Laboratoire 2 | Algorithme génétique

420-C52 | Données, mégadonnées et intelligence artificielle I

Table des matières

Introduction

Ce laboratoire consiste à réaliser une application capable de résoudre un problème d'optimisation géométrique.

Outre les considérations techniques et algorithmiques, on désire que l'application soit à forte teneur éducative au sens où la visualisation du problème est au centre de l'opération. De plus, l'étudiant doit pouvoir faire la distinction entre les éléments génériques et spécifiques de cette application pour cibler un développement adapté et modulaire.

Objectifs généraux

Les objectifs principaux de ce laboratoire sont :

- implémenter l'algorithme génétique de façon modulaire, flexible et générique;
- utiliser l'algorithme génétique réalisé pour résoudre un problème d'optimisation géométrique;
- réaliser une application complète supportant ces fonctionnalités.

Objectifs spécifiques

Plus spécifiquement, ce laboratoire vise à :

- bien comprendre la nature du problème afin de déterminer adéquatement tous les paramètres liés à résolution de ce dernier par l'algorithme génétique;
- implémentation de l'algorithme génétique;
- utilisation de l'algorithme génétique pour résoudre le problème présenté,
- réalisation d'une interface utilisateur assemblant les diverses parties de votre application.

Présentation de la problématique

Problème d'optimisation géométrique

Dans plusieurs domaines du génie, les problématiques d'optimisation font partie du quotidien. La complexité croissante des problèmes adressés nécessite des techniques de plus en plus sophistiquées. Ainsi, l'optimisation est une branche importante de la mise en place de plusieurs systèmes modernes.

Ce projet consiste à résoudre un problème d'optimisation géométrique qui s'explique simplement et pour lequel il n'existe pas de solution triviale. On vous demande de trouver la transformation géométrique permettant de disposer la plus grande forme géométrique sur une surface parsemée d'obstacle.

Plus spécifiquement, vous devez résoudre le problème suivant :

- vous disposez d'un canevas à deux dimensions (surface rectangulaire) :
  - le canevas est défini par sa largeur et sa hauteur
- sur le canevas se trouvent n point à deux dimensions correspondant à des obstacles :
  - $n \geq 0$
  - chaque point est disposé aléatoirement sur le canevas
- vous disposez d'une forme géométrique quelconque à deux dimensions :
  - la forme est définie par un polygone de p côté :
    - $p \geq 3$

        - le polygone peut être convexe ou concave, mais ne doit pas se croiser lui-même
    - il est possible d'effectuer ces transformations sur le polygone :
        - translation à deux dimensions;
        - rotation;
        - homothétie (« scaling »);
    - vous ne pouvez pas déformer la forme d'aucune façon;
    - la forme peut toucher au contour du canevas et des obstacles;
    - la forme ne peut dépasser la zone rectangulaire du canevas ou posséder un obstacle à l'intérieur;
    - on cherche la transformation qui maximise la surface de la forme à l'intérieur du canevas sans entrer en contact avec les obstacles.

Mise en place d'un scénario

Pour chaque résolution de problème, trois paramètres fondamentaux doivent être mis en place et rester immuables tout au long de la résolution :

- la taille du canevas;
- le nombre d'obstacles et la disposition de ces derniers;
- la forme géométrique.

La forme peut subir une ou plusieurs transformations affines (translation, rotation et homothétie), mais ne peut être modifiée autrement. Par exemple, si la forme ressemble à un « L », elle le restera jusqu'à la fin de la résolution. Toutes les proportions du « L » seront gardées : longueur relative entre la barre verticale et la barre horizontale ainsi que l'épaisseur relative du trait.

Objectifs de la résolution

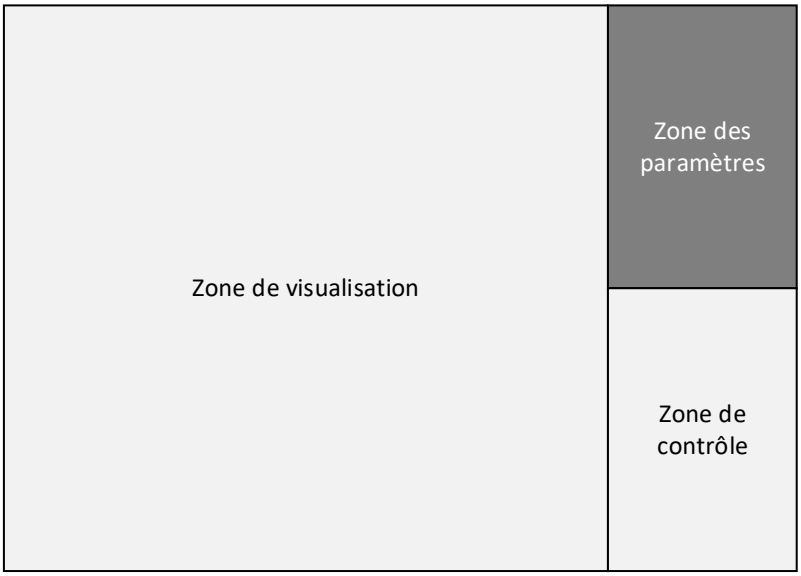
Au final, il est attendu que la forme soit disposée de façon telle à maximiser sa taille sans enfreindre les règles énoncées.

Approche opérationnelle du logiciel et interface utilisateur

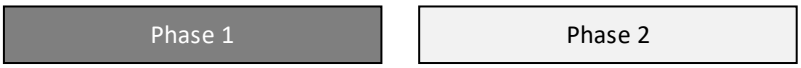
Le logiciel doit offrir une résolution du problème en deux phases :

1. D'abord, permettre à l'utilisateur de définir les paramètres du problème :
    a. Taille du canevas.
    b. Nombre de points (la disposition est automatique et aléatoire).
    c. La forme à disposer.
2. Ensuite, procéder à la résolution du problème. Pendant cette phase, ces éléments sont à considérer :
    a. Vous avez les contrôles nécessaires permettant de :
        i. faire un pas de simulation
        ii. démarrer la simulation en continu
        iii. si la simulation s'exécute en continu, il est possible de faire pause et de reprendre la simulation
        iv. il est possible de réinitialiser la simulation.
    b. Une visualisation de la simulation présente la solution le plus récente (permet de voir l'évolution) incluant ces éléments :

i. le canevas
ii. les obstacles
iii. toutes les solutions en évolution
iv. la meilleure forme avec une couleur distinctive
v. En tout temps, on peut voir les informations numériques suivantes :
    1. numéro de l'itération courante
    2. surface de la meilleure solution.



Présentation suggérée



Ajouts personnels

Outre la résolution du problème ainsi que l'implémentation de l'algorithme génétique, **vous devez faire *x*[1] ajouts personnels au projet**. Ces ajouts doivent être orientés de la façon suivante :

- Au moins un ajout doit être orienté vers une fonctionnalité supplémentaire du point de vue de l'utilisateur.
- Au moins un ajout doit être orienté vers l'algorithme génétique et proposer une amélioration pour au moins un aspect des différents processus.

Les ajouts à réaliser ne sont pas nécessairement des éléments complexes ou nécessitant beaucoup de temps à produire. Les ajouts sont d'abord jugés sur leur pertinence lié au projet et ensuite sur la qualité de l'implémentation.

[1] *x* représente le nombre d'ajouts personnels à réaliser par équipe. Il doit y avoir 2, 3 et 4 ajouts personnels pour les équipes de 3, 4 et 5 étudiants respectivement.

Contraintes liées au développement logiciel

Vous devez porter une attention particulière à cette partie, car elle constitue une ligne guide sur la réalisation du travail :

- Il est attendu que votre travail soit modulaire et organisé de façon à être réutilisable.
- Dans cet esprit, vous devez créer au minimum trois classes (ou groupes de classes) visant à représenter ces concepts :
    1. Une classe hautement générique de l'algorithme génétique.
    2. Une classe organisant la résolution spécifique de ce problème (utilisant la classe de l'algorithme génétique).
    3. La classe de l'interface utilisateur effectuant l'opération de la classe de résolution de problème.
- Vous devez mettre la structure documentaire suivante :
    - Vous faites un effort particulier pour mettre en pratique l'autodocumentation de votre code (noms de classes, de types, de fonctions et de variables pertinentes).
    - Tous vos fichiers *.h présente un en-tête incluant ces informations :
        - Que contient le fichier : une ligne très sommaire.
        - Qui sont les membres de l'équipe.
        - La date de création.
    - Avant chaque classe, vous mettez une courte description de cette dernière. Sachez qu'on retrouve beaucoup plus de commentaires dans la zone déclarative (*.h) que dans la zone de définition (*.cpp).
    - Pour le reste, vous mettez quelques commentaires pour des raisons pertinentes.
    - Vous évitez de mettre des commentaires creux. Par exemple mettre en commentaire après une boucle for que vous faites un parcours. Les commentaires doivent complémenter le code.
    - Vous mettez en commentaires toutes les références que vous avez utilisées (sauf pour les références évidentes – par exemple la documentation en ligne de la classe `QLabel`).

Contraintes techniques

Pour la réalisation de ce projet, vous devez respecter ces contraintes :

- Ce projet doit être réalisé en C++ avec la librairie Qt en utilisant l'IDE Visual Studio.
- Ce projet doit être réalisé en équipe de 3 ou 4 étudiants.
- Vous avez 2 semaines pour réaliser ce projet.

Retour sur l'algorithme génétique

L'algorithme génétique est une heuristique (stratégie d'évaluation rapide, mais pas nécessairement optimale) inspirée du processus de sélection naturelle décrit par Darwin. Cet algorithme fait partie de la grande famille des algorithmes évolutifs (algorithmes évolutionnaires).

L'algorithme est basé sur la prémisse où il existe une population de solutions candidates à un problème donné. Cette population évolue en créant de nouvelles solutions générées à partir des solutions existantes

les plus performantes. Progressivement, les solutions sont de plus en plus adaptées à la résolution du problème.

Une métrique de la performance des solutions candidates permet de déterminer lesquelles sont plus intéressantes et susceptibles d'être utilisées pour la création de nouvelles solutions.

L'algorithme utilise un vocabulaire issu de la biologie afin de représenter les constituants et les étapes du processus de résolution. Évidemment, les termes utilisés représentent une simplification des processus biologiques réels, mais constituent une infrastructure algorithmique générique permettant la résolution de problèmes très variés.

L'algorithme génétique présente des qualités très intéressantes :

- il est facile et intuitif à comprendre
- à priori, il ne requiert pas de connaissances avancées en mathématiques;
- il est adaptable, autant par :
    - les structures internes de représentation des problèmes et des algorithmes appliqués pour chaque processus;
    - sa possibilité à résoudre tout type de problème.

Toutefois, il reste sensible à la façon de formuler les éléments cruciaux du problème. Ainsi, il peut être parfois difficile d'obtenir des résultats intéressants avec des problèmes complexes de hautes dimensions.

Vocabulaire

On divise le vocabulaire de l'algorithme génétique en deux catégories : les constituants et les processus. On présente aussi quelques éléments de vocabulaire représentant la problématique.

La problématique

- Le problème :
    - C'est l'élément central du projet. Il détermine tous les paramètres sous-jacents et l'objectif à atteindre.
    - C'est lui qui permet de définir tous les paramètres de l'algorithme génétique. Il établit aussi la performance des solutions obtenues.
- Les intrants :
    - Les intrants sont les données qui entrent dans le système.
    - Ce sont généralement des éléments imposés issus du problème et qui doivent être utilisés (pour différentes parties de l'algorithme).
- Les extrants :
    - Les extrants sont les données qui sortent du système.
    - Ils sont généralement le résultat du processus de résolution. Ils correspondent à la solution finale attendue.

Constituants de l'algorithme génétique

- Une solution :
    - Une solution représente une instance d'extrant.
    - Les solutions sont des hypothèses représentant une réponse possible au système.
    - Elles sont en fait un point dans l'espace des solutions.

    - Une solution n'est pas bonne ou mauvaise en soi, toutefois il est possible de déterminer si elle est plus ou moins performante qu'une autre.
- La population :
    - La population constitue un ensemble de plusieurs solutions.
    - Elle permet de contenir plusieurs hypothèses simultanément et, en les utilisant toutes adéquatement, de trouver de meilleures solutions.
- Le phénotype :
    - Le phénotype est la représentation d'une solution dans l'espace de solution.
    - Il correspond aux traits observables d'une solution. C'est-à-dire, aux caractéristiques compréhensibles par un humain de la solution.
    - Par exemple : les yeux bleus.
- Le génotype :
    - Le génotype est la représentation encodée (de plus bas niveau) d'une solution.
    - Il correspond aux traits non observables (ou difficilement observables) de la solution.
    - Par exemple : 0xFF0000FF (pour la représentation hexadécimale ARGB32 de la couleur bleue). En fait, cette représentation n'est pas tout à fait juste. Il serait plus juste de représenter les 32 bits individuels utilisés pour décrire la couleur.
- Le gène :
    - Un gène correspond à un trait du problème.
    - Il correspond généralement à une donnée issue d'une seule dimension de l'espace de solution.
    - Le gène est la version encodée de l'information.
- Le chromosome :
    - Le chromosome est constitué de tous les gènes qui forment une solution.
    - Autrement dit, le chromosome représente une solution encodée.
    - Le chromosome est à la base de tout le processus de l'algorithme génétique. C'est lui qui rend l'algorithme génétique générique.
- Géniteurs (parents) :
    - Ce sont des solutions qui sont utilisées pour produire une nouvelle solution.
- Progénitures (enfants) :
    - Ce sont les nouvelles solutions issues du processus de création de l'algorithme génétique.

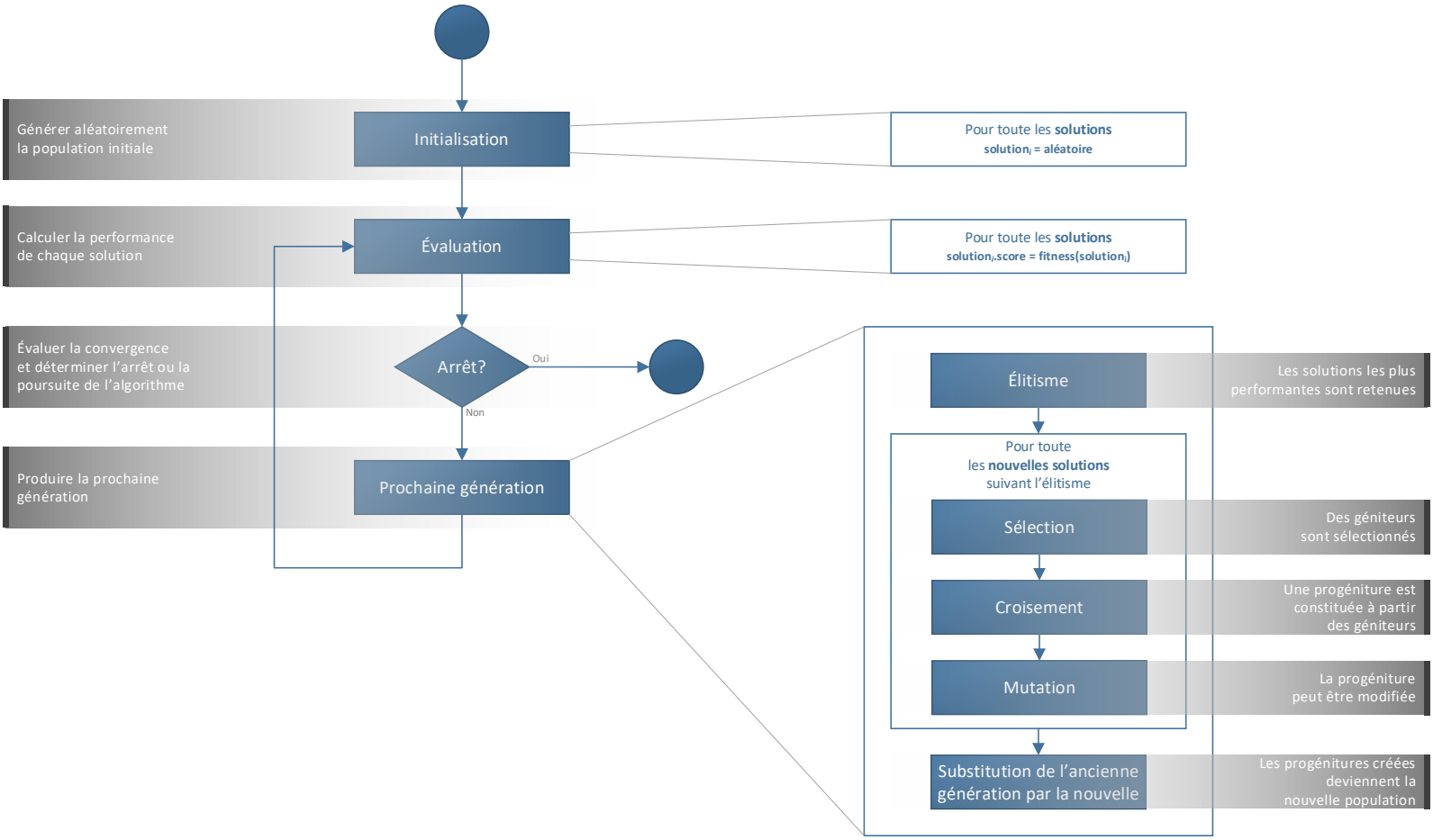
Processus de l'algorithme génétique

- Encodage :
    - C'est le processus permettant de passer du phénotype au génotype.
    - Il permet de créer le chromosome à partir de données utilisables.
- Décodage :
    - C'est le processus permettant de passer du génotype au phénotype.
    - Il permet de créer des données de l'espace de solution à partir d'un chromosome.
- Sélection :
    - La sélection est le processus qui détermine les parents requis à la création d'une nouvelle solution.
    - Ce processus vise habituellement à favoriser les solutions les plus performantes.
    - Généralement, deux géniteurs sont choisis pour produire une progéniture.

- Croisement :
    - Le croisement est l'opération qui consiste à produire une progéniture à partir de géniteurs.
- Mutation :
    - La mutation est le processus permettant d'apporter une modification à une progéniture.
    - Cette modification est généralement incontrôlée et permet la poursuite exploratoire de l'espace de solution de façon à sortir d'un extremum local.
- Fonction objective (*fitness*) :
    - La fonction objective ou simplement « *la fitness function* » est la fonction permettant d'évaluer la performance relative d'une solution.
    - C'est elle qui détermine si le problème en est un de minimisation ou de maximisation.

Résumé de l'algorithme

Le schéma suivant illustre les grands principes de l'algorithme. Il est important de garder en tête que toutes les sous-parties de l'algorithme sont génériques.



Paramètres de l'algorithme

Considérant que même les processus sont ajustables, la résolution d'un problème avec l'algorithme génétique est divisée en trois grandes familles de paramètres :

1. Définition du problème :
    - Les paramètres de l'espace de solution. le phénotype
    - La forme que prend le chromosome. le génotype
    - Les fonctions d'encodage et de décodage. le passage du phénotype au génotype et inversement
    - La fonction objective (« fitness »). la fonction d'évaluation de la performance d'une solution
2. Les paramètres intrinsèques de l'algorithme :
    - La taille de la population. détermine le nombre de solutions participant à l'évolution
    - La taille de l'élitisme. détermine la quantité des meilleures solutions maintenues à chaque génération
    - Le nombre de générations max détermine un critère d'arrêt correspondant au budget disponible
    - Le taux de mutation. détermine la probabilité de mutation pour une nouvelle progéniture
3. Les stratégies algorithmiques :
    - Initialisation. mise en place de la population initiale
    - Évaluation. application du calcul de performance
    - Sélection. choix de géniteurs
    - Croisement. production d'une progéniture
    - Mutation. poursuite exploratoire de l'espace de solution

Pour le troisième groupe, il existe des stratégies génériques proposées à même les fondements de l'algorithme génétique.

Stratégies génériques des opérations fondamentales

On présente ici les outils fondamentaux de l'algorithme génétique : comment sont réalisées les opérations génériques de l'algorithme.

Encore une fois, il existe plusieurs variantes possibles en fonction du type d'encodage et de la représentation du problème. Par exemple, le problème peut être représenté par des nombres entiers, des réels, des étiquettes, des positions à permuter, des arbres symboliques, etc. De plus, il est tout à fait possible de mélanger toutes ces représentations dans le même chromosome.

Pour les exemples qui suivent, une présentation sommaire des deux représentations numériques (entiers et réels) est faite.

Il est important de savoir que l'algorithme fondamental n'utilise que la représentation correspondant à une chaîne de bits. C'est-à-dire que toutes les informations du problème sont amalgamées dans une suite de 0-1 qui sont utilisés sans distinction par les différents processus internes. Ainsi, chaque bit est anonyme pour toutes les étapes du processus sauf pour les fonctions d'encodages et de décodage. Cette approche permet la généricité de l'algorithme.

Mise en situation

Pour la suite, on suppose deux problèmes pour lesquels un type d'encodage différent est utilisé. Dans le premier cas, on utilisera un encodage par chaîne de bits et ensuite par une suite de réels.

Premier problème, supposons qu'on désire déterminer les paramètres suivants :

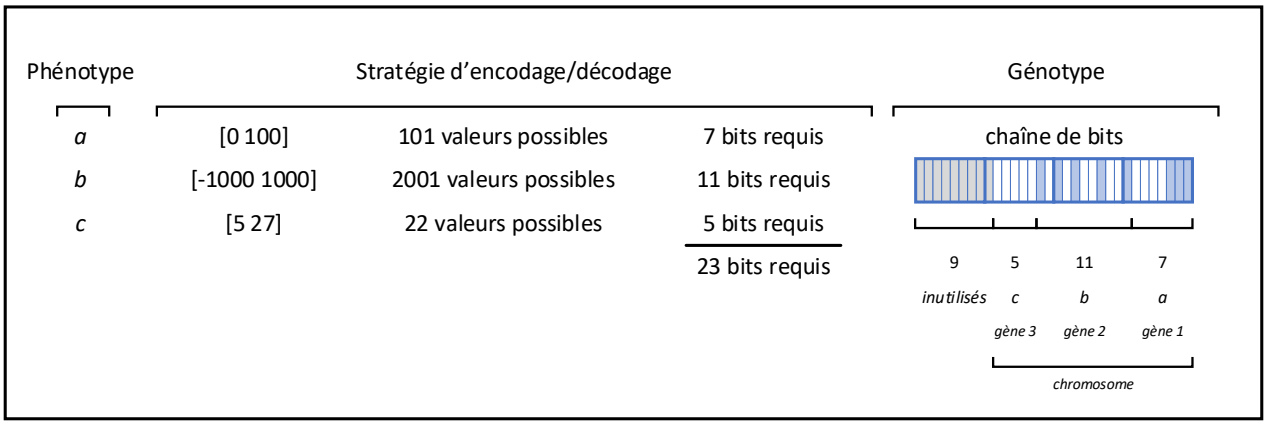
- a : un nombre entier inclue dans l'intervalle [0 100]
- b : un nombre entier inclue dans l'intervalle [-1000 1000]
- c : un nombre entier inclue dans l'intervalle [5 27]

Deuxième problème, supposons qu'on désire déterminer les paramètres suivants :

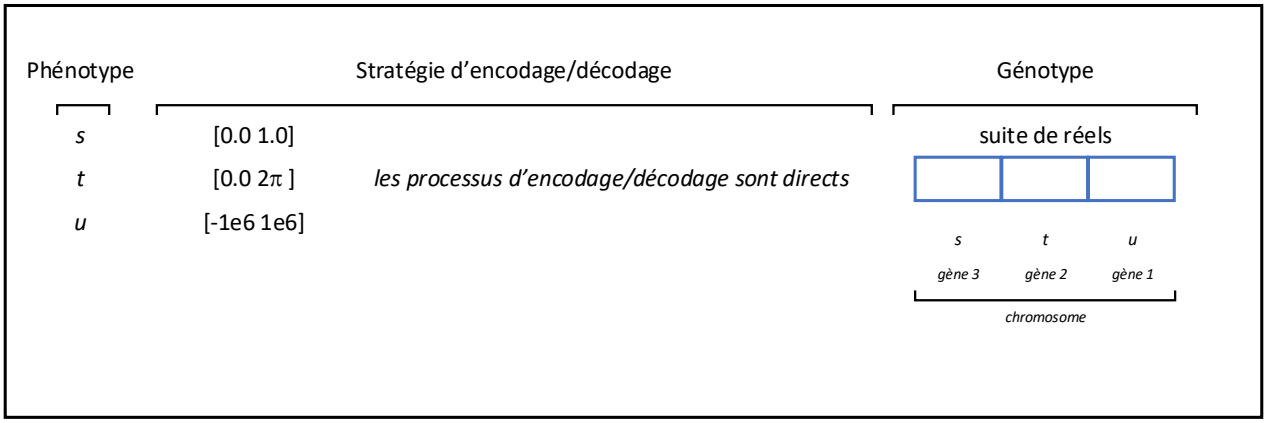
- *s* : un nombre réel inclue dans l'intervalle [0.0 1.0]
- *t* : un nombre réel inclue dans l'intervalle [0.0 2π]
- *u* : un nombre réel inclue dans l'intervalle [-1 000 000.0 1 000 000.0]

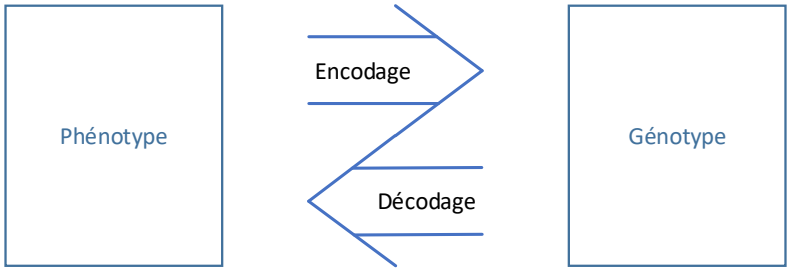
Représentation, encodage et décodage

Problème 1 | Représentation par chaîne de bits



Problème 2 | Représentation par nombres réels





Initialisation

- Représentation par chaîne de bits :
  - On détermine aléatoirement l'état de chaque bit utilisé

- Représentation par une suite de réels :
    - On détermine chaque réel par un nombre aléatoire inclus dans les intervalles déterminés

Évaluation

L'évaluation consiste à appliquer la fonction objective sur chacune des solutions et de stocker leur performance relative.

- Représentation par chaîne de bits :
    - Il est possible de créer une fonction objective autant sur le phénotype que le génotype.
    - Sur le phénotype :
        - Avantage : représentation simplifiée de la fonction objective.
        - Désavantage : devoir décoder le chromosome avant de pouvoir calculer la fonction objective.
    - Sur le génotype :
        - Avantage : performance accrue en ne devant pas appliquer le décodage.
        - Désavantage : une telle fonction est souvent abstraite et parfois difficile (voir impossible) à faire réellement sans décodage direct ou indirect.
    - On privilégiera la première approche.
- Représentation par une suite de réels :
    - On applique la fonction objective directement sur le phénotype.

Élitisme

L'élitisme s'applique simplement en copiant dans la nouvelle population les *n* solutions les plus performantes de la génération actuelle (où *n* correspond au paramètre prédéterminé du nombre d'élite).

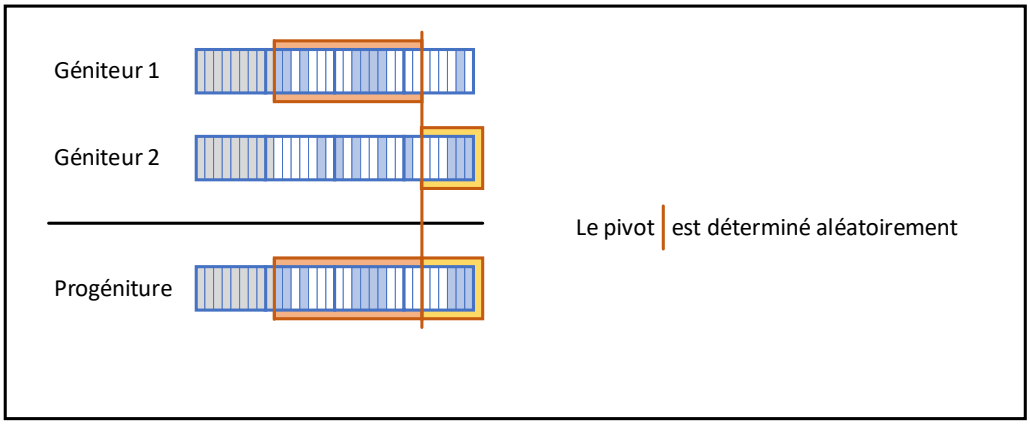
Sélection

La sélection consiste à déterminer les géniteurs d'une nouvelle solution (la progéniture).

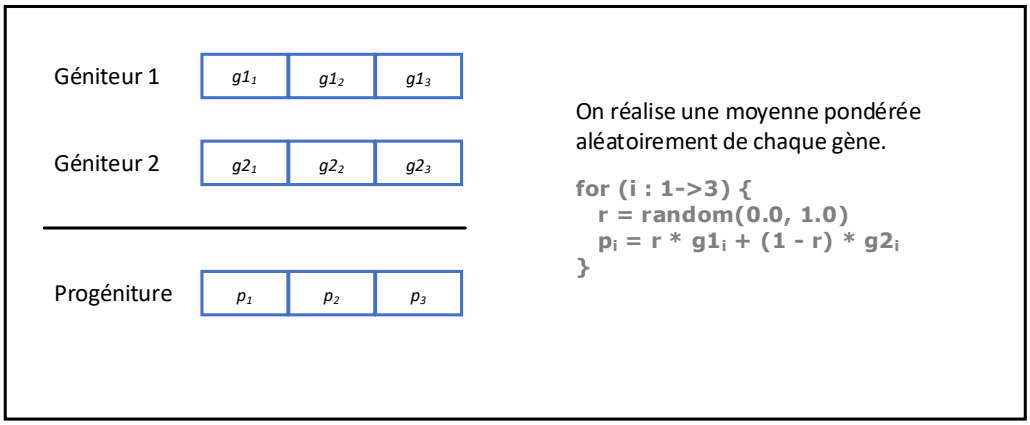
Il n'existe pas de méthode officiellement fondamentale. Toutefois, la méthode « *Roulette Wheel* » est généralement utilisée comme approche générique. Cette approche utilise le poids relatif de chaque individu pour déterminer leur probabilité de sélection. Le choix des géniteurs se fait ensuite par une sélection aléatoire de cette distribution.

Croisement

Problème 1 | Représentation par chaîne de bits



Problème 2 | Représentation par nombres réels



Mutation

La mutation s'effectue en deux étapes :

1. On détermine s'il y a mutation. Pour chaque progéniture, on génère un nombre aléatoire et détermine si ce dernier est inférieur ou égal au taux de mutation préalablement défini.
2. Si la mutation est effective, alors on applique la stratégie de mutation :
    a. Représentation par chaîne de bits : on bascule un bit aléatoirement.
    b. Représentation par une suite de réels : on modifie un seul réel par un nombre aléatoire inclus dans les intervalles déterminés.

On remarque que les deux approches proposent une différence importante : dans le premier cas, seulement un bit est modifié alors que dans le deuxième, on modifie un gène au complet. L'effet est similaire, mais plus significatif dans le 2^e cas.

Quelques outils informatiques

Les classes suivantes vous seront utiles.

QImage & QPixmap

Les classes `QImage` et `QPixmap`, que vous connaissez déjà, permettent de produire les images nécessaires à la visualisation. Vous pouvez utiliser l'une ou l'autre de ces classes sans problème. Toutefois, considérant qu'il vous est demandé uniquement d'afficher à l'écran l'image et non pas de la sauvegarder, la classe `QPixmap` devrait être privilégiée.

C'est la classe `QPainter` qui vous permettra de dessiner sur votre image.

N'oubliez pas que vous devez utiliser le widget `QLabel` pour afficher une image à même l'interface graphique.

QPointF, QRectF, QPolygonF & QPolygonUtilities

La classe `QPointF` représente un point en 2D et propose plusieurs fonctions utilitaires.

La classe `QRectF` représente un rectangle et offre aussi plusieurs fonctionnalités intéressantes telles que :

- la détection d'un point à l'intérieur du rectangle `QRectF::contains`
- la détection d'intersection entre deux rectangles `QRectF::intersects`
- la détection d'un rectangle à l'intérieur du rectangle `QRectF::contains` (surcharge)

La classe `QPolygonF` représente un polygone et offre certains outils intéressants comme :

- la construction du polygone à partir de plusieurs points `QPointF`
- la détection d'un point à l'intérieur du polygone avec `QPolygonF::containsPoint`
- la boîte capable du polygone (le « bounding box ») avec `QPolygonF::boundingRect`

Il est possible d'afficher toutes ces primitives géométriques avec la classe `QPainter` avec les fonctions `QPainter::drawPoint`, `QPainter::drawRect` et `QPainter::drawPolygon`.

Malheureusement, certaines fonctions pratiques de géométrie ne sont pas disponibles telles que les calculs du périmètre et de l'aire. À cet effet, l'enseignant vous donne une petite classe utilitaire `Q2DGeometrytilities`.

Voici un exemple :

```
QPointF point(10.0, 25.0);                          // défini un point

QRectF rectangle(-100.0, -100.0, 200.0, 200.0);     // défini un rectangle

QPolygonF polygonalShape;                           // défini un polygone vide
polygonalShape << QPointF(-10.0, 0.0)               // ajoute quatre sommets définissant un
               << QPointF(0.0, 20.0)                // losange
               << QPointF(10.0, 0.0)
               << QPointF(0.0, -20.0);

// calcul le périmètre du polygone
double perimeter{ Q2DGeometryUtilities::perimeter(polygonalShape) };

// calcul l'aire du polygone
double area{ Q2DGeometryUtilities::area(polygonalShape) };

// détermine si le point entre en collision avec le rectangle
bool isPointCollidingRectangle{ rectangle.contains(point) };

// détermine si le point entre en collision avec le polygone
bool isPointCollidingPolygon{ polygonalShape.contains(point) };
```

QPainter

`QPainter` est une classe permettant de dessiner sur diverses surfaces graphiques de la librairie `Qt`. Cette classe utilitaire permet de dessiner plusieurs primitives telles que :

- point
- ligne
- rectangle (un carré étant un rectangle spécifique)
- ellipse (un cercle étant une ellipse spécifique)
- polygone
- image (`QImage` ou `QPixmap`)
- texte
- et autres.

Il existe un concept récurrent à souligner pour l'utilisation de cette classe. Comme plusieurs librairies de dessin, l'action de dessiner est séparée de l'action définissant les caractéristiques visuelles telles que la couleur de remplissage et la couleur du trait.

Il existe ainsi deux mutateurs permettant de modifier les caractéristiques visuelles de l'objet `QPainter` avant qu'il ne dessine :

- `QPainter::setBrush(const QBrush & brush)` : détermine tous les paramètres de remplissage tels que la couleur. Voir la classe `QBrush`.
- `QPainter ::setPen(const QPen & pen)` : détermine tous les paramètres du contour tels que la couleur et l'épaisseur de ce dernier. Voir la classe `QPen`.

La stratégie consiste donc à modifier les caractéristiques visuelles avant de faire le dessin.

Voici un exemple :

```
QSize size(800, 600);                                   // défini l'objet size de 800 x 600 px
QPixmap pixmap(size);                                   // défini l'objet pixmap de la taille déf.

QPainter painter(&pixmap);                              // défini l'objet painter associé à l'objet
                                                        // pixmap – painter dessine sur le pixmap


painter.setPen(QPen(Qt::blue, 1.0));                    // détermine le trait bleu à 1 pixel
painter.setBrush(QBrush(QColor(255, 128, 0)));          // détermine la couleur interne orange

QPointF point(100.0, 150.0);                            // défini un point
painter.drawPoint(point);                               // dessine le point

QRectF rectangle(10.0, 10.0, 75.0, 125.0);              // défini un rectangle
painter.drawRect(rectangle);                            // dessine le rectangle

QPolygonF shape;                                        // défini un polygone
shape << QPointF(0.0, 0.0)
      << QPointF(100.0, 0.0)
      << QPointF(0.0, 100.0);
painter.drawPolygon(shape);                             // dessine le polygone
```

QTransform

La classe `QTransform` permet d'appliquer des transformations affines sur les primitives géométriques telles que la translation, la rotation et l'homothétie. Les fonctions `QTransform::translate`, `QTransform::rotate` et `QTransform::scale` détermine les transformations à appliquer. Les fonctions `QTransform::map` (plusieurs surcharges) appliquent les transformations définies sur les primitives géométriques.

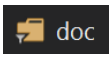
Il est aussi pratique de savoir que la classe `QPainter` utilise à l'interne la classe `QTransform` et qu'il est possible de l'utiliser.

Voici un exemple d'utilisation de la classe `QTransform` :

```
QPointF point(10.0, 5.0);                           // déclare un objet point

QPolygonF shape;                                    // déclare un objet polygone
shape << QPointF(0.0, 0.0)                          // ajoute des points au polygone de façon
      << QPointF(-1.0, 1.0)                         // à créer une flèche pointant à droite
      << QPointF(2.5, 0.0)
      << QPointF(-1.0, -1.0);

QTransform transform;                               // déclare un objet de transformation

transform.translate(100.0, -150.0);                 // applique une translation de 100, -150
transform.rotate(45.0);                             // applique une rotation de 45°
transform.scale(2.0, 2.0);                          // applique une homothétie de 2x

// applique la transformation au polygone
// attention, dans cet exemple on efface le polygone original
shape = transform.map(shape);

// déclare un nouveau point et le défini au point existant transformé
QPointF transformedPoint(transform.map(point));
```

Rapport

Vous devez ajouter dans votre solution Visual Studio un filtre nommé `doc` dans lequel se trouve un fichier nommé `rapport.txt`.



Veuillez répondre brièvement à ces questions :

1. Qui sont les membres de l'équipe?
2. Pour chacun des ajouts personnels :
    a. une courte description de l'objectif
    b. la catégorie :
        i. application
        ii. algorithme génétique
    c. dans le cas des ajouts à l'algorithme génétique, expliquez en quoi ils améliorent les résultats de convergence.
3. Expliquer en quoi votre implémentation de l'algorithme génétique est générique. Comment serait-il possible de l'utiliser pour un autre problème?

Évaluation

Ce travail est évalué par ces critères (en ordre d'importance) :

1. Qualité des paramètres de résolution du problème :
    - identification de l'espace de solution
    - définition du chromosome
    - fonctions d'encodage et de décodage
    - fonction objective
    - les paramètres spécifiques de l'algorithme génétique compte pour moins :
        - taille de la population
        - taille de l'élitisme
        - taux de mutation
2. Implémentation de l'algorithme génétique.
3. Convergence et des résultats obtenus
4. Ajouts personnels
5. Qualité de la modularité et de la réutilisabilité (surtout de la classe gérant l'algorithme génétique)
6. Implémentation de l'interface usager.
7. Qualité générale du code C++ pour les éléments couverts pendant le cours.

Stratégie d'évaluation

L'évaluation se fera en 2 parties. D'abord, l'enseignant évaluera le projet remis et assignera une note de groupe pour le travail. Ensuite, chaque équipe devra remettre un fichier Excel dans lequel sera soigneusement reportée une cote représentant la participation de chaque étudiant dans la réalisation du projet. Cette évaluation est faite en équipe et un consensus doit être trouvé.

Une pondération appliquée sur ces deux évaluations permettra d'assigner les notes finales individuelles.

Ce projet est long et difficile. Il est conçu pour être réalisé en équipe. L'objectif est que chacun prenne sa place et que chacun laisse de la place aux autres.

Ainsi, trois critères sont évalués :

- **participation** (présence en classe, participation active, laisse participer les autres, pas toujours en train d'être sur Facebook ou sur son téléphone, concentré sur le projet, pas en train de faire des travaux pour d'autres cours, …)
- **réalisation** (répartition du travail réalisé : conception, modélisation, rédaction de script, documentation, …)
- **impact** (débrouillardise, initiative, amène des solutions pertinentes, motivation d'équipe, …)