Notes préliminaires sur le modèle démographique/génétique

version 1

18/02/21

Préambule

Partie en cours de développement : éditeur de support génétique. Mais tout peut encore facilement être modifié, rien n'est figé pour le moment, mis a part les technos utilisées pour coder le projet, les outils et méthodes sont en place.

Il y a encore beaucoup à faire et à quoi réfléchir encore ! Mais voici les contours du modèle que j'ai définis pour le moment, dont certains mécanismes ont déjà été testés.

Tu m'excuseras ma vision très réduite et simplifié du génome et de ses mécanismes, basée sur mes connaissances du lycée… Je me disais que tu pourrais apporter des nuances, des mécanismes et une vision plus riche pour étayer ce « toy model ».

En amont j'avais pas mal bossé l'année dernière sur le combinatoire et la statistique des phénotypes résultants d'un génome simplifié donné, pour comprendre l'influence de différentes paramètres (nombre de gènes, nombre d'allèles, ploïdie, mécanisme méiose, répartition des gènes sur différents chromosomes etc.) Retrouver le théorème central limite (dans mes souvenirs a partir de 3 gènes on commençait déjà à avoir un spectre gaussien des phénotypes) Par contre, en ce qui concerne les mécanismes de lecture/écriture du code génétique, de la topologie d'interactions entre gènes, des couplages, des différentes types de gène et tout je suis complètement néophyte et peu éduqué.

Les questions que j'adresse, notamment à toit spécialiste en la matière, c'est peut on mieux faire ? Peut on imaginer davantage de libertés que présentées ici avec les contraintes données ? Quels mécanismes inspirés du réel pourraient venir enrichir ou reformuler le modèle esquissé ici ? Est ce que ce modèle peut

produire des choses intéressantes au vu de l'effort qu'il demandera a être développé ? (je le ferai quand même)

Objectifs du projet

Construire une bibliothèque que n'importe qui peut installer et utiliser dans ses propres projets informatiques. Cette bibliothèque est constituée d'un moteur (un ensemble de fonctions implémentant le modèle) ainsi que de données (avec un format bien défini) exportables et importables dans des fichiers/base de données. Ainsi, les utilisateurs pourront partager les données entre eux (design du génome, des traits phénotypiques et espèces) et une communauté *pourrait* se former construisant petit à petit des design de plus en plus élaborés.

Pour permettre cela tout en étant *user-friendly* des éditeurs avec interface graphique seront développés :

a) Un éditeur de support génétique

b) Un éditeur de traits phénotypiques

c) Un éditeur d'espèces

d) Un bac à sable pour tester rapidement ses designs

Les 3 éditeurs permettront d'éditer, importer, exporter, dupliquer, supprimer, ces 3 type de données facilement. Ces données seront ensuite importées dans le projet informatique de l'utilisateur et celui-ci, après import du moteur, pourra faire tourner le modèle dans le but qu'il s'est fixé.

Dans quels buts

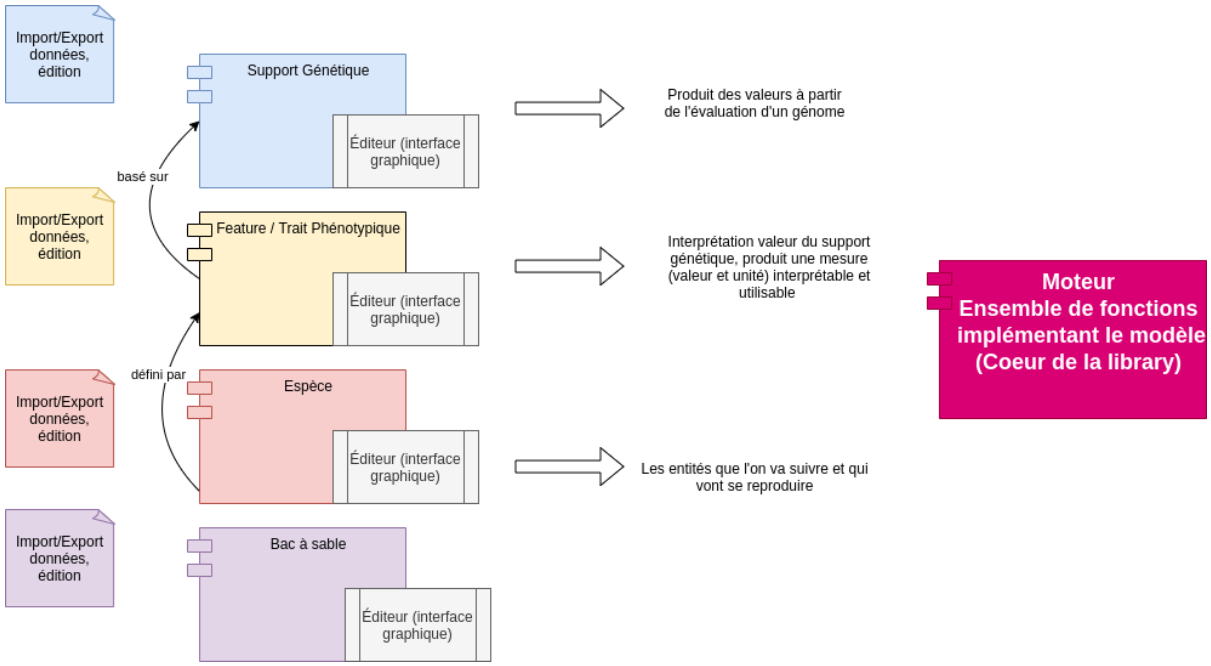
S'amuser. Fournir un moteur permettant de simuler des entités (biologiques ou non) possédant des caractéristiques (traits phénotypiques) interprétables par l'utilisateur. Ces caractéristiques sont le résultat de l'évaluation d'un support génétique (principalement). Les entités pourront se reproduire et transmettre à leur descendance une part de leur génome pour créer de l'hérédité, de la filiation (et de l'histoire). Ce génome pourra également être soumis à des mutations et évoluer. L'utilisateur pourra alors facilement simuler de la sélection et des démographies. Les espèces pourront muter, les populations évoluer etc. Enfin tu vois l'idée, un gros bac à sable. Ici j'essaie de généraliser un peu car là on parle de simuler de la vie,

avec un langage très « bio » mais on pourrait utiliser ce moteur sur d'autres entités auxquelles je n'ai pas encore pensées.

Contraintes du modèle

La contrainte forte du modèle c'est qu'il faut définir à l'avance des traits phénotypiques auxquels l'utilisateur pourra donner **du sens**. J'insiste là dessus car le but est de pouvoir **interpréter fonctionnellement les évaluations des phénotypes pour prendre des décisions.** Ce que j'entends par là c'est que si j'ai une espèce avec un trait phénotypique qui renvoie une valeur disons '2' ou 'foo' il faut que cette valeur **puisse** (pas forcément) avoir un sens pour l'utilisateur et pour son programme. En ce sens, les traits phénotypiques doivent être définis dès le départ et le réservoir de phénotypes possibles doit pouvoir être interprétable à l'avance. En espérant être clair sur ce point.

Aperçu général des composants du modèle



Un aperçu global du projet pour mettre un peu l'ambiance et l'idée générale. On retrouve les principales structures de données:

- Support génétique
- Feature/Trait Phénotypique
- Espèce

Pour éditer chaque structure de données (et l'import/export pour les échanger facilement) un éditeur avec interface graphique (qui tournera dans le navigateur) sera développé. Les données seront exportables (fichiers, base de données). On ne va pas parler détails sur la persistance des données, pour le moment on s'en fout. Ce qui compte c'est que c'est un modèle qui va être « piloté par des données », et qu'on pourra les importer/exporter. Enfin le moteur de la library, tout l'ensemble de fonctions qui va permettre de faire tourner le modèle, qui va se nourrir des données et en fabriquer de nouvelles.

On va maintenant détailler le modèle, étape par étape, en passant en revue les structures de données et et ce qu'elles permettront de décrire. Pour le moment il n'y a pas de description du moteur à proprement parler mais plutôt de la modélisation.

Support Génétique

Un support génétique code pour une *feature* (ou trait phénotypique, même chose), qui sera définie ultérieurement par l'utilisateur. Un même support peut être réutilisé pour plusieurs features. Le support génétique est donc la structure qui définit le *code génétique* et l'expression des gènes. Un support, quand il est évalué **produit une valeur.**

Paramètres d'un support génétique :

- **Nom** (unique, identifiant)
- **Nombre de gènes** : nombre de gènes contribuant à la valeur du support (min : 1)
- **Nombre de bits** sur lequel est encodé chaque gène du support n : l'utilisateur pourra choisir d'encoder des gènes sur des structures allant de 1 à 8 bits (limite arbitraire, on pourrait aller au delà). Ce paramètre défini par défaut le nombre de variations d'un même gène (allèles) accessibles. Par exemple, sur un support à 2 bits on a 4 valeurs possibles : 00 (0), 01 (1), 10 (2) et 11 (3). On a donc, au maximum 4 allèles pour ce gène dans notre « réservoir ». Sur 8 bits on a 256 valeurs possibles, donc 256 allèles possibles pour un même gène. En ce sens, **une allèle est une valeur.** La structure binaire est suffisante pour créer assez de variations (pas besoin de 4 caractères AGCT). Le binaire est aussi intéressant car il est très connu et a des propriétés sympas, qui seront utiles pour simuler les mutations (décaler un bit vers la droite multiplie la valeur par 2 par exemple)
- **Allèles** : on a vu que le réservoir d'allèles était défini par le nombre de bits sur lequel est encodé un gène. C'est le réservoir par défaut. L'utilisateur pourra décider de n'utiliser que certaines allèles, un sous-ensemble du réservoir s'il le souhaite.
- **Règles de codominance** : valeurs comprises entre 0 et 1 permettant de calculer la contribution de chacune des 2 allèles du même gène à l'évaluation totale. Si m est le nombre d'allèles il est nécessaire de définir $m(m+1)/2$ règles pour évaluer deux allèles du même gène. L'utilisateur pourra générer ces coefficients de manière aléatoire ou suivant des règles prédéfinies, ou les écrire manuellement (il y a toujours le contrôle total si on le souhaite).
- **Nature :** Discret ou Continu

- **Discret**: le support génétique, quand il est évalué, produit uniquement des valeurs discrètes. Utile pour modéliser une expression booléenne (ex : absence ou présence d'un trait, ensemble prédéfini de valeurs comme une liste de couleurs etc.)
- **Continu** : le support génétique, quand il est évalué, produit des valeurs dans un spectre continue (ex : taille d'un membre). En fonction des contributions de chaque allèle l'évaluation pourra prendre n'importe qu'elle valeur.

C'est tout pour le support génétique. Voyons un exemple pour être un peu plus concret, un exemple volontairement simple pour les besoins de concision de la doc

Exemple : support génétique continu

Nom : foo

Nombre de gènes : 3

Nature : Continu

Nombre de bits : 2

Allèles : tout le pool, défini par le nombre de bits, $2^2 = 4$

a => 00 : valeur de a = 0

b => 01 : valeur de b = 1

c => 10 : valeur de c = 2

d => 11 : valeur de d = 3

Règles de codominances :

	a	b	c	d
a	1	0.3	0.7	1
b	-	1	0.2	0.4
c	-	-	1	0.5
d	-	-	-	1

Soit u(i,j) le coefficient de codominance entre l'allèle i et l'allèle j, avec par définition

$$u(j,i) = 1 - u(i,j)$$

Évaluons ce support génétique, avec par exemple un génome initialisé

$$v_f = \frac{u(a,b)v_a + u(b,a)v_b + u(c,b)v_c + u(b,c)v_b + u(a,b)v_a + u(b,a)v_b}{3}$$

comme ceci. On est sur du diploïde tout ce qui y'a de plus classique. On a donc 2 allèles pour chacun des gènes (au nombre de 3) : (Le chromosome se lit dans le sens vertical, avec les allèles du même gène face à face, c'est plus rapide que de faire un dessin)

a-b

c-b

a-b

Soit v_f la valeur du support on a

ce qui donne si on fait le calcul, 0.9 (si j'ai pas fait d'erreur, heureusement l'ordi n'en fera pas)

Généralisation : Évaluation d'un support génétique « continu »

$$v_f = \frac{1}{n}\sum_{i=1}^{n} u(i_a, i_b)v_a + (1 - u(i_a, i_b))v_b$$

avec n le nombre de gènes, i_a et i_b allèles a et b du gène i.

Exemple : support discret

Nom : bar

Nombre de gènes : 2

Nature : Continu

Nombre de bits : 2

Allèles : tout le pool, défini par le nombre de bits, $2^2 = 4$

a => 00 : valeur de a = 0

b => 01 : valeur de b = 1

c => 10 : valeur de c = 2

d => 11 : valeur de d = 3

Règles de codominances :

	a	b	c	d

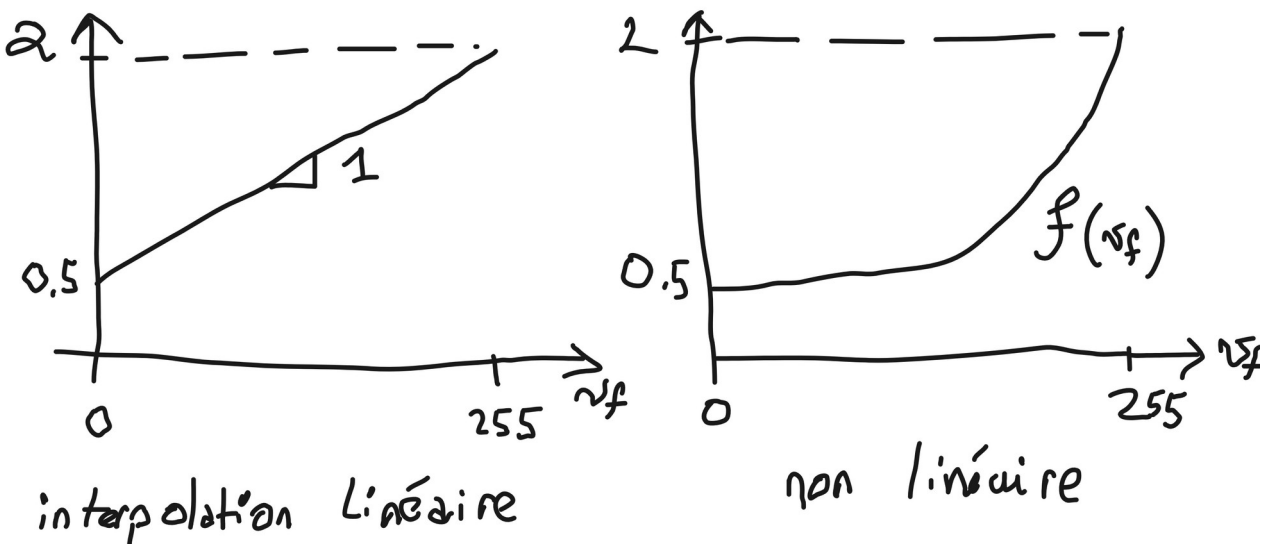


Figure 1: Différentes interpolations possibles pour différents effets : En abscisse la valeur donnée par l'évaluation du support génétique, en ordonnée la valeur de la feature.

a	1	0.3	0.7	1
b	-	1	0.2	0.4
c	-	-	1	0.5
d	-	-	-	1

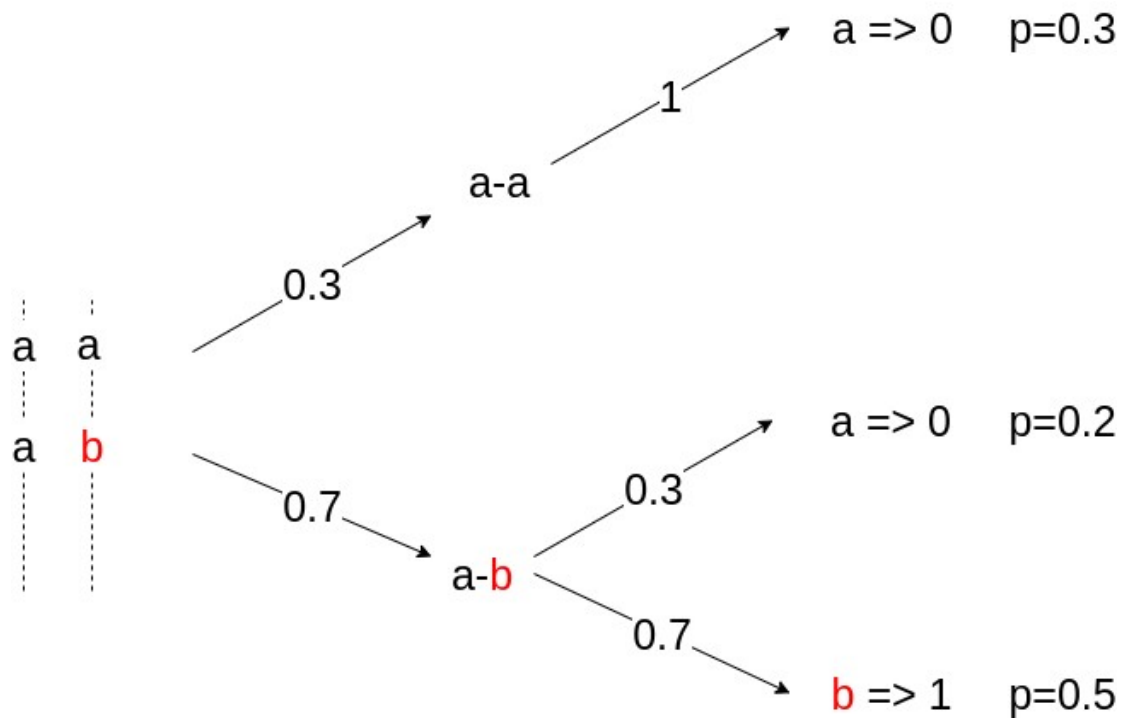
Prenons un exemple initialisé comme suit

a-b
a-a

Dans le cas discret u(a,b)=0.3 indique que lorsqu'on évalue la contribution d'un gène la probabilité que a s'exprime complètement est de 0.3 et celle

de b de 0.7. Ce n'est plus un coefficient de proportionnalité mais une probabilité. Ce système permet (comme dans le cas continu d'ailleurs) d'obtenir un modèle « dominant/récessif. Par exemple ici, u(a,d)=1 indique que a s'exprime dans 100 % des cas face à d (a est dominant, d est récessif).

L'évaluation donne :



Comme on voit ici dans le cas d'un support discret, l'évaluation donne comme résultats un spectre de valeurs discrètes déterminé à la fin par la valeur d'une seule allèle.

Pour le moment c'est tout pour le support génétique, passons à l'échelle supérieure : les features.

Feature / Trait phénotypique

Une fois qu'on a défini des supports génétiques, et qu'on en a une petite collections, on a tout ce qu'il faut pour designer des features. L'intérêt de découpler les 2 c'est que je peux réutiliser un même support pour plusieurs features et personnaliser son usage dans chaque cas. En effet, jusqu'ici on a seulement parlé de support génétique produisant des valeurs. Aucun sens n'a encore été donné à ces valeurs. C'est le rôle de la feature.

Dans la suite de l'idée une espèce sera définie (notamment) par un ensemble de features. Les éditeurs nous permettront d'éditer tout ce contenu *statique* facilement (et importer/exporter).

C'est donc ici qu'on va mettre la couche « donner un sens » aux valeurs générées par le support génétique. La feature est une grandeur que nous allons définir. Pour l'utiliser, lui donner sens, nous allons avoir besoin d'une valeur et d'une unité de mesure. En effet ici s'opère un glissement important, on passe d'une **valeur à une mesure (je vais cependant garder le mot valeur pour la feature et souvent omettre l'unité de mesure par soucis de commodité mais il faut le garder en tête, car c'est tout l'intérêt de la feature elle même)**. La mesure pourra être un entier, un nombre décimal ou une chaîne de caractères. L'unité de mesure sera définie par l'utilisateur lui même (pas abordé dans ce document. Ça pourrait être des 'm/s', des 'mols', des 'kg', des 'grade', des 'milipilpoils'. Ça servira par la suite à faire des vérifications de cohérence comme dans la vraie science. Quand on mesure un truc il faut savoir par rapport à quoi !)

Paramètres d'une feature :

Nom (unique) : identifie et désigne la feature

Support génétique : sur quel support est basé cette feature

Rescaling des valeurs : notre support génétique produit des valeurs continues ou discrètes en fonction de sa nature. Dans le cas discret ces valeurs appartiennent à un ensemble prédéfini de valeurs, l'ensemble des valeurs de chaque allèle. Dans le cas continu, ces valeurs appartiennent à un intervalle borné par un min/max. La valeur max est connue car elle est définie par le nombre de bits sur lequel est codé un support génétique. Par exemple, si le support génétique est codé sur 8bits il y a exactement 256 allèles et la valeur max est 255.

Cas continu : aucun problème particulier. On peut rescaler (remapper) les valeurs vers un intervalle qui fait *sens pour nous*, pour notre *feature*. Par exemple, si ma feature est sensée exprimer la taille d'un être humain, on veut passer de [0:255] (intervalle du support génétique) à [0.5:2] pour la taille d'un être humain. (Il manque encore l'unité de mesure ici). On peut rajouter un paramètre supplémentaire : **le choix de l'interpolation**. On peut faire une interpolation linéaire ou avec n'importe quel polynôme. Ainsi une variation infime du génome (brassage, mutation) peut conduire à des variations phénotypiques plus fortes. On peut également imaginer définir des intervalles dans le cas continu. Si la valeur est entre 0.5 et 0.7 alors la feature est évaluée(ou mesurée) à 1.9, ou à 'rouge'.

Cas discret : dans le cas discret on va avoir des problématiques nouvelles où il va falloir faire des choix.

Dans le cas d'absence de mutations d'abord. Comme la feature est discrète elle ne remonte que des valeurs prises par les allèles elle-même. Il faut donc, si l'on souhaite interpréter les valeurs du support génétique (on peut aussi vouloir les valeurs brutes mais moins d'intérêt) **donner une mesure pour chacune d'entre elles,** ce qui peut être fastidieux s'il y'en a 256 (ou plus). On pourrait définir des intervalles comme évoqué précédemment dans le cas continu. Par exemple, j'ai les allèles 0, 3 et 200. Si la valeur est supérieure à 100 la feature sera évaluée à 'or', sinon à 'argent'. C'est une possibilité qui peut être intéressante. En plus on peut donner à l'utilisateur la possibilité d'éditer ces valeurs manuellement (c'est du travail mais c'est du boulot de designer un génome intéressant). En plus, on peut lui permettre de définir la mesure seulement pour certaines allèles. Par défaut les autres seront évaluées à une valeur générée par le système, par exemple '*unknown*'. L'utilisateur pourra décider quoi faire dans ce cas dans son programme : il pourra notamment définir dynamiquement cette nouvelle mesure (au cours de l'exécution donner une valeur à cette feature). Prenons un exemple :

```
//Au sein du programme de l'utilisateur qui fait tourner une simulation

x = evaluate(individu, feature) // renvoie l'évaluation de la feature de l'individu

if( 'unknown' == x) // une mutation a eu lieu ou l'allèle qui s'exprime n'a pas été associée à une valeur dans la feature
```

setValue(individu, feature, ‘nouvelle valeur’, unité)

// l’utilisateur donne un sens de manière dynamique à cette allèle au cours de l’exécution du programme. En effet, comme c’est discret on sait déterminer quelle allèle précisément s’est exprimée ici, et on peut donc la retrouver et lui assigner une valeur.

Dans le cas de mutation. On a pas encore abordé les mutations jusqu’ici mais c’est pas grave faisons un petit détour. Le support génétique pourrait muter : par exemple un bit pourrait se décaler vers la gauche. Par exemple, un support génétique codé sur 8bits a une allèle qui vaut initialement 00000001 (1) est mutée (le bit se décale) à 00000010 (2). Il est alors possible que je n’ai pas associé de valeur (dans ma feature) à cette allèle car elle ne faisait pas partie de mon réservoir de départ. Ainsi, on garde de la souplesse : soit l’utilisateur définit une valeur pour chaque allèle (ou une valeur par défaut pour toutes les allèles/mutations) soit il peut les définir dynamiquement dans son programme. En bref, le cas discret dans le cas de mutations demande plus d’effort si on veut garder de grandes libertés de variations, mais c’est bien normal. Mais tout ça pour dire qu’on peut proposer beaucoup de solutions pour gérer ça (j’épargne tous les détails ici).

Digression rapide sur les mutations et le binaire :

Ce qui est intéressant c’est qu’on peut imaginer introduire très facilement en plus des gènes non codants sur les chromosomes. Par exemple :

….00000001**0000100**11011**11011001**00...

Gène codant 1 allèle a (valeur en décimal : 19)

Gène codant 2 allèle b (valeur en décimal :179)

Gène non codant

Maintenant on peut imaginer une forme de mutation où un bit se décale de 1 pas sur la gauche et décale tous les autres, par exemple celui en gras. Le résultat serait cette séquence (me suis peut être trompé c’est chiant à faire)

….00000010**0010011011110110011001**....

Gène 1 allèle a (valeur en décimal : **38**)

Gène 2 allèle b (valeur en décimal :**102**)

Gène non codant

On pourrait ainsi donner naturellement un rôle sympa aux séquences non codantes. Après je te laisse imaginer toutes les opérations cools de mutation qu'on peut faire par des opérations très simples sur les bits.

Suite des paramètres :

Unité de mesure : étalon, permet de recalculer la valeur de la feature dans un autre système de mesure si besoin, analyse dimensionnelle [en vrai optionnel, on peut garder la liberté de ne pas en mettre et garder des valeurs abstraites]

Couplages entre features : créer des features à partir de features. C'est notamment là où ça devient intéressant

Voici une notion un peu plus avancée du modèle mais qui permet, je crois, de lui donner beaucoup d'intérêt et de puissance. Pour le moment, on peut définir des features, basées sur des supports génétiques. On verra par la suite qu'une espèce sera définie par un ensemble de features (notamment). C'est cool déjà, mais c'est embêtant car toutes ces features sont sagement les unes à coté des autres, à s'ignorer mutuellement et à vivre leur vie (bien qu'on parlera des mutations par la suite). En soi on peut laisser au soin de l'utilisateur de combiner les effets des features pour fabriquer de nouveaux mécanismes lui même. Mais on peut aussi intégrer nativement cette possibilité pour le bonheur de tout le monde, et créer de manière récursive et *infinie des features à partir d'autres features*. Créer un réseau de features pour fabriquer des nouvelles features, pour fabriquer des nouvelles features, pour… Voyons cela plus en détails, à partir d'un exemple concret.

Espèces