Cours n°1

1 Les objectifs du cours

Ce cours a pour objectif de montrer une palette de méthodes de modélisation et de simulation pour comprendre la dynamique de phénomènes de biologiques, pour « prédire » *in silico* divers comportements relatifs aux fonctions biologiques étudiées et pour, le cas échéant, savoir comment contrôler ces fonctions biologiques. Ces « prédictions » émergent de règles généralement assez simples au niveau des interactions élémentaires et mettent en évidence des comportements globaux lors des simulations ou de l'étude théorique de ces modèles.

Principalement deux causes font émerger des résultats :
- des comportements collectifs intrinsèques à la nature des interactions et aux paramètres qui les régissent.
- ou des artefacts résultant des hypothèses, fortement simplificatrices par rapport à la réalité biologique, faites pour modéliser informatiquement les phénomènes,

Naturellement, seule la première cause a de l'intérêt et un bon bio-informaticien doit apprendre à avoir un œil critique suffisamment exercé pour détecter autant que possible la seconde cause : *l'essentiel du cours est dirigé en ce sens.*

Nous allons apprendre à définir des modèles de *systèmes dynamiques* :
- Un système dynamique requiert par définition un ensemble d'états possibles E et un ensemble de conventions qui disent comment un état $\eta \in E$ évolue au cours du temps.
- L'ensemble T des *temps* possibles, lui-même, peut être défini de différentes façons. On pense par défaut à un temps continu avec $T = \mathbb{R}^+$. Cependant un temps *discret* ($T = \mathbb{N}$) est souvent une bonne approximation plus simple à manipuler, parce qu'alors on peut définir une fonction « état suivant » qui dit comment calculer l'état à l'instant $n+1$ à partir de la connaissance de l'état à l'instant n.
- À partir des ensembles E et T choisis précédemment, un système dynamique permet alors de caractériser l'ensemble des trajectoires $\pi : T \to E$ que peut suivre le système, et le cas échéant une mesure de probabilité de suivre une trajectoire plutôt que d'autres. Les $\pi(0)$ des trajectoires π sont appelés les *états initiaux* et pour chaque temps t, l'état du système à cet instant est $\eta_t = \pi(t)$.

Les *automates* (au sens informatique du terme), dont on connaît déjà l'importance (par exemple pour tous les aspects de traitement des séquences génomiques), jouent un rôle majeur pour les applications à la biologie des systèmes car ce sont des systèmes dynamiques discrets simplement en considérant que E est l'ensemble fini des états de l'automate, que $T = \mathbb{N}$ et que chaque transition de l'automate se fait en une unité de temps.

Les connaissances acquises durant ce cours seront les suivantes :
- les *automates cellulaires*, principalement en 2D,
- une extension vers les *systèmes multi-agents* pour la simulation en 3D (HSIM),
- les méthode d'analyse de flux pour modéliser les *réseaux métaboliques* (FBA),
- les techniques de simulation des *réseaux de signalisation* et le langage BioCHAM,
- les techniques de modélisation des *réseaux biologiques* fondées sur les réseaux de Petri.

Le système multi-agent HSIM est téléchargeable sur `http://www.lri.fr/~pa/Hsim/`
Il existe des versions pour Windows, Mac et Linux. Prévenez l'enseignant en cas de difficulté à l'installer.

2 (Parenthèse sur la programmation)

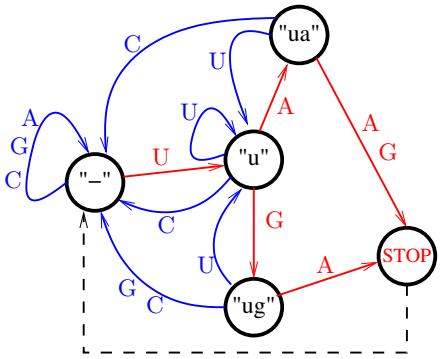
Ce cours est un des premiers de l'option BIMB. Quelques compléments sont abordés sur le *style et l'élégance de programmation*, la lisibilité des programmes, l'importance majeure des structures de données et la programmation structurée. En vrac, on a abordé en particulier les sujets suivants :
- `https://webusers.i3s.unice.fr/~bernot/Enseignement/GB3_Python1` contient un cours complet d'initiation à la programmation en utilisant Python.
- Importance du style et de l'esthétique de la programmation, en particulier pour la maintenance des programmes
- Les différents paradigmes de programmation (impératif, objet, fonctionnel, logique)
- Les deux aspects essentiels d'un langage de programmation : les *structures de données* et le *contrôle* ; au passage :
  - « contrôle de flux » c'est totalement autre chose (gestion des réseaux)
  - et `if then else` s'appelle une *instruction conditionnelle* (les « tests » en logiciel, c'est pour tester le logiciel est trouver des bugs)

— une *structure de données* est définie par un *type* (=son nom), l'ensemble des valeurs de ce type, et enfin les opérations qui travaillent dessus et leur sémantique. Des conversions de type sont parfois possibles (`str`/`int`/`float`/...)
— Comment concevoir un programme :
  — commencer par réfléchir aux difficultés intrinsèques du problème à résoudre par ce programme
  — puis réfléchir aux structures de données adéquates
  — et seulement *après* se lancer dans la programmation
— Les fonctions et procédures (`def`, différence entre `return` et `print`)
  — et surtout, notion d'*effet de bord*
— Variables globales / locales
— Toujours séparer l'interface utilisateur du noyau du programme (pas de `input` ou de `print` des résultats au sein d'une fonction qui fait des calculs ou qui gère les structures de données internes !)
— Rappels sur `while`/`for` (et existence du `until` dans certains langages)
— Rappels sur les listes et dictionnaires (les indices des structures linéaires vont de `0` à `length-1`, éviter par défaut les listes hétérogènes pour faciliter les traitements itératifs, *idem* pour les clefs des dictionnaires)
— Programmation structurée (la date du lendemain). Après avoir réfléchi aux difficultés intrinsèques du problème et choisi des structures de données adéquates, s'attaquer au problème global et si une question est compliquée, supposer qu'il existe une fonction qui la résout...

3 Les automates

La notion générale d'automate admet plusieurs variantes mais elles reposent toutes sur les principes de base suivants :
— Un automate est défini par un graphe orienté étiqueté dont les nœuds sont les *états* de l'automate (E) et les arcs sont des *transitions*, franchissables dans certaines conditions qui sont des étiquettes portées par les arcs. Par exemple la figure 1 représente les transitions utiles pour reconnaître un codon STOP (UAA, UAG ou UGA) dans une séquence d'ARN.

Figure 1 – Reconnaissance des codons STOP d'une séquence d'ARN



— On peut définir l'automate en fournissant le graphe « en extension », comme pour les algorithmes de recherche d'un motif dans une séquence. On peut aussi définir le graphe « en intension » en décrivant les propriétés qui caractérisent l'ensemble des états et en définissant les règles de transitions entre états.
— On simule le temps ($T = \mathbb{N}$) avec une fonction de calcul de l'état suivant simplement donnée par les arcs du graphe, en partant d'un des états ($\pi(0)$), appartenant à un ensemble d'*états initiaux* prédéterminé, puis en changeant successivement d'état $(\pi(1), \pi(2), \cdots)$ en suivant à chaque fois une des transitions franchissables. Le temps est donc *discret* en ce sens que chaque franchissement de transition fournit un « top horloge » et que le temps est simplement représenté par le nombre de ces tops depuis l'instant 0 à l'état initial.
— Plusieurs transitions peuvent être franchissables à partir d'un état donné ; la dynamique du système est alors à définir : ordre de priorité entre les transitions qui détermine la transition à choisir, choix aléatoire de la transition à franchir, choix probabiliste, *etc.* L'ensemble des choix autorisés définit pas à pas l'ensemble des trajectoires : si à un instant t on peut sortir de l'état $\pi(t)$ par n transitions différentes, alors cela donne lieu à n prolongations possibles de π, donc à au moins n trajectoires $\pi_1, \cdots, \pi_n$ et pour chacune de ces trajectoires π_i il y a encore autant de trajectoires possibles prolongeant π_i que de transitions possibles sortant de l'état $\pi_i(t+1)$, *etc.*
— Les états peuvent de plus être aussi compliqués que l'on veut et peuvent éventuellement porter beaucoup d'informations. On utilise souvent des *langages logiques* pour exprimer des propriétés sur ces états et des logiques dites *temporelles* pour exprimer les propriétés sur les changements d'états (la dynamique du système donc).

Pour illustrer ce propos général, commençons par l'exemple des *automates cellulaires*.

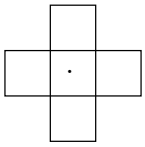
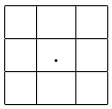
4 Les automates cellulaires

4.1 États d'un automate cellulaire

Un *automate cellulaire* est un automate dont les états sont obtenus en découpant un espace donné (par exemple une droite ou un plan [1]) en « morceaux » disjoints tous identiques. Ainsi une droite sera découpée en intervalles (par exemple de longueur 1), un plan peut être découpé en carrés jointifs ou encore en « nid d'abeille », *etc.*
Chaque *case* de l'espace considéré est appelé une « cellule », d'où la terminologie « automate cellulaire » (rien à voir avec la notion biologique de cellule donc). De plus chaque case possède des cases voisines selon un « pattern » (motif) fixé qui permet de se repérer avec une notion de directions. L'ensemble des directions permises définit une notion de *voisinage* de toute case par union des directions considérées.

La plupart du temps le découpage de l'espace est effectué selon des coordonnées cartésiennes entières, ce que l'on fera tout au long de ce cours ; mieux, on se limitera aux dimensions 1 et 2 pour les automates cellulaires (et Hsim est un système multi-agent de dimension 3).

Les deux voisinages classiques en 2D avec un découpage cartésien sont :

sans les diagonales  ou avec les diagonales 

On considère par ailleurs un ensemble fini d'états possibles d'une case. Par exemple une case peut être coloriée et la couleur est alors l'état de la case. Ou encore une case peut être vide ou pleine, ou bien encore vivante ou morte, *etc.* Un état η de l'automate dans son entier est alors la connaissance de l'état de chacune des cases qui le compose.

On fait évoluer dans le temps le contenu des cases avec pour convention que la transformation d'une case ne dépend *que* de l'état de ses cases voisines. Les phénomènes biologiques dont la dynamique est régie par des influences locales devraient donc *a priori* pouvoir être simulés ainsi.

On peut par exemple supposer que les cases sont tellement petites qu'une seule molécule rentre dedans à un instant donné. L'état d'une case est donc la molécule qu'elle contient, ou bien vide. Ensuite, si deux cases voisines contiennent des molécules qui réagissent ensemble, ou qui forment un complexe, ou autre, alors on peut expliciter comment chaque case va évoluer après la réaction.

4.2 Règles de transformations d'un automate cellulaire

Une *règle de transformation* est de la forme

état d'une case et de son voisinage → prochain état potentiel de la case

Exercice 1 : Avec un automate 1D, en supposant qu'une case puisse être blanche, noire ou grise, écrivez les règles qui font qu'une case devient de la couleur de ses deux voisines si elles sont de même couleur blanche ou noire, garde sa couleur si l'une d'elles est grise et l'autre de la même couleur qu'elle, et devient grise sinon.

Attention : une règle ne concerne que la mise à jour de la case centrale.

Dans le plan cartésien avec voisinage complet, un automate cellulaire synchrone est donc décrit par des règles de la forme

x_1	x_2	x_3
x_4	c	x_5
x_6	x_7	x_8

$\longrightarrow$ | c' |

Exercice 2 : Un cas très simple de règle en 2D avec un nombre d'états fini est par exemple que chaque case peut être blanche ou noire et prend la couleur de la majorité de ses voisins, et reste inchangée en cas d'égalité. Écrivez les règles avec un voisinage en croix.

Une extension possible est d'avoir un ensemble d'états pas nécessairement finis dans chaque case. Par exemple un état peut être un entier naturel et la règle de transformation est la moyenne arrondie de ses voisins. Cette

1. mais ce peut être aussi un tore ou une sphère ou n'importe quel autre espace topologique

règle pourtant très simple est très utile dans les *convolutions* qui lissent ou floutent des images (ce sont aussi des transformations de base dans les réseaux de neurones artificiels en I.A.).

Conventions dans ce cours :
- Les x_i et c peuvent éventuellement valoir « * » et dans ce cas on considère que n'importe quelle valeur de la case correspondante est acceptable pour appliquer la règle.
- De plus, si la notion de « case vide » a un sens pour le problème modélisé, on la notera avec un tiret « - »
- *Attention*, l'usage des * peut facilement rendre l'automate non déterministe...

Exemple de jeu de règles non déterministe : avec le voisinage en croix, on applique l'exercice 1D à la fois horizontalement et verticalement.

Pour se familiariser, on va commencer par des automates déterministes, avec la stratégie de mise à jour des état la plus simple (i.e. synchrone et non probabiliste).

5 Le jeu de la vie

L'exemple intéressant le plus facile et connu d'automate cellulaire, dans le plan, est certainement le *jeu de la vie*. Il fut inventé par John Horton Conway en 1970, alors qu'il était professeur à l'université de Cambridge, au Royaume-Uni. Comme on le verra plus loin, il s'agit d'un automate cellulaire 2D, synchrone, déterministe, avec mises à jour non probabilistes et avec la notion de voisinage à 9 cases.

Les règles locales en sont simples. À chaque étape, on met à jour toutes les cases en même temps (dynamique synchrone). L'évolution d'une cellule est entièrement déterminée par l'état de ses huit voisines de la façon suivante :
- une cellule morte possédant exactement trois voisines vivantes devient vivante (elle naît) ;
- une cellule vivante possédant deux ou trois voisines vivantes le reste
- dans tous les autres cas, elle meurt.

Certaines formes évoluent alors de manière remarquable. Par exemple un « planeur » dans le jeu de la vie est un motif ayant la forme suivante

v	v	-
v	-	v
v	-	-

où une case vide représente une cellule morte et une case vivante contient un « v ».
En simulant à la main l'évolution d'un planeur dans un plan dont toutes les autres cellules sont mortes, on redécouvre l'intérêt de cette appellation... Le faire !

Même avec des règles aussi simples que le jeu de la vie, on peut voir « émerger » bon nombre de comportements « auto-reproductifs ». En vous connectant sur Wikipédia, consultez le jeu de la vie et entre autres le canon à planeurs.

Exercice : [subsidiaire] Pour écrire les règles du jeu de la vie formellement, combien faudrait-il de règles ? et de quelle forme ?

6 TD sur la modélisation de feu de forêt

Dans cette partie, on va modéliser la propagation d'un feu de forêt au moyen d'automates cellulaires en 2D. On considérera les voisinages classiques avec 8 directions possibles autour de la case centrale. Le découpage est supposé tel qu'il y a un arbre par case.

Exercice 3 : Un arbre pas encore atteint par le feu sera dit *sain*. À l'inverse, un arbre ayant totalement brûlé sera dit *calciné*, et il n'est donc plus en feu. Écrivez les règles qui traduisent les deux idées suivantes :
- Un arbre sain, lorsqu'il est atteint par le feu, brûle pendant 3 temps de l'automate avant de devenir calciné.
- Un arbre sain ayant au moins 2 arbres en feu dans son voisinage devient en feu.

On s'autorisera des notations permettant de factoriser plusieurs règles en un seul schéma de règles, sous réserve de définir clairement le sens des notations utilisées. De plus, il faudra calculer précisément le nombre total de schémas de règles proposés.

Réponse :

*	*	*
*	f1	*
*	*	*

→ f2 — | | | |
|---|---|---|
| * | * | * |
| * | f2 | * |
| * | * | * |

→ f3 — | | | |
|---|---|---|
| * | * | * |
| * | f3 | * |
| * | * | * |

→ c

f1∨f2∨f3	f1∨f2∨f3	*
*	s	*
*	*	*

→ f1 et les $C_2^8 = \frac{8!}{6!\times 2!} = \frac{7\times 8}{2} = 28$ schémas de règles correspondant aux arrangements de 2 cases du voisinage parmi 8.

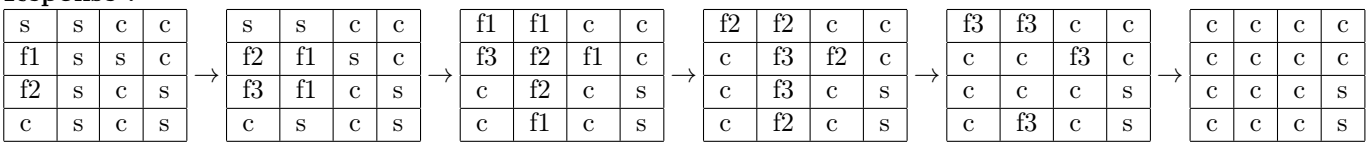
Total 28+3=31 schémas de règles.

Exercice 4 : Tracez l'évolution des états successifs de l'automate dont l'état initial est le suivant :

s	s	c	c
f1	s	s	c
f2	s	c	s
c	s	c	s

où « s » désigne un arbre sain, « f1 » un arbre qui vient d'être atteint par les flammes, « f2 » un arbre dans son deuxième temps de feu, « f3 » son troisième et « c » un arbre calciné.

Réponse :



. . . de l'intérêt de la technique du contre-feu qui sauve les 2 derniers *s*.