Service algorithme

Communication

On peut communiquer avec ce service via son API REST sur le port 4242 développée à l'aide de la librairie cpprestsdk de Microsoft. Il reçoit un document JSON en méthode POST dans le format ci-dessous :

```
{
    "carSpeed": 15,
    "walkSpeed": 5,
    "minCarDist": 1000,
    "treatmentTypes": [
        {"_id": 0, "duration": 15},
        {"_id": 1, "duration": 30}
    ],
    "nurses": [
        {"_id": 0, "treatmentTypeIds": [], "startTime": 600},
        {"_id": 1, "treatmentTypeIds": [1], "startTime": 480}
    ],
    "patients": [
        {"_id": 0, "location": {"x": -894, "y": -1752}, "treatments": [
            {"_id": 0, "treatmentTypeId": 0}
        ]},
        {"_id": 1, "location": {"x": 1058, "y": 3482}, "treatments": [
            {"_id": 1, "treatmentTypeId": 0},
            {"_id": 2, "treatmentTypeId": 1}
        ]}
    ]
}
```

Ce document contient la vitesse moyenne (en km/h) des infirmières en voiture et à pied, ainsi que la distance minimale (en m) à partir de laquelle les infirmières doivent prendre la voiture. Il contient aussi les différents types de traitement que les infirmières peuvent être amenées à soigner, la liste des infirmières avec les identifiants des types de traitement qu'elles ne peuvent pas soigner et leur heure de début de travail (en minutes), et la liste des patients avec leur localisation (coordonnées en m) et leur liste de traitements à soigner.

La réponse à cette requête est elle aussi en format JSON contenant la liste de tous les rendez-vous :

```
{
    "appointments": [
        {"nurseId": 0, "patientId": 1, "treatmentId": 1, "schedule": 800},
        {"nurseId": 1, "patientId": 1, "treatmentId": 0, "schedule": 700},
        {"nurseId": 0, "patientId": 0, "treatmentId": 2, "schedule": 1100}
    ]
}
```

Un rendez-vous contient l'identifiant de l'infirmière qui intervient, du patient et de son traitement à soigner, ainsi que son heure de début (en minutes).

Description du problème

L'algorithme doit retourner une liste de rendez-vous qui tente de minimiser au maximum la distance totale parcourue en voiture par toutes les infirmières. Aussi, ne infirmière ne doit pas avoir à soigner un traitement pour lequel elle n'est pas habilitée. Enfin, l'algorithme doit faire en sorte que chaque infirmière ait à peu près le même nombre de patients à soigner.

Une infirmière commence sa journée chez son premier patient, et finit celle-ci chez son dernier. Le point de départ de l'infirmière n'est pas prise en compte, on suppose que celle-ci se rend chez son premier patient en voiture. Lorsqu'elle va chez un patient, l'infirmière s'occupe de tous les traitements de ce dernier. Après son premier rendez-vous de la journée, l'infirmière se rend chez son prochain patient en voiture seulement si celui-ci habite à une distance supérieure à une distance minimale précisée dans le document JSON de la requête, et dans le cas contraire l'infirmière laisse sa voiture garée chez son premier patient. Notons que les distance sont calculées à l'aide de la norme 1 (aussi appelée distance Manhattan) car nous estimons que c'est une bonne estimation de la distance d'un trajet en ville. Si une infirmière doit prendre sa voiture pour se rendre chez son prochain patient, mais que celle-ci est garée plus loin, elle rejoint d'abord sa voiture à pied seulement si la distance depuis sa voiture au prochain patient est elle aussi supérieure à la distance minimale.

Obtention d'une solution

Pour obtenir ces résultats, la dernière version du service utilise un algorithme génétique. Une solution au problème est codée sous la forme d'une matrice. Chaque case de cette matrice porte un numéro compris entre 0 et le nombre total de cases de la matrice moins 1, et chaque numéro apparaît une seule et unique fois. Le nombre présent dans la matrice à la ligne i et à la colonne j correspond à l'indice du j-ième patient que doit traiter l'infirmière d'indice i dans sa journée. L'indice d'une infirmière ou d'un patient correspond à son emplacement dans la liste correspondante de l'objet *Inputs* qui stocke les données du JSON de la requête. Une matrice a donc autant de lignes qu'il y a d'infirmières et un nombre de colonnes égal à la division euclidienne du nombre de patients par le nombre d'infirmières. Ainsi, le reste de cette division équivaut au nombre de patients qui ne seront pas directement pris en compte par l'algorithme. Nous verrons plus tard comment ces patients sont réintroduits dans la solution.

La *fitness* d'une solution est un score positif permettant d'évaluer son adéquation au problème. Ici elle correspond à la distance totale parcourue en voiture par toutes les infirmières. Ainsi, de manière contre intuitive, plus la *fitness* d'une solution est petite, plus celle-ci est optimisée. De manière évidente, une solution avec un score de *fitness* nul est optimale.

En réalité, comme toutes les lignes de la matrice sont équivalentes, ce n'est pas nécessairement l'infirmière d'indice i qui va s'occuper des patients de la ligne i de la matrice. Ainsi, après que l'algorithme a abouti à une solution, chaque ligne de la matrice est attribuée à une infirmière de telle sorte que celle-ci soit capable de soigner tous les patients qu'elle contient. Il arrive qu'une telle répartition ne soit pas possible, dans ce cas certains patients sont intervertis dans la matrice pour palier à ce problème tout en essayant de conserver au maximum la fitness de la solution.

Ensuite, les patients mis de côté précédemment sont réintroduits dans le planning des infirmières de manière à toujours minimiser la *fitness* de la solution. Ces patients sont choisis de sorte à ce qu'ils soient le plus facilement possible réintroduits dans le planning sans trop changer l'optimalité de la solution d'abord trouvée par l'algorithme. La solution pour laquelle nous avons optée consiste à mettre

de côté les patients ayant le plus de patients à une distance inférieure à la distance minimale de trajet en voiture.

Enfin, il ne reste plus qu'à convertir la solution obtenue en rendez-vous en format JSON pour répondre à la requête initiale.

Fonctionnement de l'algorithme génétique

Une population de solutions générées aléatoirement est d'abord créée. Cette population doit avoir une taille divisible par 4 afin de simplifier les prochaines étapes. L'algorithme va ensuite tourner jusqu'à ce qu'il trouve une solution évidemment optimale (i.e. de *fitness* nulle), ou bien jusqu'à un nombre maximum de générations. Il renverra alors la solution qui aura obtenu la meilleure fitness jusque là. Une génération consiste à créer une nouvelle population à partir de la population précédente en trois étapes :

- Une étape de sélection qui consiste à conserver uniquement certaines solutions dans la population.
- Une étape d'enjambement qui consiste à générer de nouvelles solutions dites « enfants » à partir des solutions « parents » qui ont précédemment été sélectionnées afin de retrouver une population de même taille.
- Une étape de mutation qui consiste à modifier aléatoirement certaines solutions selon un taux de mutation afin de ne pas converger trop rapidement vers un optimum local et d'ainsi conserver de la diversité dans les solutions.

L'algorithme de sélection pour lequel nous avons opté est le plus classique ; il consiste à sélectionner la moitié de la population initiale ayant les meilleurs scores de *fitness*.

L'algorithme d'enjambement est clairement le plus complexe à mettre en place. L'idée derrière ce processus est de créer des solutions enfants qui partagent une partie du code génétique de leurs parents. Deux solutions parents génèrent deux solutions enfants. Plusieurs implémentations de cet algorithme ont été essayées, la plus concluante consiste à conserver d'une solution parent les plus longues séries de patients qui peuvent être visité sans prendre la voiture dans chaque ligne, puis de compléter les nombres manquants dans l'ordre dans lequel ils apparaissent dans l'autre parent pour créer un premier enfant, et de faire de même en changeant le rôle des deux parents pour créer le second enfant.

L'algorithme de mutation consiste à échanger un élément d'une matrice avec un élément aléatoire de cette même matrice à un taux de 0.6 %. C'est la valeur qui a empiriquement donné les meilleurs résultats.

Enfin, l'algorithme utilise une population de 100 solutions, et un nombre maximum de générations égal à 2000 pour donner des résultats assez rapides. Mais ce nombre peut aisément être poussé à 10000 afin d'obtenir des solutions un tout petit peu plus optimales.

Performances

Pour une requête composée de 40 clusters de 10 patients avec 40 infirmières où tous les patients au sein de chaque cluster sont à une distance inférieure à la distance minimale de trajet en voiture, et où deux patients dans deux différents clusters sont à une distance supérieure cette distance minimale,

l'algorithme trouve une solution optimale (de *fitness* nulle) en 220 générations. Pour 50 clusters et 50 infirmières, l'algorithme met 417 générations à trouver une solution optimale.

Pour une requête avec 20 infirmières et 200 patients situés uniformément dans une grille de côté 400m, l'algorithme met un petit peu plus de 3000 générations pour trouver une solution optimale. Néanmoins il trouve des solutions quasi-optimales au bout de 1000 générations.