottomleft", title="Type de garage",
 c("Partenaires","Susceptibles"), pch = c(4, 8),
 col = c("blue", "red"), bg = "white", horiz=TRUE, cex=0.8)


 for(k in 1:length(gge_susceptibles)){
   table = Distance[,c(gge_susceptibles[k], gge_partenaires)]
   x = table[which(apply(table, 1, which.min) == 1), 1]
   Nb_pannes[gge_susceptibles[k]] = length(x)
   Score[gge_susceptibles[k]] = sum(exp(-x/10)) }
 garages$Nb_pannes = Nb_pannes
 garages$Score = Score
```

```
id_aj = ifelse((max(garages[garages$Etat == 'Susceptibles',]$Score) > limite),
as.numeric(garages[garages$Etat == 'Susceptibles',]
[which.max(garages[garages$Etat == 'Susceptibles',]$Score), 1]), NA)
ifelse( is.na(id_aj), 0,
(garages[garages$Identifiant == id_aj,]$Etat = 'Partenaires'))

it = it + 1
gge_partenaires = which(garages$Etat == "Partenaires")
gge_susceptibles = which(garages$Etat == "Susceptibles")}
```

Bibliographie

[1] Marc S. Daskin et Kayse Lee Maass. The p-median problem. 2015.

[2] Charline Cléraux et Pierre Bougues. Relaxation lagrangienne et le problème p-médian. 2009.

[3] Jérôme Baray. Le modèle p-médian et sa résolution.

[4] Colin Fay. Machine Learning avec R. Volume 3 : k-means clustering. 2017.

[5] Andrea Trevino. Introduction to K-means Clustering. 2016.

[6] http ://scikit-learn.org/stable/modules/clustering.htmlk-means

[7] https ://rgeomatic.hypotheses.org/710

[8] https ://www.data.gouv.fr/