Diploma Programme
Programme du diplôme
Programa del Diploma

Informatique
Étude de cas : Algorithmes génétiques

A utiliser en novembre 2021, mai 2022 et novembre 2022

Instructions destinées aux candidats

- Ce livret d'étude de cas est indispensable pour l'épreuve 3 du niveau supérieur.

Introduction

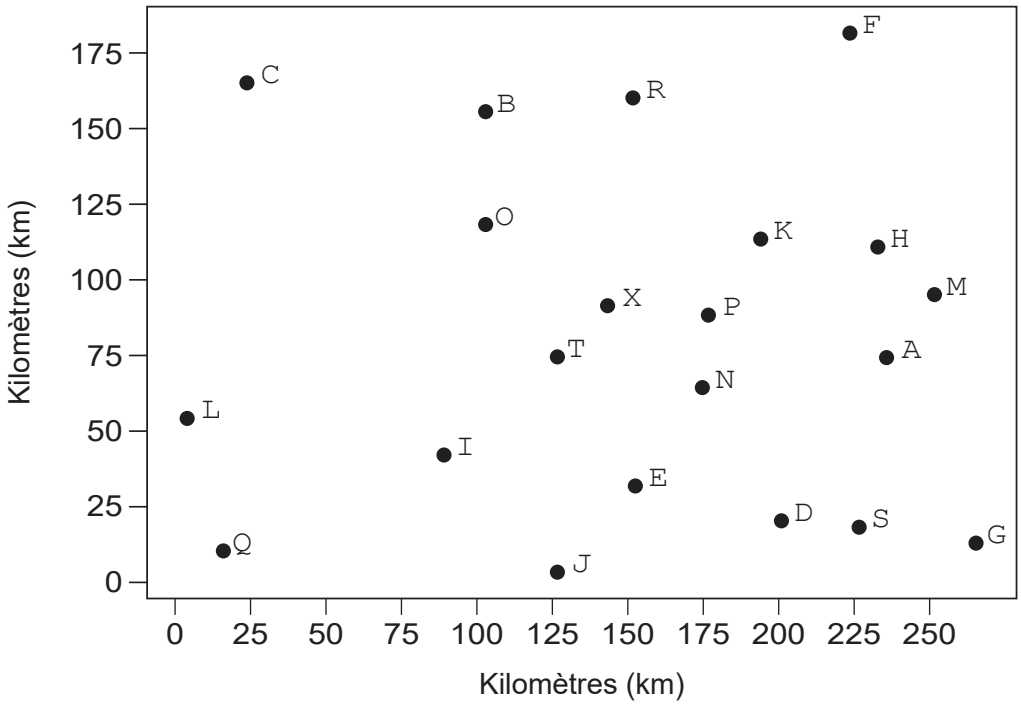
Nous sommes samedi soir et il est 19 h 00. Lotte fait sa valise pour son voyage en moto dans le Vlakland prévu pour lundi matin. Elle a repéré 20 villes qu'elle veut visiter avant de rentrer. Son amie Fenna, une élève de terminale en informatique, lui demande quel itinéraire elle pense emprunter. « Eh bien, je pensais justement que tu pourrais m'aider à le choisir, répond Lotte. J'ai trouvé une table des distances entres les villes que je voudrais visiter. » (Voir **figure 1**). « Est-ce que ça te serait possible de m'écrire un petit programme qui testerait tous les itinéraires possibles ? »

Figure 1: Distances en kilomètres (km) entre les villes que Lotte veut visiter

	X	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
X	0	94	76	141	91	60	120	145	91	74	90	55	145	108	41	49	33	151	69	111	24
A	94	0	156	231	64	93	108	68	37	150	130	57	233	26	62	140	61	229	120	57	109
B	76	156	0	80	167	133	124	216	137	114	154	100	141	161	116	37	100	169	49	185	84
C	141	231	80	0	229	185	201	286	216	139	192	178	113	239	182	92	171	155	128	251	137
D	91	64	167	229	0	49	163	65	96	114	76	93	200	91	51	139	72	185	148	26	92
E	60	93	133	185	49	0	165	115	112	65	39	91	151	117	39	99	61	139	128	75	49
F	120	108	124	201	163	165	0	173	71	194	203	74	254	90	127	136	104	269	75	163	144
G	145	68	216	286	65	115	173	0	103	179	139	123	265	83	104	194	116	250	186	39	152
H	91	37	137	216	96	112	71	103	0	160	151	39	236	25	75	130	61	239	95	93	112
I	74	150	114	139	114	65	194	179	160	0	54	127	86	171	89	77	99	80	134	140	50
J	90	130	154	192	76	39	203	139	151	54	0	129	133	155	78	117	99	111	159	101	71
K	55	57	100	178	93	91	74	123	39	127	129	0	199	61	53	91	30	206	63	101	78
L	145	233	141	113	200	151	254	265	236	86	133	199	0	251	171	118	176	46	182	226	125
M	108	26	161	239	91	117	90	83	25	171	155	61	251	0	83	151	75	251	119	81	127
N	41	62	116	182	51	39	127	104	75	89	78	53	171	83	0	90	24	168	99	69	49
O	49	140	37	92	139	99	136	194	130	77	117	91	118	151	90	0	80	139	65	159	50
P	33	61	100	171	72	61	104	116	61	99	99	30	176	75	24	80	0	179	76	86	52
Q	151	229	169	155	185	139	269	250	239	80	111	206	46	251	168	139	179	0	202	211	128
R	69	120	49	128	148	128	75	186	95	134	159	63	182	119	99	65	76	202	0	161	90
S	111	57	185	251	26	75	163	39	93	140	101	101	226	81	69	159	86	211	161	0	115
T	24	109	84	137	92	49	144	152	112	50	71	78	125	127	49	50	52	128	90	115	0

Remarque : X représente le domicile de Lotte

Figure 2: Image générée par ordinateur au moyen des données de la figure 1 montrant où se situent les villes que Lotte veut visiter



Fenna fronce les sourcils. Elle a reconnu le cas de figure de Lotte : c'est un exemple du problème d'*optimisation combinatoire* appelé *problème du voyageur de commerce*. Dans celui-ci, le nombre de solutions possibles augmente extrêmement rapidement avec le nombre d'entrées, si bien que même pour un nombre modeste, par exemple trouver l'itinéraire le plus court entre 20 villes différentes, le problème est *non traitable computationnellement*.

« Mettons que ton point de départ est X et que les villes que tu veux visiter sont appelées A, B, C, etc. explique Fenna. Si tu ne visites qu'une seule ville, il n'y a qu'une seule solution, c'est-à-dire d'aller de X à A et de revenir. On peut la noter XAX. Pour deux villes, cela donne XABX ou XBAX. Pour trois villes, il y a six solutions :

XABCX XACBX XBACX XBCAX XCABX XCBAX »

« Mais les itinéraires XABX et XBAX sont les mêmes, à part la direction !, fait remarquer Lotte. »

« Exact, répond Fenna. Le nombre de permutations est toujours réduit de moitié parce que chaque itinéraire apparaît deux fois, une fois dans chaque direction. Seulement, voilà le problème : pour chaque ville ajoutée, celle-ci peut être insérée à n'importe quel endroit dans tous les itinéraires possibles. L'ajout d'une énième ville multiplie le nombre de solutions par N. »

« On n'est pas obligé de faire le calcul à la main, dit Lotte en riant. On a un ordinateur ! »

« D'accord, répond Fenna, mais avec 20 villes, on a $20!/2$ permutations. » Elle tape les chiffres dans l'appli calculatrice de son téléphone. « C'est-à-dire 1 216 451 004 088 320 000. » Fenna continue à faire des calculs sur son téléphone, puis finit par relever la tête : « Ça prendrait plus de 30 000 ans pour toutes les tester. »

Tournez la page

Problème du voyageur de commerce

En partant d'une ville donnée, le but est de visiter toutes les autres villes avant de revenir au point de départ. Chacune des villes est reliée aux autres par un chemin direct, les villes devant toutes être visitées exactement une fois. Une séquence de villes se conformant à ces règles représente une *tournée* valide. Dans la version la plus stricte du problème, le but est de trouver la tournée la plus courte.

De multiples méthodes, dont un grand nombre demande des connaissances mathématiques considérables, existent pour résoudre le problème du voyageur de commerce mais jusqu'à présent on ne connait pas d'algorithme qui puisse trouver la solution optimale dans un délai raisonnable. Cependant, il existe des *heuristiques* ayant réussi à trouver des solutions satisfaisantes qui sont suffisamment rapides pour être exploitables. C'est l'une d'entre elles qui fait l'objet de la présente étude de cas.

Algorithmes génétiques

Les algorithmes génétiques imitent le processus de la sélection naturelle pour tenter de développer des solutions aux problèmes autrement non traitables computationnellement.

Les implémentations varient considérablement, mais un algorithme génétique standard comporte les étapes suivantes :

```
Initialisation
While true
    Évaluation
    If (condition de fin) quitter boucle
    Sélection
    Croisement
    Mutation
Output meilleur résultat
```

Le reste de la présente section étudie certaines implémentations d'algorithmes génétiques appliqués au problème du voyageur de commerce.

Initialisation

L'initialisation génère une *population* aléatoire de circuits, chacune d'entre elles étant une solution possible au problème. Pour le problème du voyageur de commerce, avec un point de départ `X`, un circuit possible est :

F	J	G	I	L	C	M	E	S	Q	P	H	T	B	K	N	R	D	O	A

Évaluation

L'algorithme détermine si la *condition de fin* a été remplie et si oui, affiche la meilleure solution trouvée jusqu'à présent et prend fin. Autrement, il détermine la *fitness* (ou *adaptation*) de chaque circuit. Dans le cas de Lotte, il s'agit de calculer la distance totale qu'elle parcourrait pour un circuit donné.

La valeur de fitness est attribuée à chaque circuit au moyen d'une *fonction d'évaluation*. Les circuits sont triés par ordre de fitness. La valeur de fitness la plus élevée est attribuée au circuit le plus courte.

Sélection

Un échantillon de circuits est prélevé parmi la population et placé dans un groupe qui constitue la *population initiale*. Il est possible d'effectuer cette étape de plusieurs façons, mais en général, les circuits les plus adaptés sont plus susceptibles d'être sélectionnés. Voici quatre *stratégies de sélection* courantes :

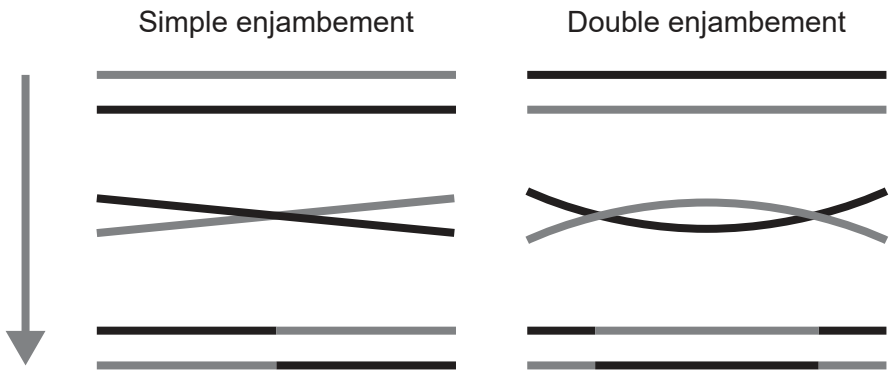
- *sélection proportionnelle à l'adaptation* ;
- *sélection stochastique universelle* ;
- *sélection par tournoi* ;
- *sélection par troncation*.

Un autre facteur de sélection est de décider si la ou les meilleures solutions doivent être transmises avec certitude à la prochaine génération. C'est ce qu'on appelle l'*élitisme*.

Croisement

En biologie, le *croisement* est le mécanisme qui crée les nouveaux chromosomes à partir d'une recombinaison des chromosomes des parents.

Figure 3: Types de croisements



Dans le problème du voyageur de commerce, les simples ou doubles enjambements (**figure 3**) sont problématiques car les villes peuvent être répétées ou omises dans la *descendance*, ce qui résulte en un circuit non valide car il ne visite pas toutes les villes.

Soit le croisement suivant (**figure 4**) entre deux circuits valides d'un problème du voyageur de commerce de 10 villes, dans lequel les parents, `P1` et `P2`, sont recombinés en utilisant un simple enjambement pour produire un nouveau circuit, `F1`.

Figure 4: Exemple de croisement

P1	**B**	**F**	**C**	**A**	**D**	**H**	**G**	**I**	**E**	**J**
P2	G	J	C	D	I	A	E	B	F	H
F1	**B**	**F**	**C**	**A**	**D**	A	E	B	F	H

Le descendant F1 n'est pas un circuit valide car les villes A, B et F sont répétées et les villes G, I and J sont omises. Le même problème se pose avec un double enjambement. Il existe plusieurs mécanismes différents de croisement, chacun tentant de préserver au mieux les caractéristiques des parents.

Une fois qu'on a choisi les circuits qui feront partie de la population initiale, il faut ensuite décider comment ils doivent être recombinés pour produire la descendance. Trois méthodes sont expliquées :
- Croisement partiellement mappé (PMX)
- Croisement d'ordre (OX)
- Croisement de cycle (CX)

Croisement partiellement mappé (PMX)

On choisit une sous-séquence aléatoire dans P1 et on la copie dans F1 :

```
P1: J B F C A D H G I E
P2: F A G D H C E B J I

F1: * * F C A D H * * *
```

On établit les mappages au niveau des éléments entre P1 et P2 pour les villes de P1 qui ne figurent pas encore dans F1. Si la ville correspondante C dans P2 figure déjà dans F1, on continue d'établir le mappage à partir de l'emplacement de C dans P1, puis on répète jusqu'à ce qu'on rencontre une ville qui ne figure pas dans F1 :

J ↔ G B ↔ E G ↔ B I ↔ J E ↔ I

On ajoute les villes restantes de P1 dans F1 et on les remplace à l'aide des mappages :

```
P1: J B F C A D H G I E
P2: F A G D H C E B J I

F1: * * F C A D H * * *
    ↓ ↓         ↓ ↓ ↓
F1: J B F C A D H G I E
    ↓ ↓         ↓ ↓ ↓
F1: G E F C A D H B J I
```

Croisement d'ordre (OX)

On choisit une sous-séquence dans un parent tout en préservant l'ordre relatif des villes restantes de l'autre parent.

On prend une sous-séquence aléatoire S dans P1 et on la copie dans F1. En commençant par l'élément vide situé juste après S dans F1, on copie à partir de P2 toutes les villes qui ne figurent pas encore dans F1 dans l'ordre où elles apparaissent dans P2.

```
P1: J B F C A D H G I E
P2: F A G D H C E B J I

F1: * B F C A D * * * *
    H B F C A D E J I G
```

Croisement de cycle (CX)

Dans `F1`, les villes conservent les positions qu'elles occupent dans au moins un de leurs parents.

```
P1: J B F C A D H G I E
P2: F A G D H C E B J I
```

On choisit la première ville de `P1` et on la copie dans `F1`. On vérifie la ville correspondante dans `P2` (ici, il s'agit de `F`), puis on la copie dans `F1`, à la même position à laquelle elle apparaît dans `P1`. Puis on répète.

```
F1: J B F * A * H G I E
```

Lorsqu'on rencontre une ville qui figure déjà dans `F1` le cycle est terminé. On complète ensuite les trous avec les villes restantes de `P2`.

```
F1: J B F D A C H G I E
```

Mutation

En biologie, la *mutation* représente des erreurs de copie de l'information génétique d'une génération à la suivante. Dans un algorithme génétique, les mutations sont introduites délibérément dans la descendance en fonction du *taux de mutation*.

Discussion

On considère que les avantages des algorithmes génétiques sont leur aptitude à échantillonner simultanément de vastes *paysages de fitness* tout en évitant les *extrema locaux* qui peuvent constituer un piège pour les méthodes *hill-climbing* plus traditionnelles. Les implémentations réussies d'algorithmes génétiques arrivent naturellement à un équilibre entre *exploration* et *exploitation*. Certaines méthodes, par exemple celle du *recuit simulé*, sont en mesure de perfectionner cet équilibre au fur et à mesure de l'évolution de l'algorithme vers la *convergence*. La recherche s'est dernièrement concentrée spécifiquement sur la récompense de la *nouveauté* afin d'encourager les algorithmes à explorer les régions éloignées de l'*espace de problème*. D'autres choix de conception, comme les paramètres initiaux, la *stratégie de sélection* et l'*opérateur de croisement*, ont tous une influence sur les performances de l'algorithme, même s'il n'est généralement pas possible d'en anticiper les effets. La méthode essai-erreur est donc habituellement adoptée.

Tournez la page

Défis rencontrés

Plusieurs défis à relever sont associés aux algorithmes génétiques. Ceux-ci incluent :

- comprendre le rôle de la convergence dans les algorithmes génétiques et les facteurs qui ont une incidence sur celle-ci ;
- évaluer l'utilisation et l'implémentation des stratégies de sélection proportionnelle à l'adaptation (*fitness*), de sélection par tournoi et de sélection par troncation dans les algorithmes génétiques ;
- discuter les différentes solutions à la question de l'inadaptation des stratégies de simple enjambement au problème du voyageur de commerce. En particulier :
  - pourquoi elles sont nécessaires ;
  - comment elles sont appliquées ;
  - comment les caractéristiques parentales sont préservées ;
  - quelles autres méthodes sont disponibles.
- comprendre les avantages et les inconvénients des algorithmes génétiques par rapport aux autres méthodes appliquées au problème du voyageur de commerce et aux problèmes d'optimisation combinatoire en général.

Les candidats n'ont pas besoin de connaître en détail l'implémentation des autres méthodes.

Terminologie ne figurant pas dans le guide

Circuit (*tour*)
Classement (*ranking*)
Condition de fin (*termination condition*)
Convergence (*convergence*)
Convergence prématurée (*premature convergence*)
Croisement / opérateur de croisement (*crossover* / *crossover operator*)
Descendance (*offspring*)
Élitisme (*elitism*)
Espace de problème (*problem space*)
Exploration et exploitation (*exploration vs exploitation*)
Extrema locaux (*local extrema*)
Fitness (adaptation) / fonction d'évaluation / paysage de fitness (*fitness* / *fitness function* / *fitness landscape*)
Heuristique (*heuristic*)
Hill-climbing
Méthode de force brute (*brute force approach*)
Mutation / taux de mutation (*mutation* / *mutation rate*)
Non traitable computationnellement (*computational intractability*)
Optimisation (*optimization*)
Optimisation combinatoire (*combinatorial optimization*)
Paramètres d'initialisation (*initialization parameters*)
Population (*population*)
Population initiale (*mating pool*)
Recherche de la nouveauté (*novelty search*)
Recuit simulé (*simulated annealing*)
Sélection par tournoi (*tournament selection*)
Sélection par troncation (*truncation selection*)
Sélection proportionnelle à l'adaptation (*roulette wheel selection*)
Sélection stochastique universelle (*stochastic universal sampling*)
Stratégie de sélection (*selection strategy*)

Références :