Projet 2 | Algorithme génétique

420-C52 | Données, mégadonnées et intelligence artificielle I

Table des matières

Introduction

Ce projet consiste à réaliser une application capable de résoudre trois problèmes d'optimisation différents.

Outre les considérations techniques et algorithmiques, on désire que l'application soit, encore une fois, à forte teneur éducative au sens où la visualisation des problèmes est au centre des opérations. 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 projet sont :

- utiliser l'algorithme génétique proposé pour résoudre les problèmes d'optimisation sélectionnés;
- réaliser une application complète supportant ces fonctionnalités.

Objectifs spécifiques

Plus spécifiquement, ce projet vise à :

- bien comprendre la nature des problèmes abordés afin de déterminer adéquatement les paramètres liés à résolution par l'algorithme génétique;
- utilisation et amélioration d'une petite librairie implémentant l'algorithme génétique;
- utilisation de l'algorithme génétique pour résoudre les problèmes présentés,
- réalisation d'une interface utilisateur assemblant les diverses parties de l'application.

Présentation des outils informatiques mis à votre disposition

Vous avez à votre disposition une petite bibliothèque réalisant une implémentation simple de l'algorithme génétique ainsi qu'un squelette d'application permettant de visualiser simplement la résolution de problèmes.

Bibliothèque de l'algorithme génétique

Les caractéristiques principales de cette bibliothèque sont :

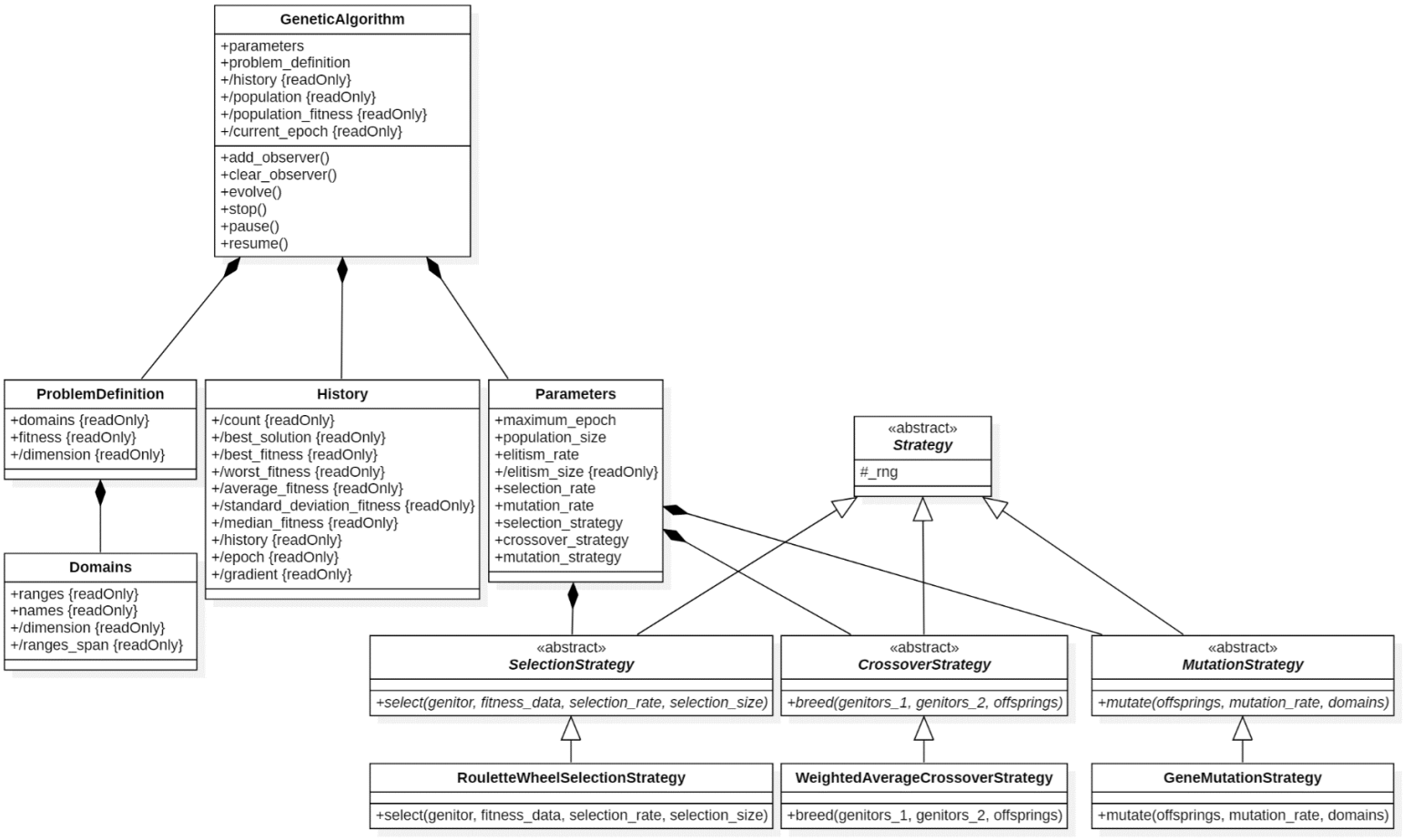
- L'algorithme génétique est basé sur une représentation par des nombres réels.
- La représentation par nombres réels implique que les stratégies d'encodage et de décodage sont inutiles car les génotypes et phénotypes sont confondus.
- Basée sur la bibliothèque `NumPy`.
- Utilise quelques patrons de conception pour son implémentation : stratégie et observateur notamment.
- La classe `ProblemDefinition` permet de déterminer les paramètres du problème à résoudre.
- La classe `Domains`, utilisée par la classe `ProblemDefinition`, standardise la définition de la dimension du problème et, pour chacune d'entre elle, l'intervalle des valeurs possibles.
- La classe `Parameters` permet de déterminer tous les paramètres intrinsèques de l'algorithme.

- Trois algorithmes fondamentaux sont basés sur le patron de conception stratégie et, via trois classes abstraites, rendent possible la spécialisation de ces opérations. De plus, une implémentation de base est disponible pour chacun des cas :

Opérations	Classe de base	Implémentation disponible par défaut
Sélection	`SelectionStrategy`	`RouletteWheelSelectionStrategy`
Croisement	`CrossoverStrategy`	`WeightedAverageCrossoverStrategy`
Mutation	`MutationStrategy`	`GeneMutationStrategy`

- La classe `GeneticAlgorithm` encapsule tous les éléments et le moteur de résolution.
- **IMPORTANT** : L'algorithme réalise toujours une maximisation. C'est-à-dire que si votre problème en est un de minimisation, il faudra le transformer pour en faire un de maximisation. À cet effet, la fonction de « fitness » attendue doit toujours produire :
  - des valeurs positives (≥ 0)
  - au moins une valeur strictement positive parmi toute la population (> 0)
- Le fichier `gacvm.py` possède tous ces éléments. Seulement 2 dépendances externes sont requises :
  - `numpy`
  - `umath` (fournie avec le projet)

Le schéma de classe UML suivant aide à mieux comprendre les constituants importants de cette petite bibliothèque.



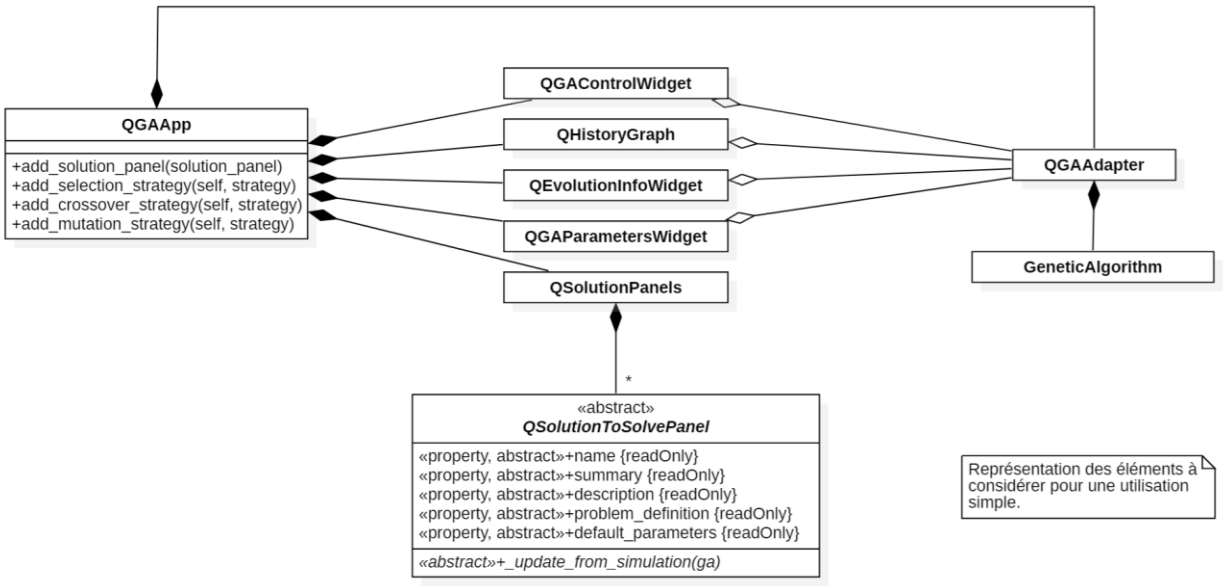
Application de base

Vous avez à votre disposition une application fonctionnelle facilitant la résolution de problème avec l'algorithme génétique proposé. Cette application consolide la gestion de plusieurs éléments et permet de traiter différents problèmes à résoudre tout en n'ayant pas à répéter les mêmes opérations récurrentes.

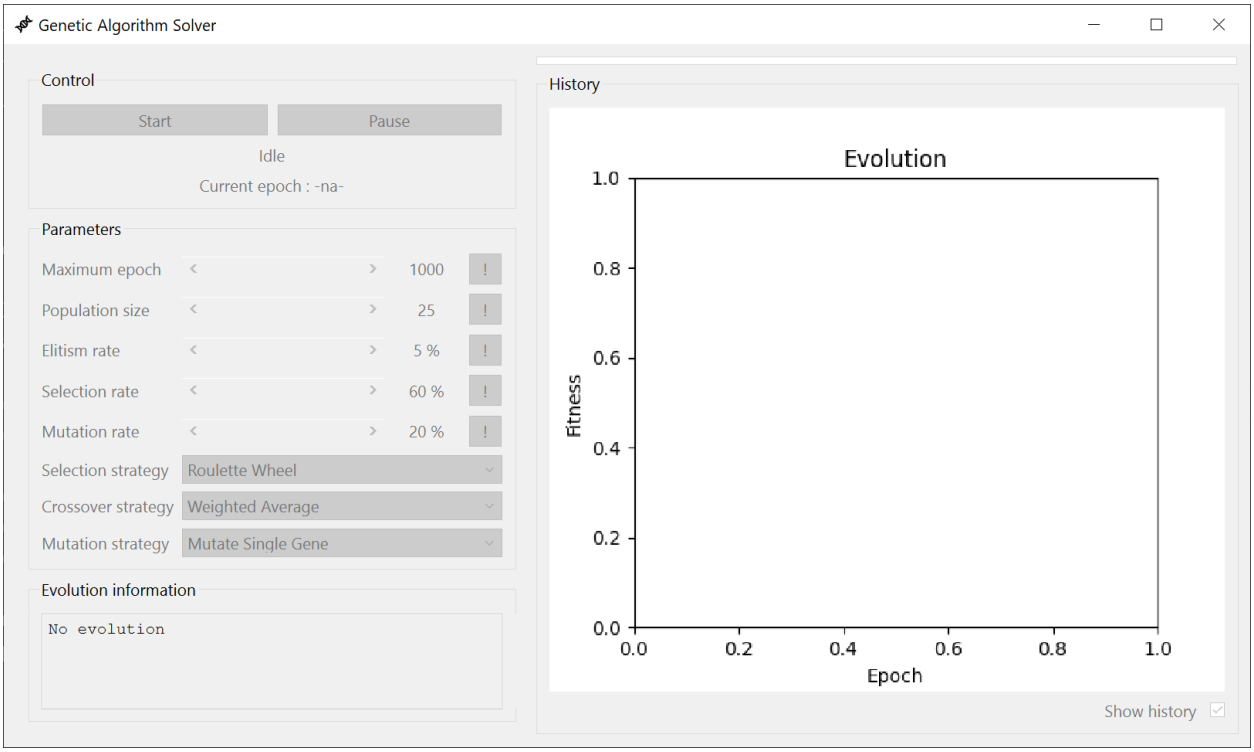
Notamment, on retrouve ces caractéristiques :

- Possède une instance de la classe `gacvm.GeneticAlgorithm`. De plus, une classe permet d'adapter le moteur générique à l'infrastructure `Qt`.
- Utilise aussi quelques patrons de conception tels qu'adaptateur et observateur (partie utilisation).
- Plusieurs sous `widgets` facilitent la gestion des éléments récurrents :
    - contrôle des simulations
    - définition des paramètres intrinsèques
    - information sur la solution courante la plus performante
    - visualisation de l'historique de performance
    - gestion de plusieurs problématiques à résoudre via des panneaux adaptés à chaque problème.
- La classe abstraite `QSolutionToSolvePanel` offre une interface de programmation uniforme pour la résolution de problème quelconque.
    - On y retrouve plusieurs éléments devant obligatoirement être définis :
        - `name` le nom qui se retrouvera sur l'onglet
        - `summary` un court résumé du problème et de l'objectif de résolution
        - `description` un texte descriptif pouvant être affiché via une boîte de dialogue
        - `problem_definition` définition du problème
        - `default_parameters` paramètres par défaut initialisant l'interface utilisateur
        - `_update_from_simulation` fonction offrant une rétroaction pendant l'évolution
    - De plus, la classe héritant de `QWidget` offre l'opportunité de créer tous les éléments pertinents à la résolution d'un problème. Ici, aucune contrainte n'est imposée toutefois il est attendu que soient offerts au moins deux ensembles d'outils :
        - une section permettant de paramétrer le problème
        - une section permettant d'afficher une rétroaction pendant l'évolution
- La classe `GAApp` encapsule tous ces éléments dans une application complète. Quatre fonctions simples permettent de personnaliser l'application :
    - `add_solution_panel` ajoute un panneau de résolution de problème
    - `add_selection_strategy` ajoute une stratégie de sélection
    - `add_crossover_strategy` ajoute une stratégie de croisement
    - `add_mutation_strategy` ajoute une stratégie de mutation
- Le fichier `gaapp.py` possède tous ces éléments (avec ces dépendances externes : `pyside6`, `matplotlib`, `gacvm` et `uqtwidgets`).

Le schéma de classe UML suivant présente les constituants importants de l'application de base.

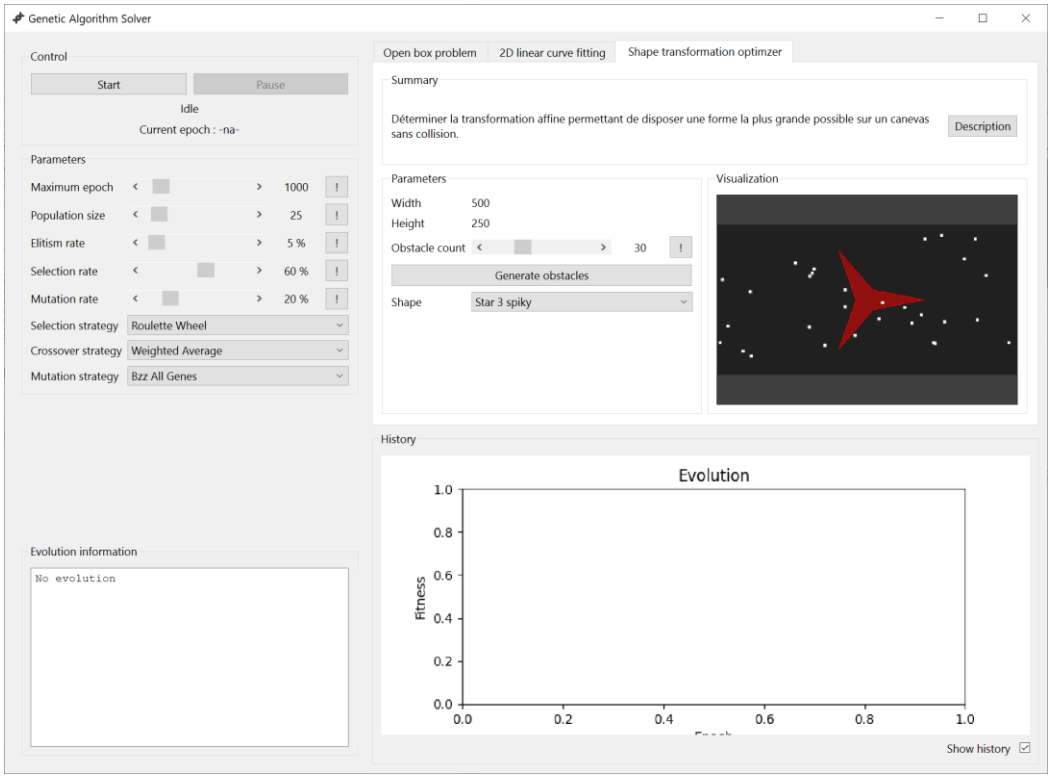


Une première capture d'écran sans panneau de résolution de problème.

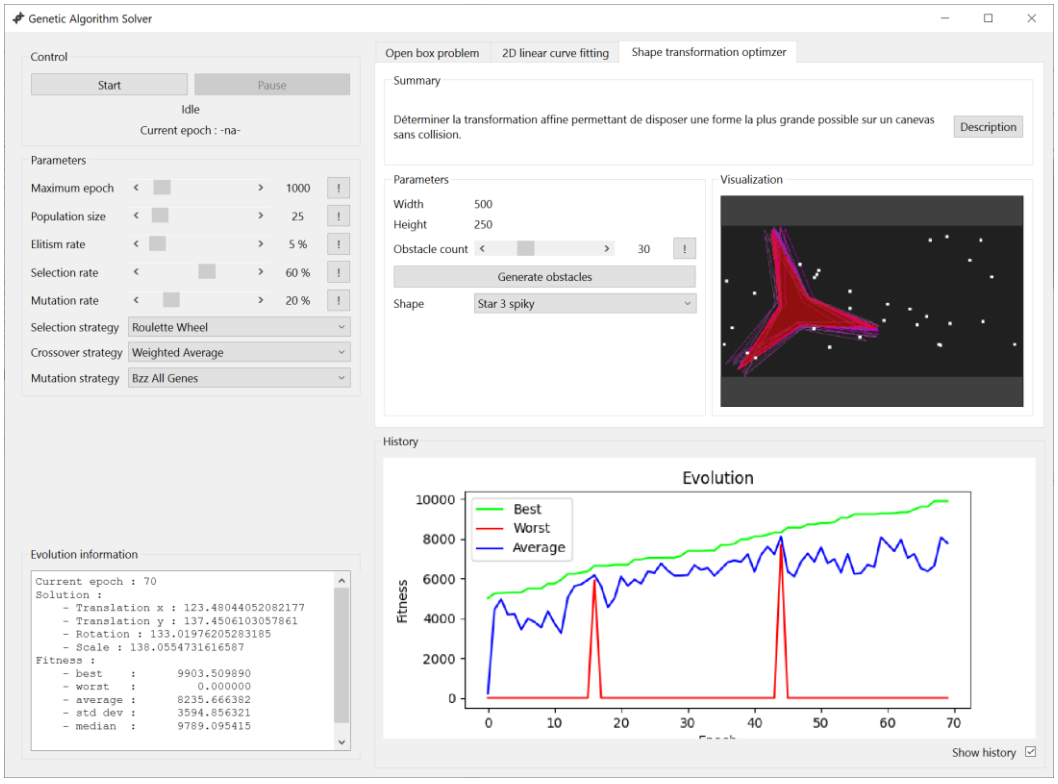


Un exemple avec plusieurs panneaux de résolution de problème et plusieurs stratégies supplémentaires ajoutées.

Avant la résolution de problème :



Pendant la résolution de problème :



Présentation des problématiques à résoudre

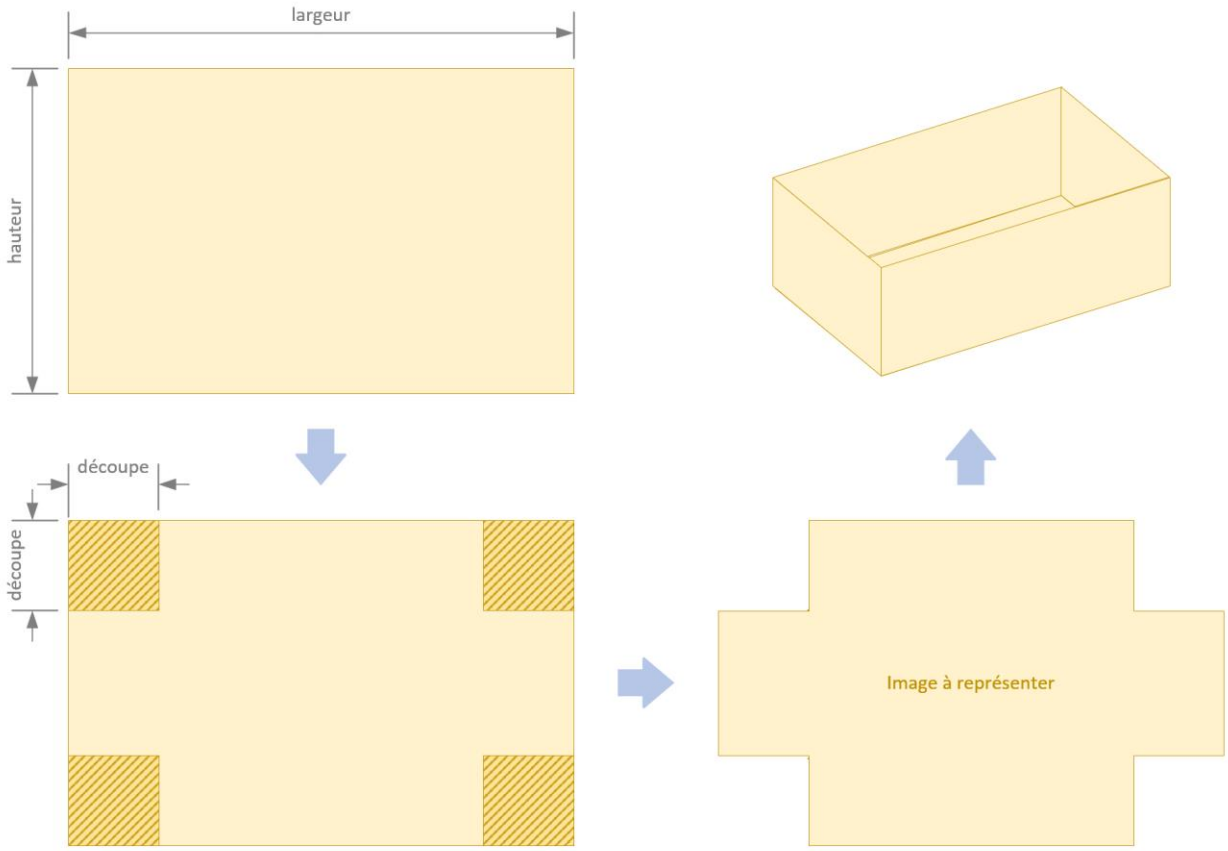
Vous avez à réaliser la solution à trois problèmes différents :

- problème de la boîte ouverte
- optimisation géométrique
- problème de votre choix

Problème de la boîte ouverte

On cherche à obtenir la plus grande boîte ouverte formée à partir d'une surface rectangulaire de taille fixe et initialement connue (la largeur et la hauteur).

L'idée consiste à déterminer la taille des carrés à découper aux quatre coins pour permettre la formation de la boîte en repliant les quatre côtés restants.

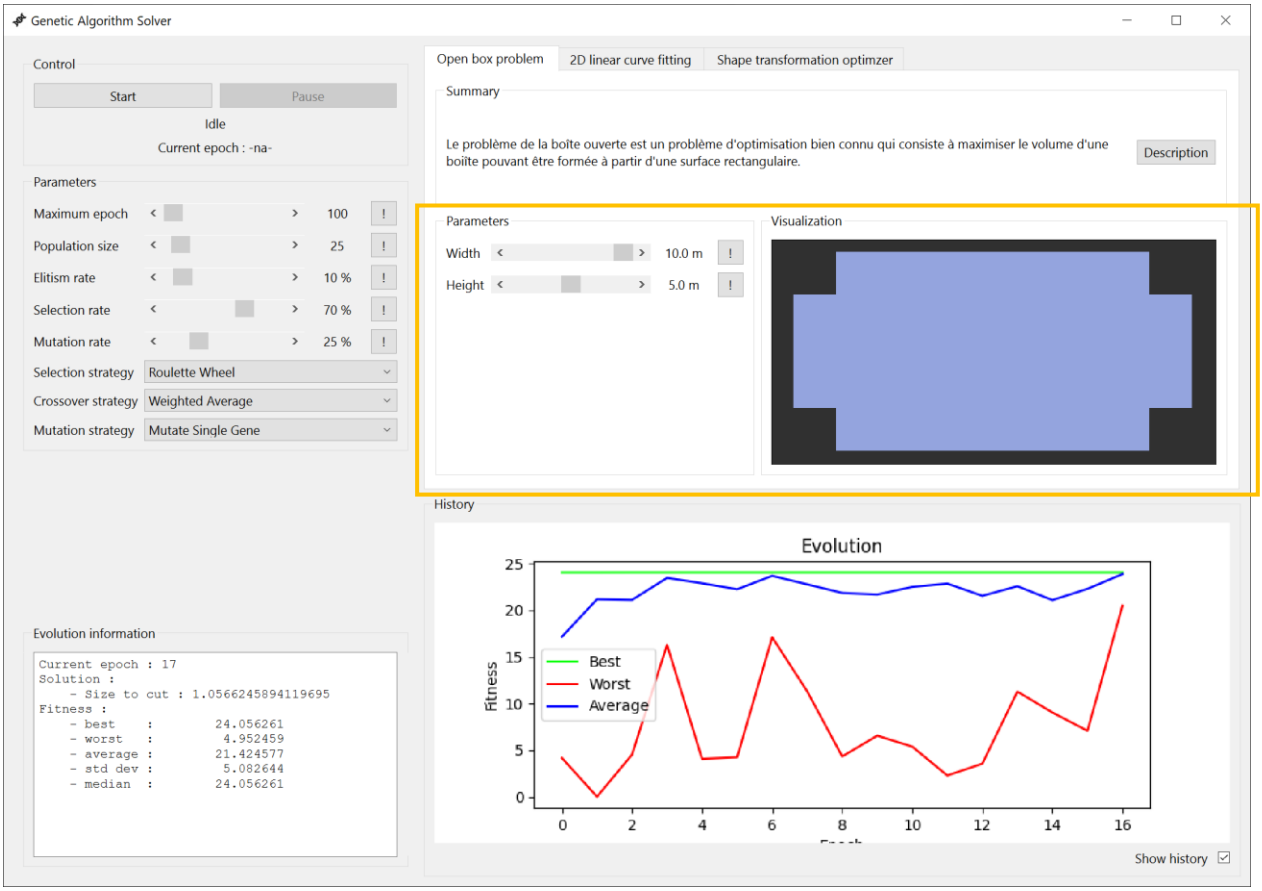


Interface utilisateur

Pour ce problème, vous devez offrir :

- la paramétrisation de la surface rectangulaire :
  - hauteur
  - largeur
- une rétroaction sous forme d'image de la forme de la surface rectangulaire coupée aux 4 coins de la meilleure solution.

La capture suivante donne un exemple possible de cette interface utilisateur (voir l'encadré jaune).



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 au début du problème
- 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.

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

Interface utilisateur

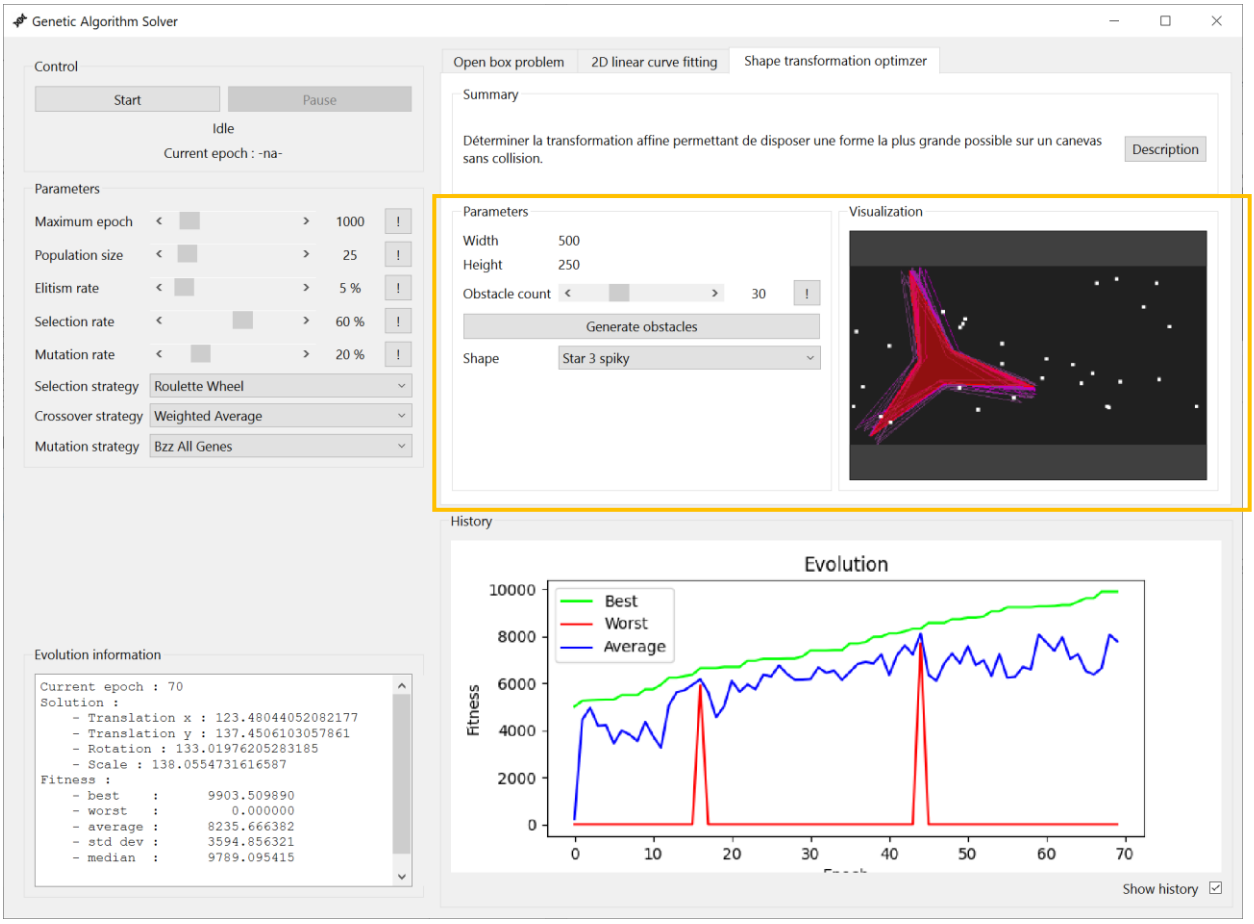
Pour chaque résolution de problème, trois paramètres fondamentaux doivent être déterminés et rester immuables tout au long de la résolution du problème :

- la taille du canevas (la paramétrisation par programmation seulement est suffisante);
- le nombre d'obstacles et la disposition de ces derniers;
- la forme géométrique (au moins 3 formes géométriques doivent être offert).

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.

De plus, on s'attend à une rétroaction où sont affichés le canevas, la liste d'obstacles, toutes les formes de la population ainsi que la meilleure forme d'une couleur évidente.

La capture suivante donne un exemple possible de cette interface utilisateur (voir l'encadré jaune).



Problème de votre choix

Vous devez produire la solution pour un problème de votre choix. Vous devez considérer les contraintes suivantes :

- plus votre problème a de dimensions, plus vous avez de points
- le problème doit être initialement paramétrable pour au moins 2 aspects
- vous devez offrir une visualisation pertinente
- vous devez être capable de converger (même si la solution n'est pas intéressante)

Attention, l'implémentation de l'algorithme génétique mise à votre disposition utilise exclusivement une représentation avec des réels. Ainsi, la nature des problèmes pouvant être résolu est limitée à ce type de représentation. Par exemple :

- les problèmes d'optimisation numérique sont d'excellents candidats pour une telle approche
- les problèmes à nature d'ordonnancement s'appliquent moins bien

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 les solutions à résoudre.
- Vous devez mettre la structure documentaire suivante :
    - Vous faites un effort particulier pour mettre en pratique l'auto documentation de votre code (noms de classes, de types, de fonctions, de variables pertinentes et surtout le respect de la norme PEP 8).
    - Tous vos fichiers *.py 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.
    - 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 avec Python en utilisant les modules suivants :
    - `pyside6`
    - `matplotlib`
    - `numpy`
    - `gacvm`[1]
    - `gaapp`[1]
    - `umath`[1]
    - `uqtgui`[1]
    - `uqtwidgets`[1]
- Vous devez créer une stratégie supplémentaire de votre choix (sélection, croisement ou mutation)
- Vous devez utiliser l'IDE Visual Studio Code.
- Ce projet doit être réalisé en équipe de 3 ou 4 étudiants.
- Vous avez jusqu'à la fin de la session pour réaliser ce projet.

Rapport

Un micro-rapport est demandé à même le logiciel. Ainsi, pour chaque problème abordé, vous devez indiquer ces éléments dans la description du panneau :

1. Qui sont les membres de l'équipe.
2. Une description du problème.
3. Des détails sur la définition du problème :
    a. Le nombre de dimension, à quoi correspondent chacun d'entre elles et quelle est le domaine déterminée.

[1] Disponibles dans les fichiers fournis.

b. Les données paramétrables, mais fixées au début de la résolution de problème (intrants).
c. La fonction de « fitness ».

Évaluation

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

1. Qualité des paramètres de résolution des problèmes abordés :
    - identification de l'espace de solution
    - définition du domaine
    - fonction objective
    - les paramètres spécifiques de l'algorithme génétique compte pour moins :
        - taille de la population
        - taille de l'élitisme
        - taux de mutation
2. Qualité et pertinence de la stratégie développée.
3. Convergence et résultats obtenus
4. Qualité de la modularité et de la réutilisabilité
5. Implémentation de l'interface usager.
6. Qualité générale du code `Python` 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 …)

Annexe 1 – 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 ou à 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 soit, 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.

Paramètres de l'algorithme

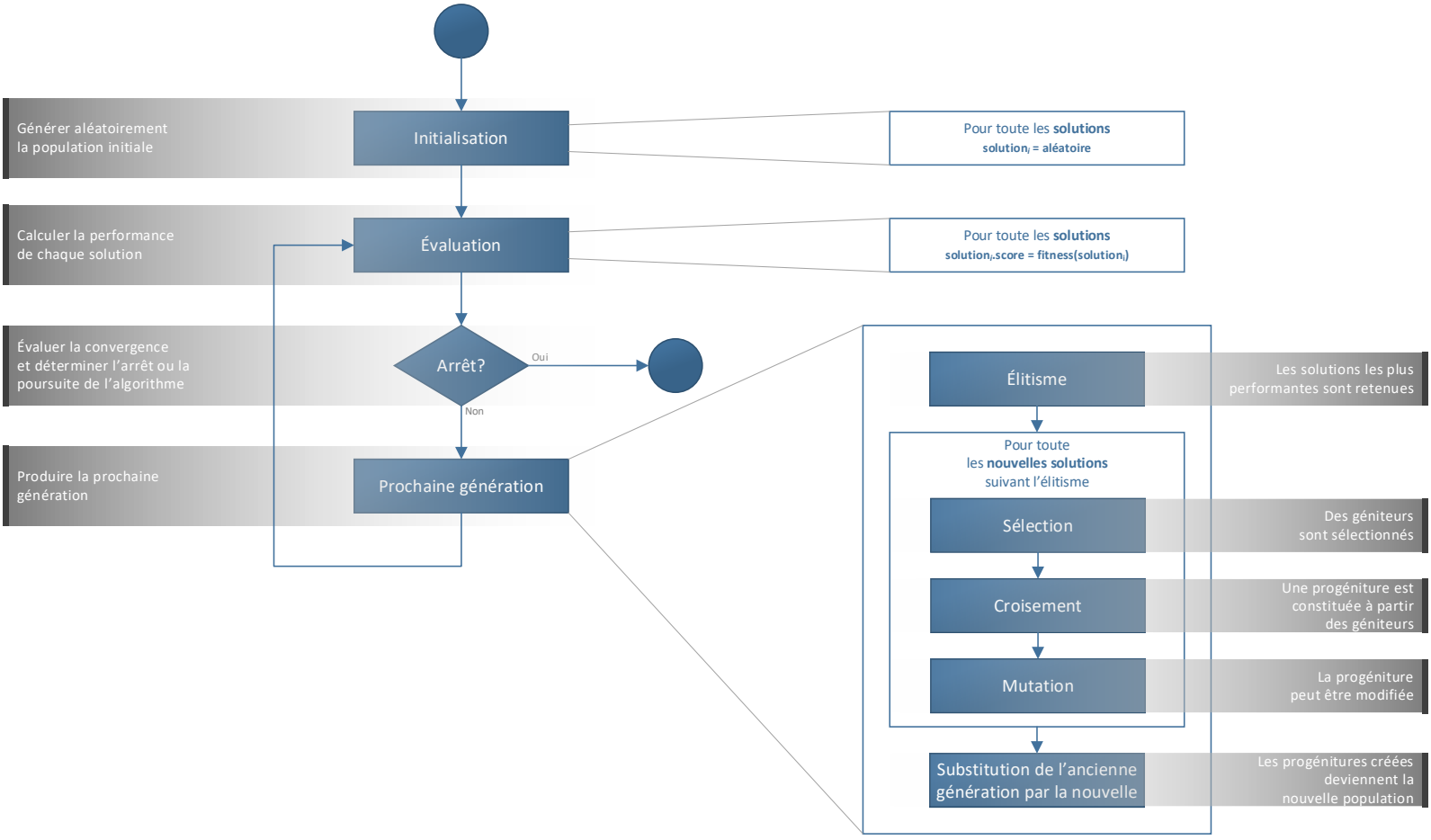
Considérant que même les processus sont ajustables, la résolution d'un problème avec l'algorithme génétique est divisée en trois grandes familles de paramètres :

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 évaluant la performance relative d'une solution
2. Les paramètres intrinsèques de l'algorithme :
    - Le nombre de générations max — détermine un critère d'arrêt correspondant au budget disponible
    - La taille de la population — détermine le nombre de solutions participant à l'évolution
    - Le taux sélection — détermine le % de la population qui pourra être sélectionné comme géniteur
    - Le taux d'élitisme — détermine le % de la population des meilleures solutions maintenues
    - Le taux de mutation — détermine la probabilité de mutation pour une nouvelle progéniture
3. Les stratégies algorithmiques (faisant partie des paramètres intrinsèques) :
    - 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.

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.



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 :

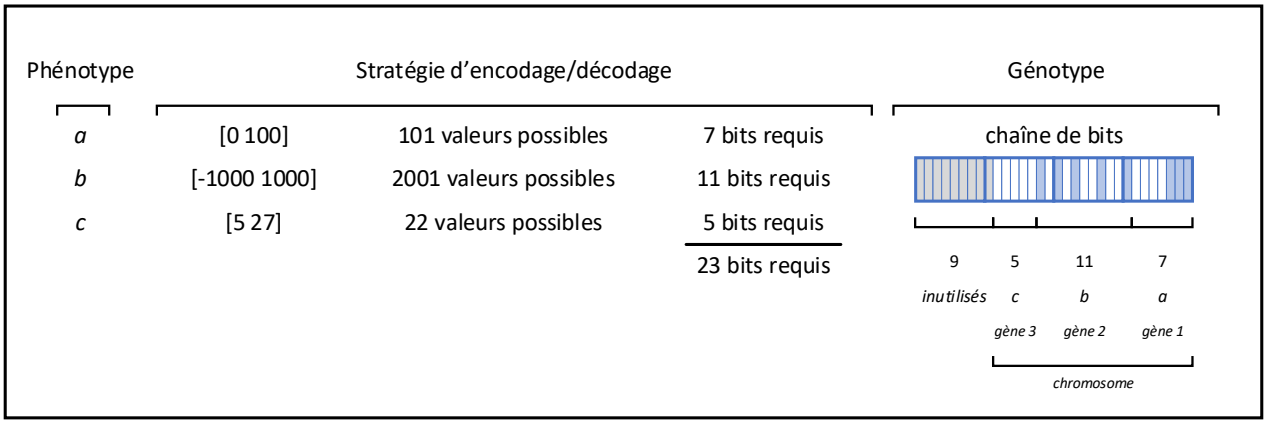
- *a* : un nombre entier inclue dans l'intervalle [0 100]
- *b* : un nombre entier inclue dans l'intervalle [-1000 1000]
- *c* : un nombre entier inclue dans l'intervalle [5 27]

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

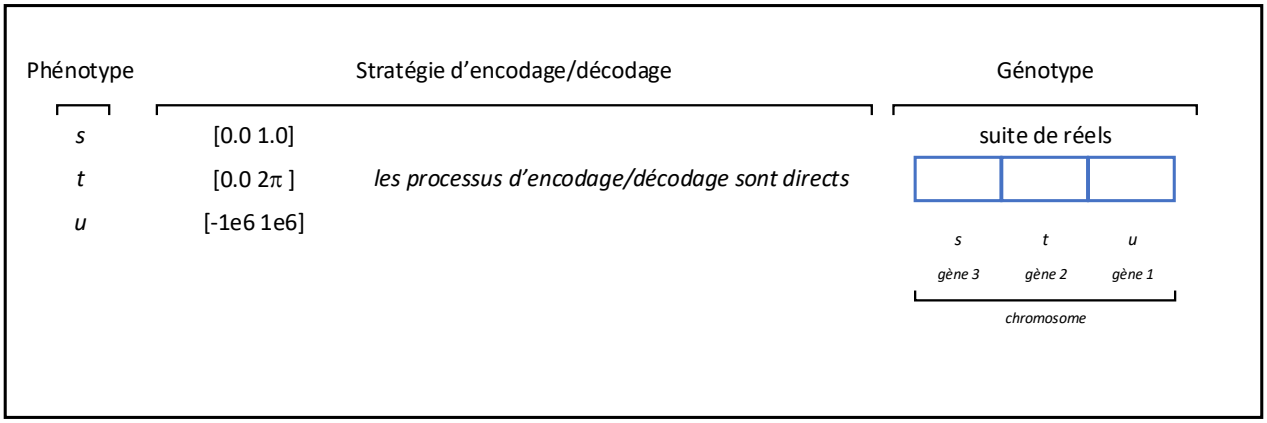
- *s* : un nombre réel inclue dans l'intervalle [0.0 1.0]
- *t* : un nombre réel inclue dans l'intervalle [0.0 2π]
- *u* : un nombre réel inclue dans l'intervalle [-1 000 000.0 1 000 000.0]

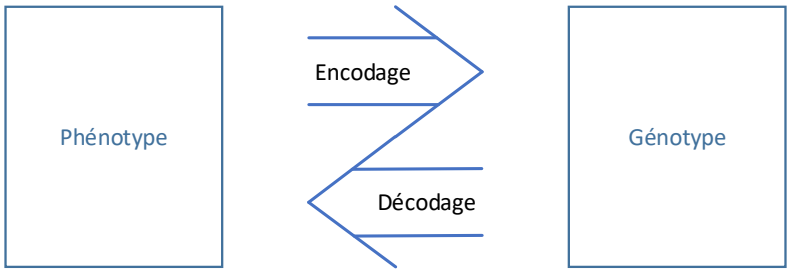
Représentation, encodage et décodage

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).

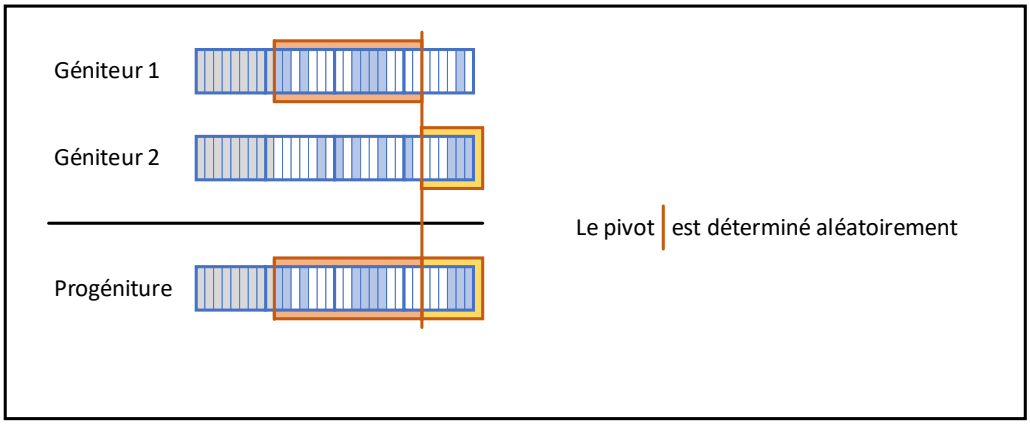
Sélection

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

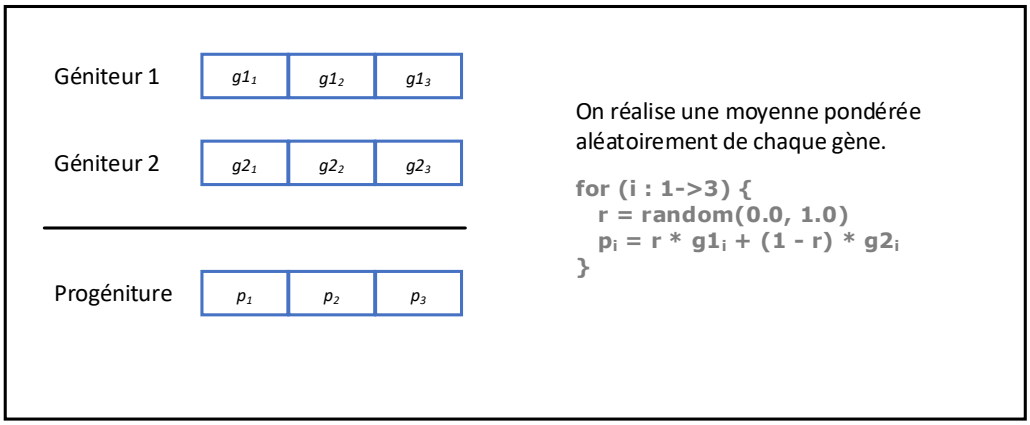
Il n'existe pas de méthode officiellement fondamentale. Toutefois, la méthode « *Roulette Wheel* » est généralement utilisée comme approche générique. Cette approche utilise le poids relatif de chaque individu pour déterminer leur probabilité de sélection. Le choix des géniteurs se fait ensuite par une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

Annexe 2 – Quelques outils informatiques

gacvm

Classes du moteur de l'algorithme génétique.

gaapp

Classes de l'application supportant la résolution de problème par l'algorithme génétique.

umath

Fonctions mathématiques utilitaires :

- `clamp`

uqtwidgets

Fonctions et classes utilitaires reliées aux widgets de Qt :

- fonction `create_scroll_int_value`
- fonction `create_scroll_real_value`
- classe `QSimpleImage`

uqtgui

Fonctions utilitaires pour les éléments de gui de Qt :

- éléments de géométrie :
  - `perimeter_from_QRectF`
  - `area_from_QRectF`
  - `perimeter_from_QPolygonF`
  - `area_from_QPolygonF`

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 et QPolygonF

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. Vous trouverez ces outils dans `uqtgui`.

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(brush)` : détermine tous les paramètres de remplissage tels que la couleur. Voir la classe `QBrush`.
- `QPainter.setPen(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.

Finalement, contrairement au langage `C++`, il est essentiel d'appeler la fonction `QPainter.end()` lorsqu'on a terminé de dessiner et surtout avant de terminer le contexte local. Ceci permet de libérer les ressources adéquatement.

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` déterminent les transformations à appliquer. Les fonctions `QTransform.map` 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.