Projet 2 | Algorithme génétique

420-C52 | Données, mégadonnées et intelligence artificielle I

Table des matières

Introduction

Ce projet consiste à réaliser une application informatique capable de résoudre des problèmes d'optimisation différents par l'utilisation de l'algorithme génétique.

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

De plus, l'étudiant doit pouvoir faire la distinction entre les éléments génériques et spécifiques de cette application afin de cibler des développements adaptés et modulaires pertinents. Un effort particulier est mis sur la pratique du paradigme de programmation orientée objet où le polymorphisme constitue les architectures fondamentales du projet.

Objectifs généraux

Les objectifs principaux de ce projet sont :

- utiliser l'algorithme génétique pour résoudre deux problèmes d'optimisation;
- étendre les fonctionnalités d'une application informatique en supportant la résolution des problèmes adressés.

Objectifs spécifiques

Plus spécifiquement, ce projet vise à :

- bien comprendre la nature des problèmes abordés afin de déterminer adéquatement les paramètres liés à la 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 sommaire du projet

Ce projet consiste à produire une application résolvant 4 problèmes différents à l'aide de l'algorithme génétique. Une bonne partie du projet existe déjà et on vous demande de poursuivre le développement pour les parties manquantes. Ainsi, vous avez à produire :

- la solution du problème 3 qui est un problème imposé d'optimisation géométrique
- la solution du problème 4 qui est un problème à sujet ouvert
- une stratégie basée sur l'une des opérations fondamentales de l'algorithme génétique.

Contrairement au laboratoire précédent, vous avez à votre disposition :

- `gamain.py` : le fichier où se trouve les constituants de démarrage de l'application. Vous y retrouverez la possibilité de configurer l'application en lui ajoutant des problèmes variés et des stratégies supplémentaires.
- `gacvm.py` : le fichier où se trouve une implémentation de l'algorithme génétique basée principalement sur la classe `GeneticAlgorithm`.
- `gaapp.py` : le fichier où se trouve l'application graphique `QGAApp` permettant la résolution de problèmes variés par l'utilisation de `gacvm`.
- L'application possède déjà, à titre d'exemple complet, deux problèmes déjà solutionnés et une stratégie disponible :
  - `ga_problem_unknown_number.py` : le fichier où se trouve la résolution du problème où on recherche un nombre déterminé par l'utilisateur, mais inconnu par l'algorithme génétique, certainement l'un des problèmes les plus simples.
  - `ga_problem_open_box.py` : le fichier où se trouve la résolution du problème de la boîte ouverte, le « *hello world* » des problèmes d'optimisation.
  - `ga_strategy_genes_mutation.py` : le fichier où se trouve l'algorithme de mutation de plusieurs gènes à la fois.
- Plusieurs autres utilitaires sont disponibles dans divers autres fichiers : `umath.py`, `uqtgui.py` et `uqtwidgets.py`.

Voici les étapes suggérées de développement de ce projet :

1. Formation des équipes (3 ou 4).
2. Lectures individuelles et rapides :
   a. de l'énoncé
   b. du code existant. [1]
3. Faites une lecture plus approfondie, en équipe, des deux solutions données à titre d'exemple. Assurez-vous que tous les membres de l'équipe comprennent ces deux exemples. Surtout, posez des questions à l'enseignant rapidement. Tentez de respecter ces étapes :

[1] Attention, on vous suggère de faire une première lecture rapide du code pour avoir une idée de la métastructure que vous avez entre les mains. Suivez les instructions de l'enseignant pour faciliter cette phase du projet. Vous devez absolument éviter de tout lire. Pour le temps que vous avez, il est impossible de tout comprendre cette infrastructure. Dans le cadre du cours, c'est la vue d'ensemble de qui est intéressante en mettant l'emphase sur la compréhension de la philosophie des outils mis à votre disposition et des points d'entrée de l'architecture imposée. Après la session, si vous désirez approfondir vos connaissances, une lecture plus approfondie peut être pertinente et très formatrice.

a. Bien comprendre le problème.
b. Bien comprendre l'architecture logicielle de résolution du problème via le mécanisme d'héritage de la classe `QSolutionToSolvePanel` et du polymorphisme en place.
c. Où se trouvent les éléments de paramétrisation du problème.
d. La définition adéquate des trois chaînes de caractères demandées. Voir les propriétés en lectures seules `name`, `summary` et `description`.
e. Identifier et comprendre la définition du problème qui inclue la définition du domaine (dimensionnalité et intervalles) et la fonction objective (fonction de *fitness*). Voir la méthode `problem_definition`.
f. Identifier et comprendre la définition des paramètres initiaux de l'engin de résolution. Voir la fonction `default_parameters`.
g. Comprendre plus largement le panneau `Qt` :
    i. d'abord pour l'assemblage des *widgets*
    ii. ensuite pour la configuration de la paramétrisation du problème
    iii. finalement, pour le mécanisme de rétroaction via la visualisation de l'évolution (voir la méthode `_update_from_simulation`).

4. Faites une réflexion en équipe de la définition de la solution du problème 3. Assurez-vous de définir en équipe les éléments suivants :
    a. Le domaine (dimensionnalité et intervalles).
    b. La fonction objective (la fonction de *fitness*).
    c. Quels sont les paramètres d'entrée qui seront configurable par l'utilisateur via le panneau.
    d. Quelle est la rétroaction visuelle à produire.
5. Identifier un problème que vous voulez résoudre.
    a. Gardez en tête que l'implémentation de l'algorithme génétique que vous avez s'appuie sur la manipulation de nombres réels et que, dès lors, certains types de problème sont mieux adapté que d'autres.
    b. Choisissez un problème réaliste dans le temps que vous avez.
    c. Valider votre problème avec l'enseignant.
    d. Applique l'étape 4 pour ce problème.
6. Débuter la programmation de vos panneaux de résolution en équipe.
    a. Un à la fois.
    b. En mettant en pratique les bonnes pratiques encouragées dans le cadre du cours : `CALTAL`, `DRY` et `UMUD`.
7. Produire au moins une stratégie générique, mais pertinente pour vos problèmes.
8. Produisez la documentation finale de votre projet. Les chaînes de caractères `summary` et `description` sont cette documentation.

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.

Dans les deux cas, un fort biais est mis sur le paradigme de programmation orientée objet afin de favoriser une meilleure compréhension et une mise en pratique du concept.

Bibliothèque de l'algorithme génétique

Les caractéristiques principales de cette bibliothèque sont :

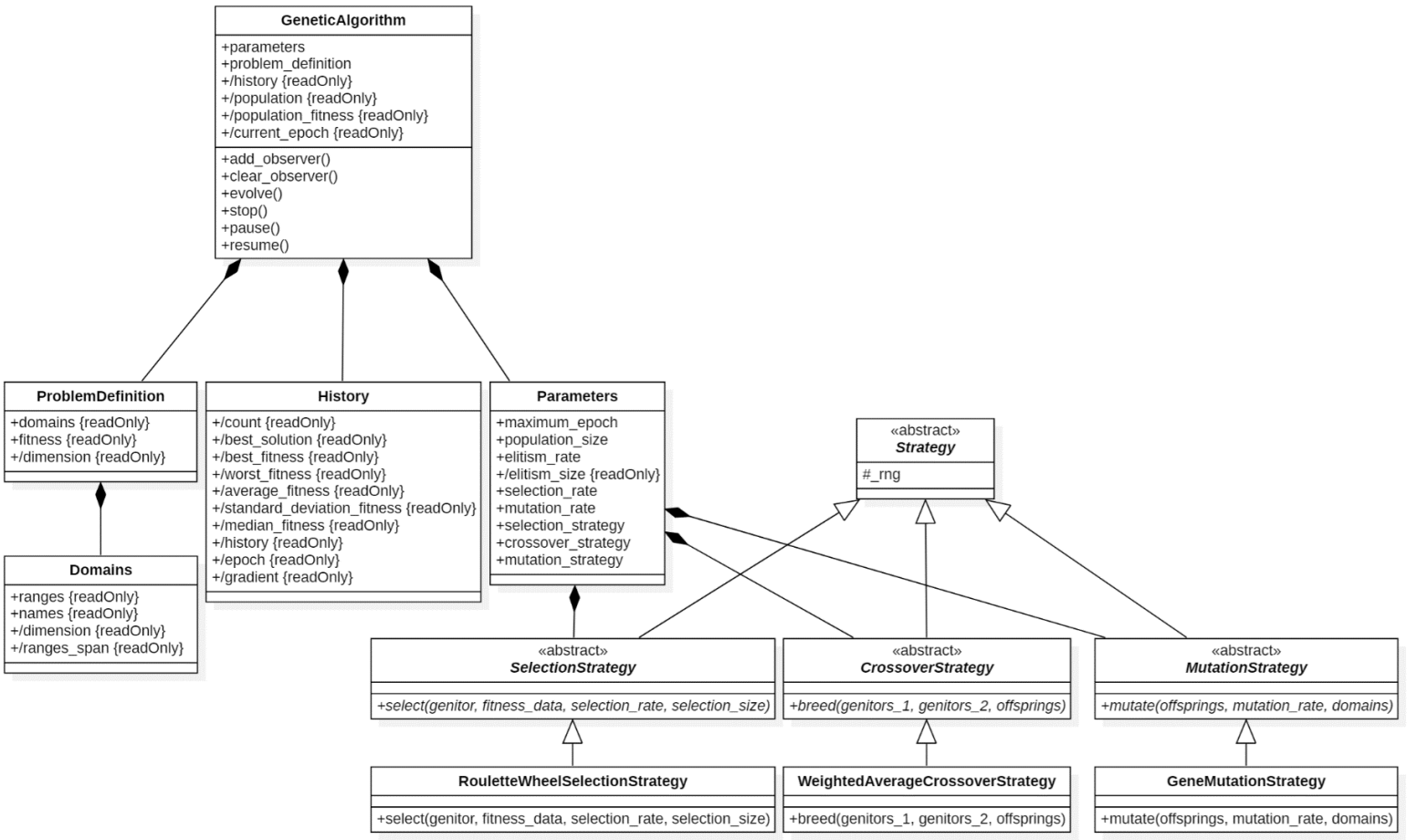
- L'algorithme génétique est basé sur une représentation par des nombres réels.
- La représentation par nombres réels implique que les stratégies d'encodage et de décodage sont inutiles, car les génotypes et phénotypes sont confondus.
- L'implémentation est entièrement basée sur la bibliothèque `NumPy`.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

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

- La classe `History` permet un accès simplifié aux résultats lors de l'évolution.
- 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)

- Il est important de noter que cette classe ne possède aucune autre dépendance. Ceci inclut `Qt`. Ainsi, l'algorithme génétique mis à votre disposition peut être utilisé dans n'importe quels autres projets `Python`.

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



Utilisation de la bibliothèque gacvm

L'utilisation la plus simple de la bibliothèque consiste en trois étapes principales :

1. créer une instance de la classe `ProblemDefinition` et à la paramétrer afin de définir le problème à résoudre.
2. créer une instance de la classe `GeneticAlgorithm` et à la configurer par l'instance créée en 1 et, optionnellement, par une instance de la classe `Parameters`.
3. démarrer l'évolution par la méthode `evolve` de la classe `GeneticAlgorithm`.

Vous trouverez un exemple détaillé dans la fonction `main` du fichier `gacvm.py`.

Application graphique

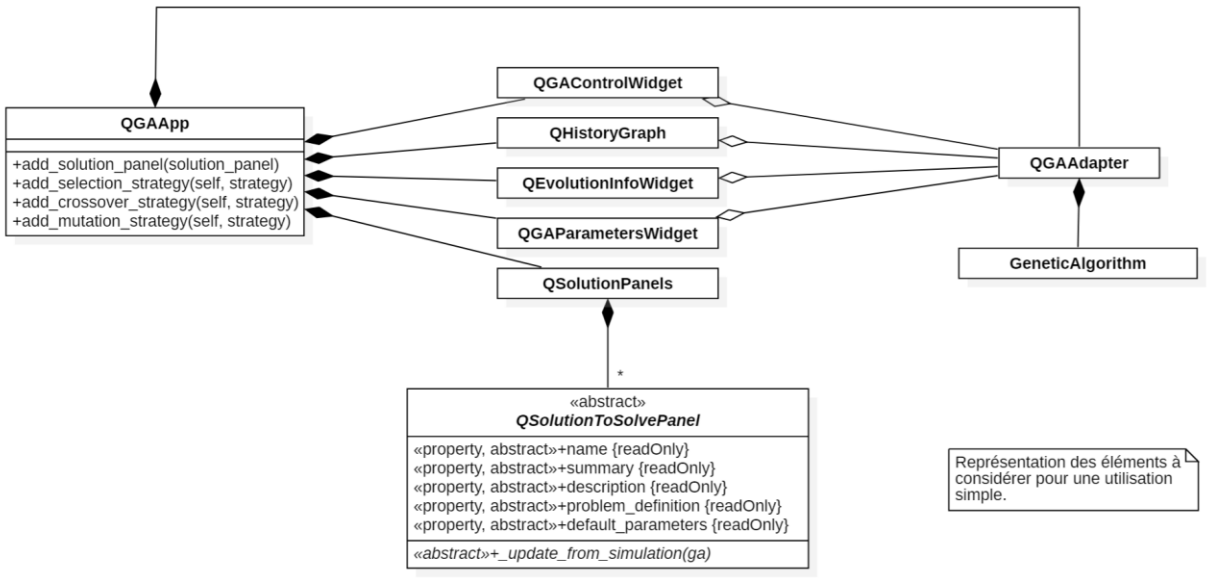
Vous avez à votre disposition une application entièrement fonctionnelle facilitant la résolution de problème avec l'algorithme génétique proposé. Cette application consolide la gestion de plusieurs

éléments et permet de traiter divers problèmes à résoudre tout en évitant la répétition des opérations de gestion récurrentes.

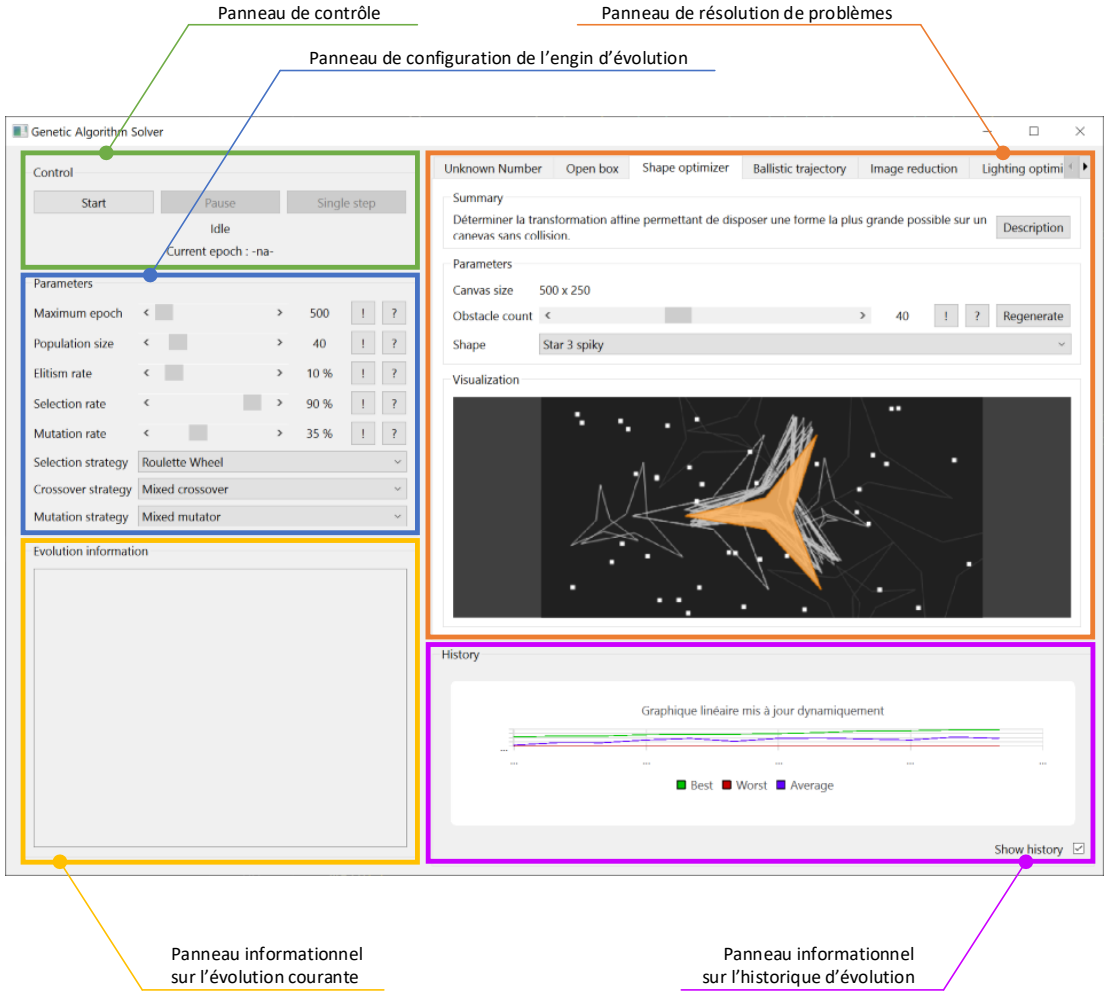
Notamment, on retrouve ces caractéristiques :

- Possède une instance de la classe `gacvm.GeneticAlgorithm` et la gère entièrement. Cette gestion se fait de façon relativement transparente via la classe `QGAAdapter`, permettant l'adaptation du 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`, `gacvm` et `uqtwidgets`).

Diagramme de classes de l'application



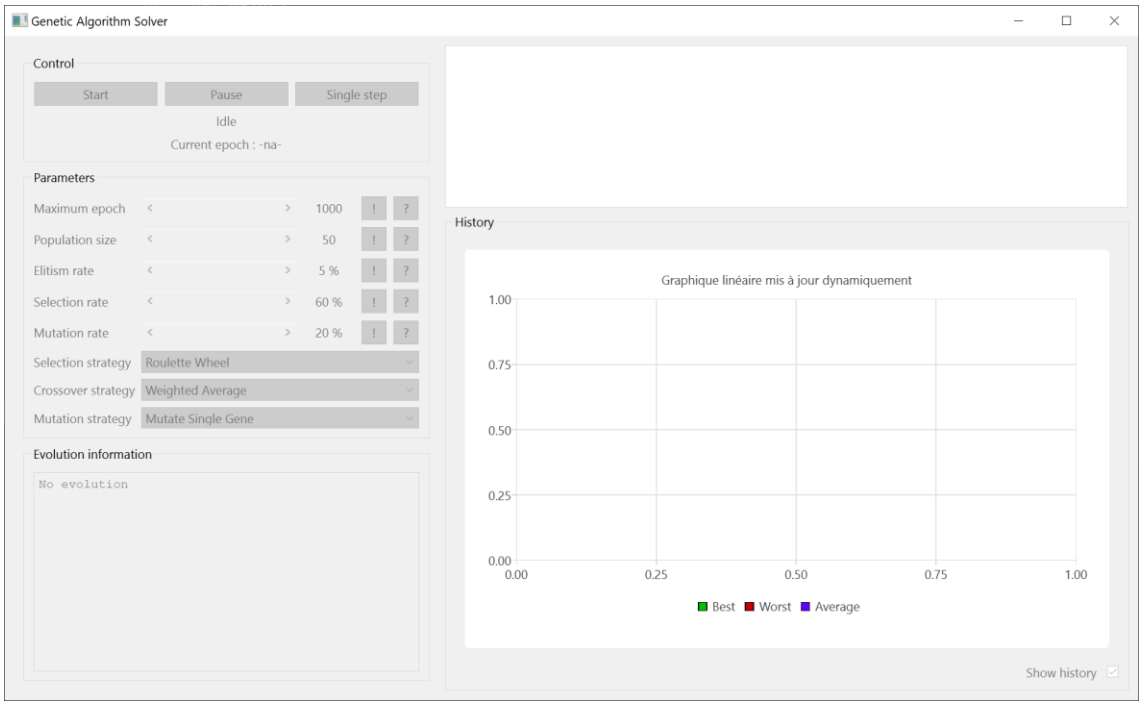
Captures d'écran présentant les panneaux principaux de l'application



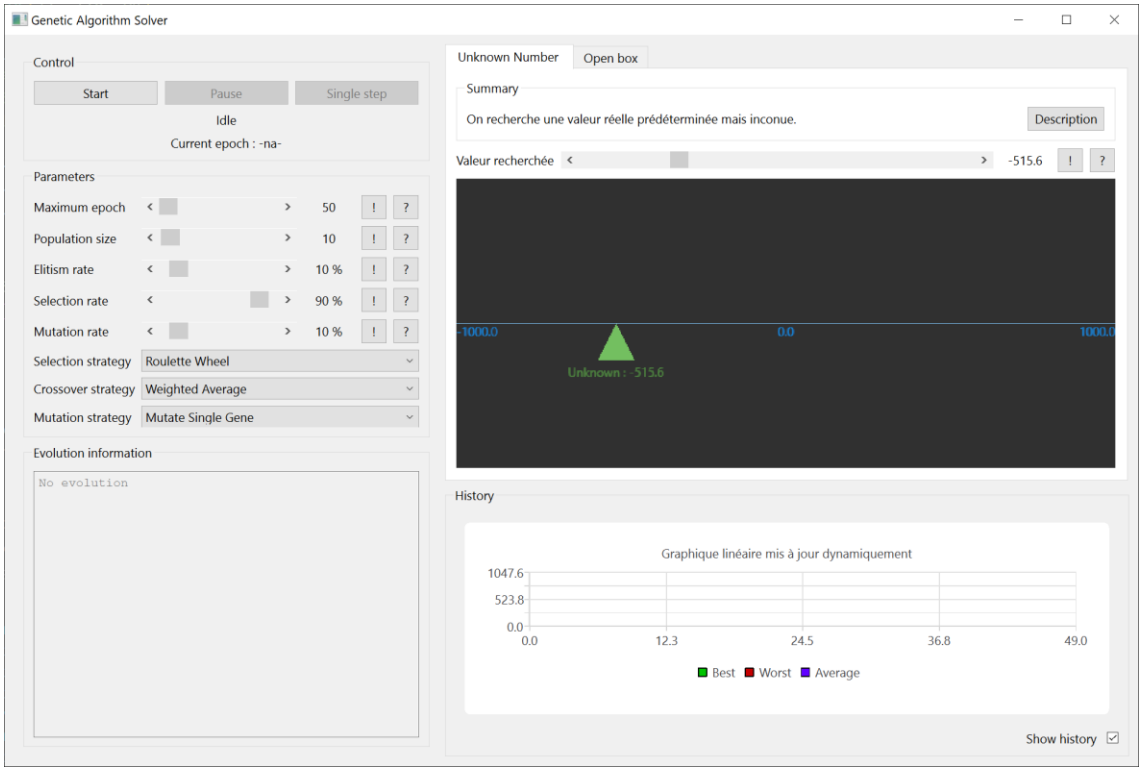
On retrouve les 5 panneaux principaux suivants :

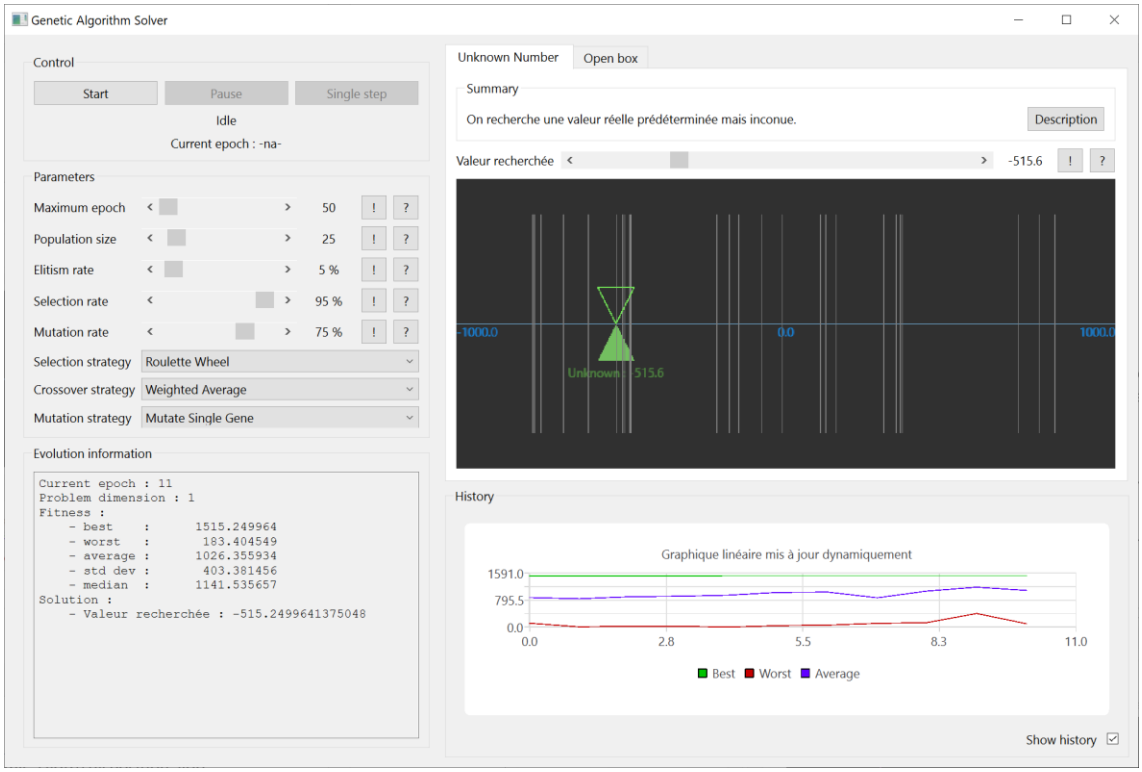
- Le panneau de contrôle : permet le contrôle de la simulation.
- Le panneau de configuration de l'engin d'évolution : permet la configuration des paramètres génériques de l'algorithme génétique.
- Le panneau informationnel sur l'évolution courante : permet l'affichage d'information sur la meilleure solution pour chaque pas d'évolution.
- Le panneau informationnel sur l'historique d'évolution : permet de voir comment l'algorithme évolue en termes de performance au fil de l'évolution.
- Le panneau de résolution de problèmes : permet de choisir un problème à résoudre et d'en faire la configuration et la résolution. Ce panneau est vide par défaut, il est essentiel d'ajouter des problèmes à résoudre.

Cette capture d'écran présente l'application sans panneau de résolution de problème.



Les deux captures d'écran suivantes présentent l'application avec ses deux panneaux de résolution donnés initialement. La première capture présente l'application avant le démarrage de la simulation pendant la configuration du problème alors que la suivante montre l'application pendant l'évolution.





Présentation des problématiques à résoudre

L'application doit posséder 4 panneaux de résolution de problème. Deux sont mis à votre disposition alors que vous devez produire les deux derniers.

Les deux problèmes déjà résouts sont disponibles à titre de référence pour vous aider à mieux comprendre comment développer les panneaux de résolution. Ces panneaux visent la résolution de ces problèmes :

- nombre inconnu
- boîte ouverte

Deux autres panneaux de résolution sont à créer par vous et correspondent à ces défis :

- optimisation géométrique
- problème de votre choix

Les sections suivantes présentent les problèmes existants et ceux à résoudre.

Problème du nombre inconnu

Présentation

Le problème du nombre consiste à identifier un nombre connu de l'utilisateur, mais inconnu par l'algorithme génétique.

Paramétrisation

La paramétrisation du problème consiste à déterminer le nombre inconnu.

Rétroaction

La visualisation permet de voir les nombres :

- le nombre inconnu
- toutes les solutions courantes d'une couleur spéciale
- la meilleure solution d'une couleur identifiable

Domaine

Puisqu'on recherche un nombre réel inconnu, le problème est à une dimension. L'intervalle est donné initialement dans le problème et consiste en l'espace de recherche.

$$D = 1$$

$$d_1 = [-1000, 1000]$$

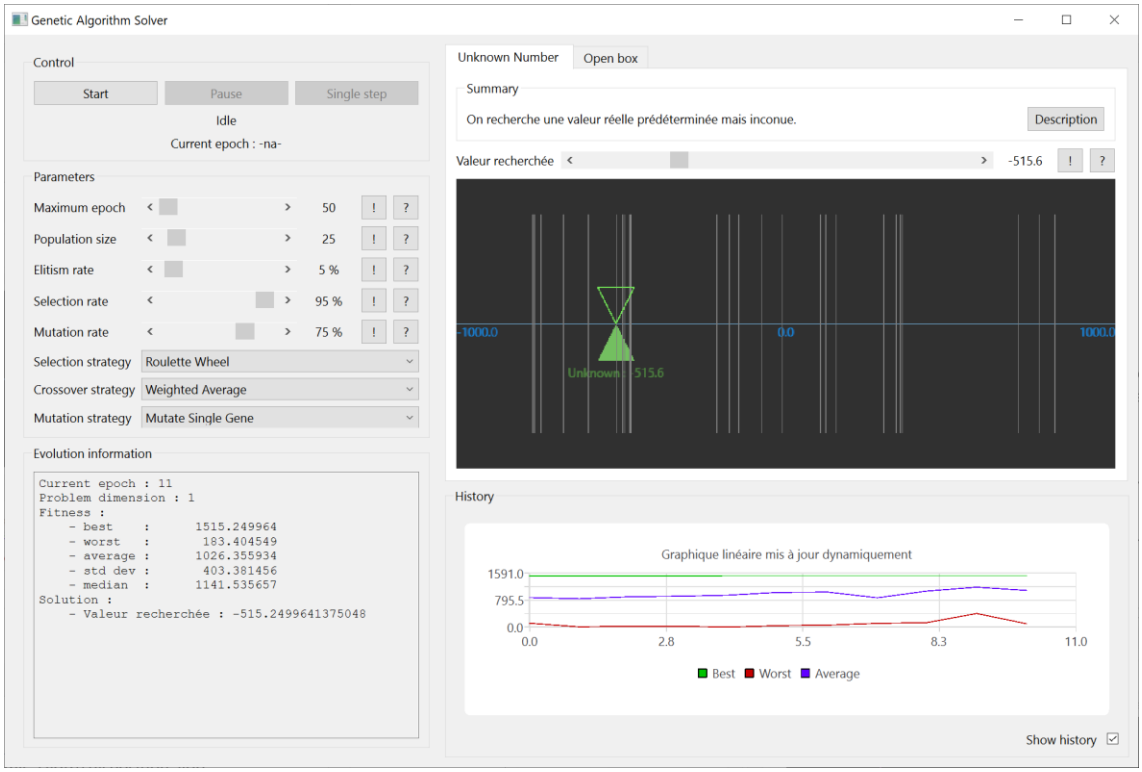
Où *D* représente le nombre de dimensions et d_1 représente l'intervalle de la première dimension. Pour cet exemple, l'intervalle est arbitraire et donné à même le constructeur du panneau de résolution.

Fonction objective

La fonction objective consiste à déterminer à quel point une solution performe bien. Dans ce cas-ci, on considère la distance 1d entre la solution courante et la valeur recherchée.

Puisque cette valeur oriente la résolution vers une minimisation, indiquant que la plus petite valeur est la meilleure, on inverse la réponse en fonction de la plage maximum des données. Plusieurs approches sont possibles pour transformer une minimisation en maximisation.

Capture d'écran

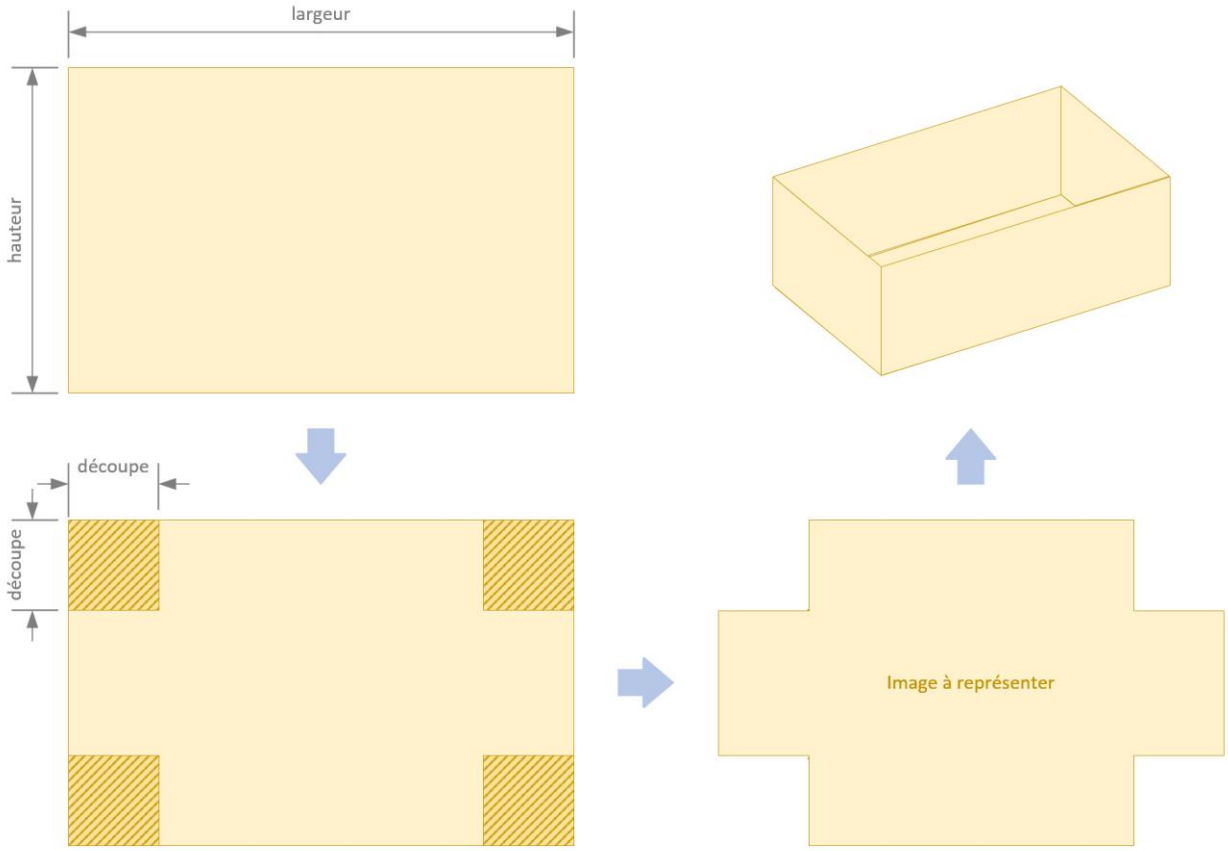


Problème de la boîte ouverte

Présentation

Le problème de la boîte ouverte consiste à trouver la taille des découpes carrées à effectuer dans les coins d'une feuille rectangulaire, de manière à former une boîte ouverte avec le plus grand volume possible.

On commence avec une feuille rectangulaire de dimensions fixes (largeur et longueur). L'idée est de découper des carrés de côté égal aux quatre coins, puis de plier les bords restants pour former les parois de la boîte. L'objectif est de déterminer la taille optimale de ces découpes pour maximiser le volume de la boîte obtenue.



Paramétrisation

L'utilisateur peut définir la taille de la feuille rectangulaire, autant la largeur que la hauteur.

Rétroaction

La rétroaction consiste à afficher la feuille rectangulaire découpée issue de la découpe donnant le meilleur volume de la boîte.

Domaine

Ce problème consiste à trouver la taille de la découpe, qui est un seul nombre réel.

$$D = 1$$

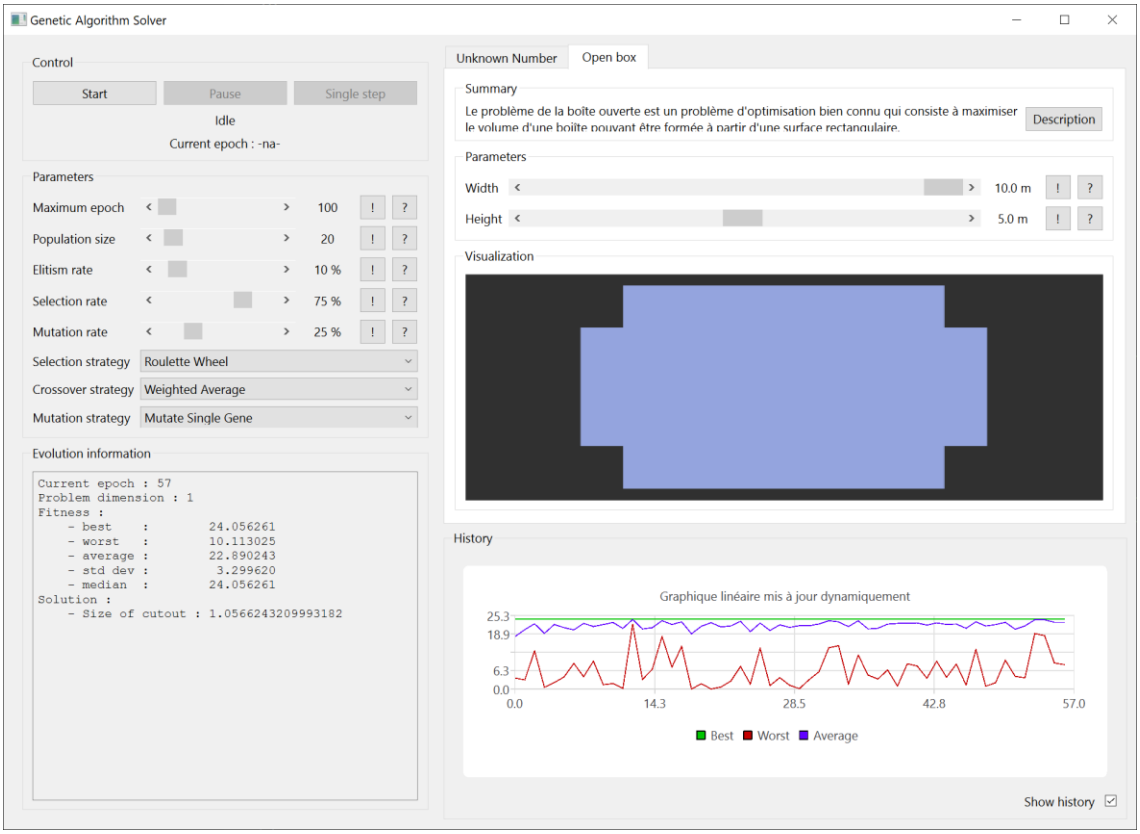
$$d_1 = \left]0, \frac{\min\left(largeur, hauteur\right)}{2}\right[$$

Où *D* représente le nombre de dimensions et d_1 représente l'intervalle de la première dimension.

Fonction objective

La fonction objective consiste à évaluer le volume de la boîte issu de la découpe liée à la solution courante.

Capture d'écran



Problème d'optimisation géométrique

Présentation

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 problème 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.

Paramétrisation

Trois paramètres fondamentaux déterminent le problème et restent 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 offertes).

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.

Rétroaction

Le panneau de résolution doit offrir une rétroaction graphique très explicite de l'évolution. Cet affichage contient :

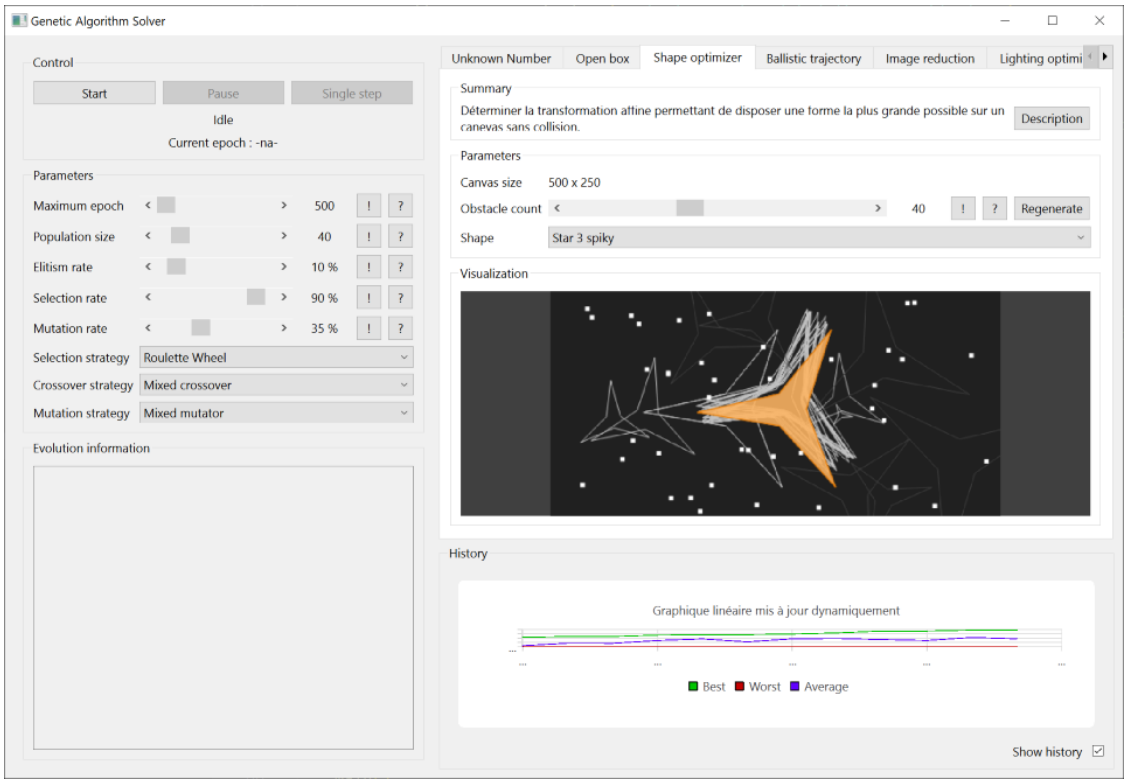
- le canevas
- tous les obstacles
- toutes les formes de la population affichée avec un contour seulement (pas de remplissage)
- la meilleure forme avec une couleur de remplissage évidente

Domaine et fonction objective

C'est à vous d'identifier ces deux éléments.

Capture d'écran

La capture suivante donne un exemple possible de cette interface utilisateur.



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 pertinentes, plus vous avez de points
- le problème doit être 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 optimale)

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 résolue 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

Présentation de la stratégie à produire

Vous devez produire au moins une nouvelle stratégie. Cette stratégie peut être de n'importe quelle nature parmi celles proposées :

- `SelectionStrategy`
- `CrossoverStrategy`
- `MutationStrategy`

Une considération à avoir : plusieurs stratégies de mutation sont parmi les plus rentables à réaliser. D'abord parce qu'elles sont relativement simples et ensuite pour l'impact sur les résultats de convergences. Certaines permettent une exploration de l'espace très intéressante et aident à sortir des minimums locaux.

Vous pouvez vous inspirer des 4 exemples donnés :

- `RouletteWheelSelectionStrategy`
- `WeightedAverageCrossoverStrategy`
- `GeneMutationStrategy`
- `GenesMutationStrategy`

Sachez que vous serez évalué sur la pertinence de votre stratégie et de sa polyvalence. Voir la documentation demandée dans la section *Rapport* plus bas.

Il est évidemment possible de faire plusieurs stratégies afin de peaufiner votre projet. Certaines de ces stratégies peuvent être spécifiques à un projet aussi. Comme on le fait dans un contexte réel en entreprise.

Contraintes liées au développement logiciel

Vous devez porter une attention particulière à cette partie, car elle constitue les lignes guides pour 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 fournissez un effort particulier pour mettre en pratique l'autodocumentation de votre code (noms de classes, de types, de fonctions, de variables pertinentes…
    - Le respect de la norme PEP 8 est un critère important.
    - Tous vos fichiers *.py présente un en-tête incluant ces informations :
        - Que contient le fichier : une ligne 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`
    - `numpy`
    - `gacvm`[2]
    - `gaapp`[2]
    - `umath`[2]
    - `uqtgui`[2]
    - `uqtwidgets`[2]
- 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.

[2] Disponibles dans les fichiers fournis.

Rapport

Aucun rapport explicite n'est demandé. Toutefois, les trois chaînes de caractères requises par les panneaux de résolution de problème constituent un mini-rapport.

Pour les problèmes 3 et 4, assurez-vous d'avoir :

- title : un titre court, mais pertinent
- summary : une présentation sommaire du problème
- description : une description plus détaillée du problème présentant :
    - une présentation plus détaillée du problème présentant clairement quel est le résultat attendu
    - les intrants configurant du problème
    - le domaine (nombre de dimensions et intervalles pour chaque dimension)
    - la fonction objective (de *fitness*) clairement exprimée
    - la rétroaction proposée
    - tout autre élément pertinent

De plus, pour votre stratégie, vous devez mettre, sous forme de `docstring` ou de commentaires à même la déclaration de la classe, les informations suivantes :

- Quel est l'objectif de cette stratégie.
- Expliquez pourquoi vous l'avez choisi et en quoi elle est générique sur un problème inconnu à *n* dimensions.
- En prenant les problèmes 3 et 4, expliquez dans quel(s) contexte(s) cette stratégie pourrait être pertinente.

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 :

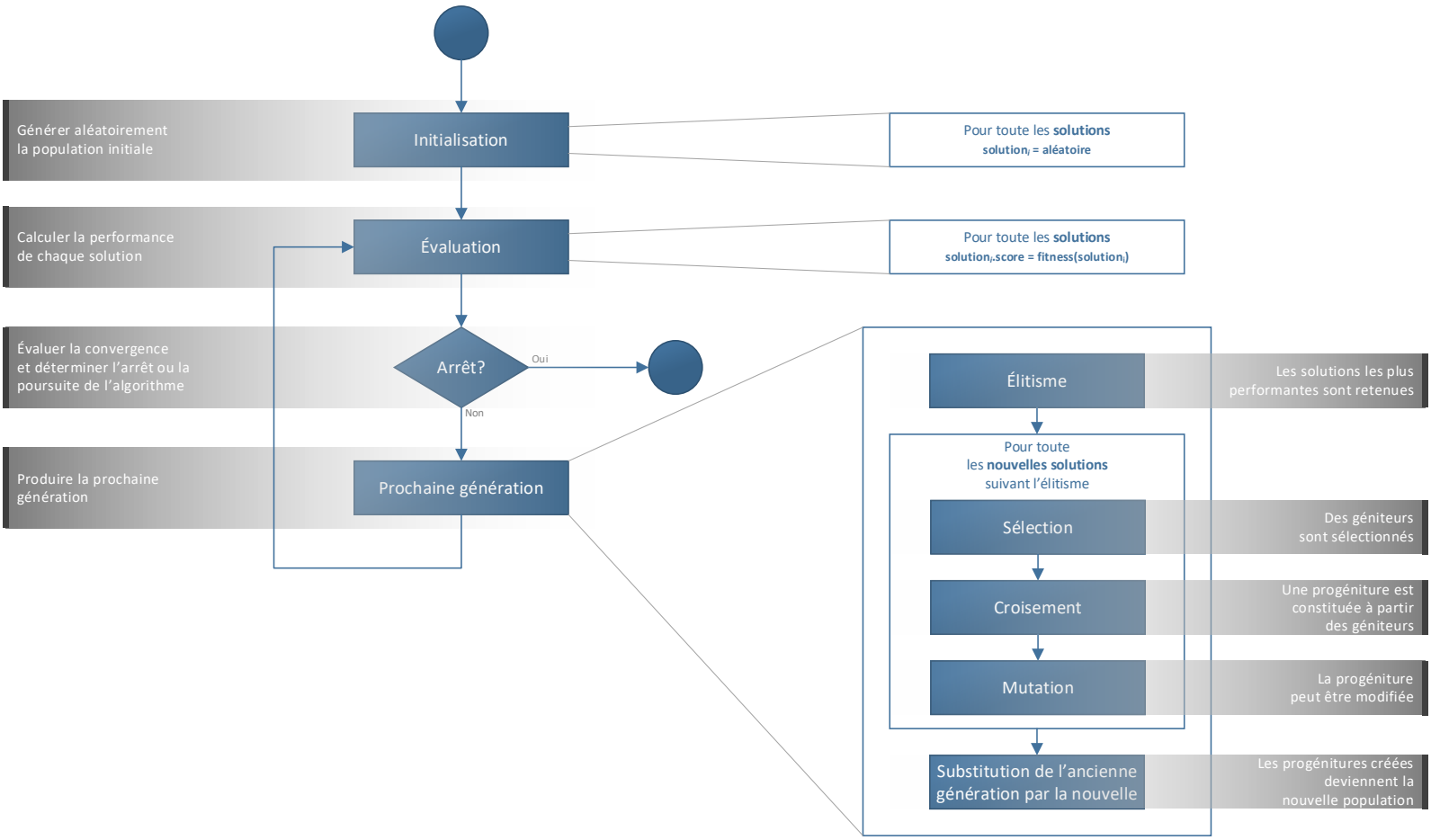
- Le nombre de générations max — détermine un critère d'arrêt correspondant au budget disponible
- La taille de la population — détermine le nombre de solutions participant à l'évolution
- Le taux sélection — détermine le % de la population qui pourra être sélectionné comme géniteur
- Le taux d'élitisme — 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 généralité 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 :

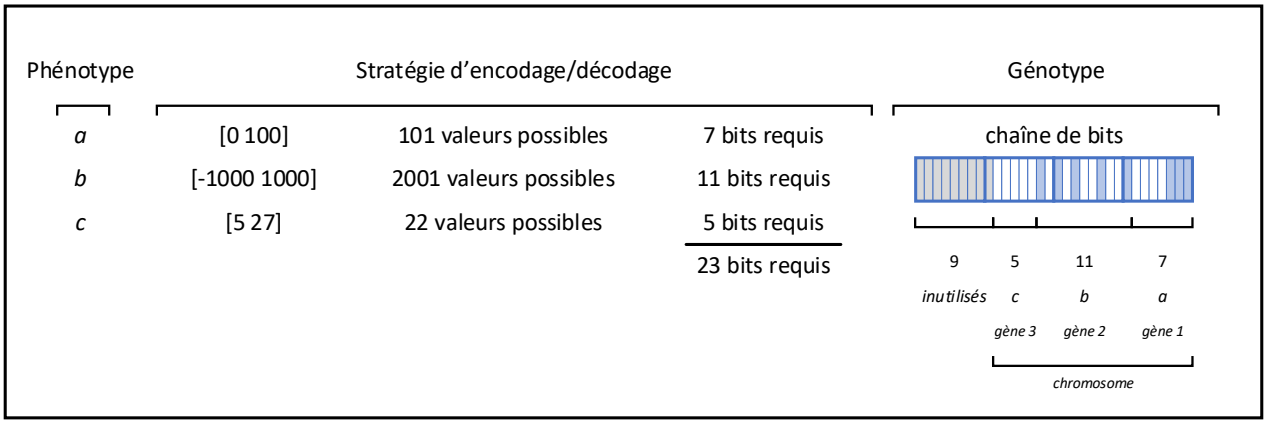
- a : un nombre entier inclue dans l'intervalle [0 100]
- b : un nombre entier inclue dans l'intervalle [-1000 1000]
- c : un nombre entier inclue dans l'intervalle [5 27]

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

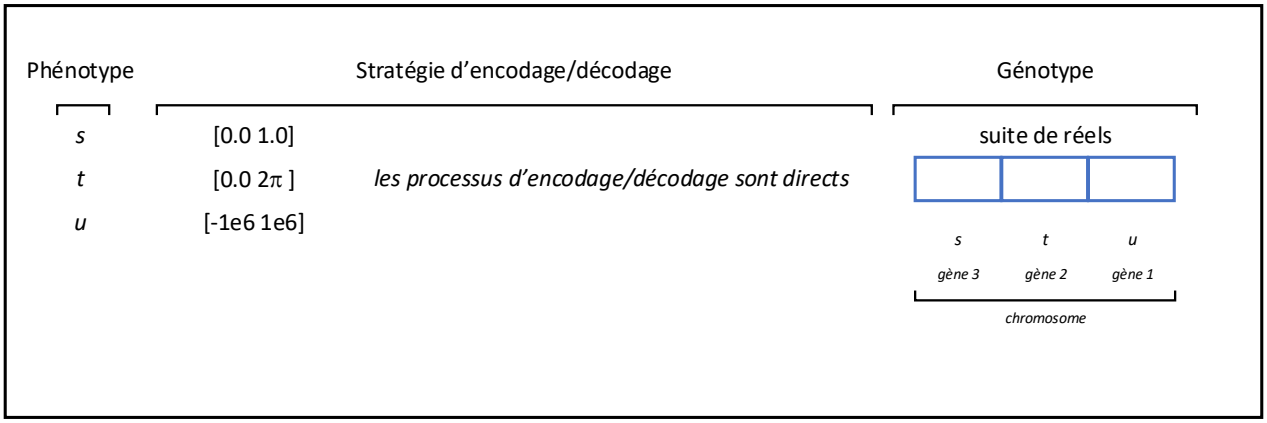
- s : un nombre réel inclue dans l'intervalle [0.0 1.0]
- t : un nombre réel inclue dans l'intervalle [0.0 2π]
- u : un nombre réel inclue dans l'intervalle [-1 000 000.0 1 000 000.0]

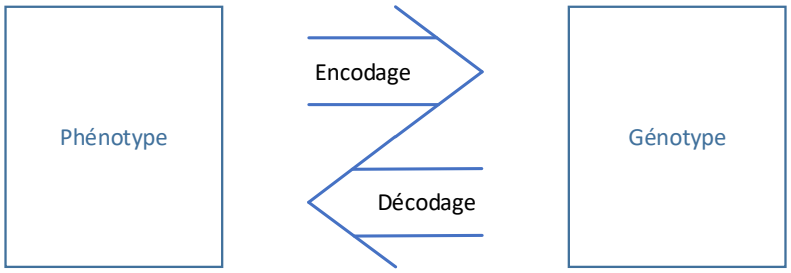
Représentation, encodage et décodage

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).

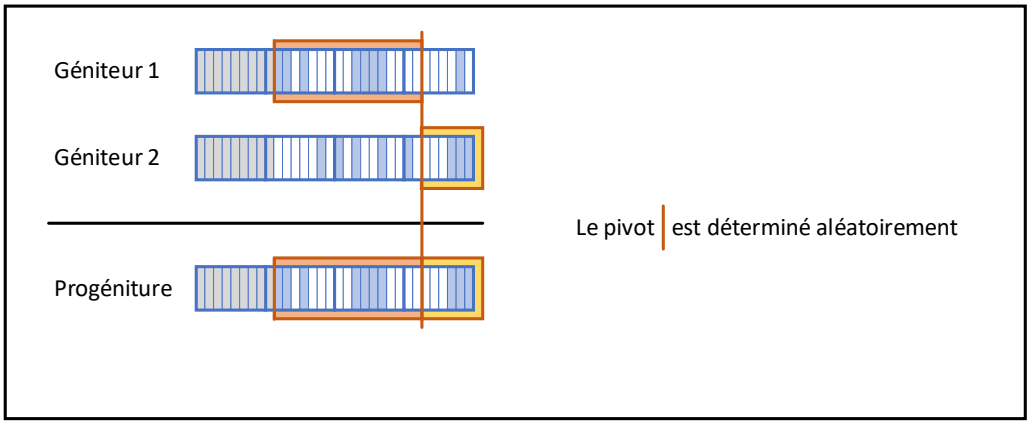
Sélection

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

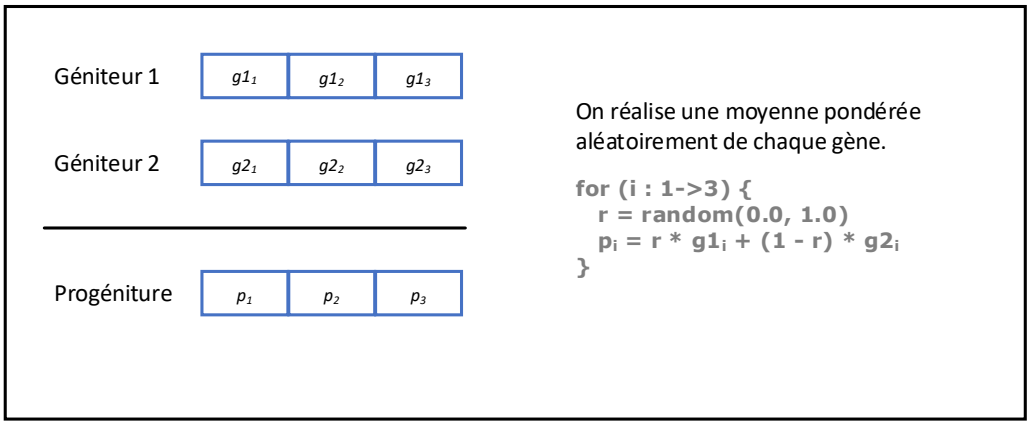
Il n'existe pas de méthode officiellement fondamentale. Toutefois, la méthode « *Roulette Wheel* » est généralement utilisée comme approche générique. Cette approche utilise le poids relatif de chaque individu pour déterminer leur probabilité de sélection. Le choix des géniteurs se fait ensuite par une 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 2e 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.

Finalement, il est important d'effectuer la translation en premier lorsque vous avez plusieurs transformations successives (fait référence au calcul matriciel et à l'ordre de multiplication). Par exemple, pour effectuer les trois transformations, l'ordre de transformation pourrait être :

1. `QTransform.translate`
2. `QTransform.rotate`
3. `QTransform.scale`