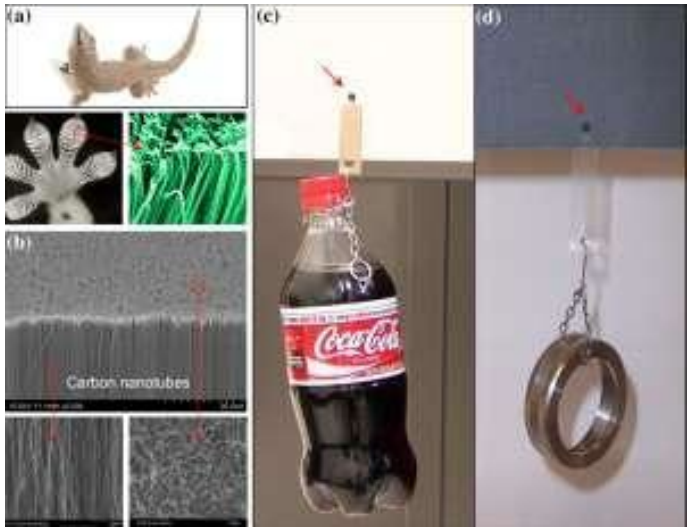
S8 - BE
Introduction aux systèmes collaboratifs
(BE Colonie de Fourmis)
Ecole Centrale de Lyon
2022-2023

Alexander Saidi

I.1 Intelligence Emergente



Intelligence Naturelle : l'homme s'inspire de la nature (source *http ://sciencenordic.com*)

- Les oiseaux et les avions
- Toiles d'araignée artificielle pour des gilets pare-balles
- Le scratch (*Velcro* est une marque !) s'inspire de *bardane* (une plante avec bcp. de propriétés).
- Super adhésif inspiré des *geckos* : création de nanotubes de carbone pour collage

  (ici une bouteille de 650gr sur une bande de 4x4mm, ...)

Intelligence Naturelle : ../..

- La peau des requins contient un motif spécial qui permet une circulation rapide d'eau et empêche l'encrassement.

  ➛ Utilisé dans les technologies pour les bateaux / nageurs / surfeurs / ... qui s'en inspirent !

- Les gouttelettes d'eau ruissellent sur une *feuille de la fleur de lotus*, qui est couverte de nombreuses pointes microscopiques qui empêchent l'eau de pénétrer. . . .

- Et les médicaments copiés sur les molécules naturelles ...

- Les systèmes *biologiquement inspirés* ont pris de l'importance

➙ beaucoup d'idées / innovations *imitent* la nature

- La nature a inspiré / inspire l'homme de multiples façons :

Pour les formes et les surfaces,

Pour les procédés et les matériaux,

Pour les écosystèmes (et organisations)

  - Les avions ⇔ structures d'ailes d'oiseaux.
  - Les robots et leurs mouvements ⇔ insectes.

➙ On a découvert que les insectes qui rampent avec à la fois 3 pattes au sol sont moins énergivores.

  - La soie d'araignée plus résistante que le *Kevlar* ; la soie nat. résiste à -200
  - Réseaux routiers (logistique) imitant la nature (plantes),
  - Algorithmes Génétiques, Algorithme immunitaires, ...
  - &c.

Constatation : certains systèmes sociaux dans la nature présentent un **comportement intelligent collectif**, même si elles sont composées de **simples individus** avec des capacités limitées (cf. les insectes).

La **Stigmergie** : $\simeq$ (auto) incitation :

- On remarque (p.ex. en biologie) que des solutions intelligentes émergent naturellement de l'**auto - organisation** et de la **communication directe ou indirecte** entre individus.
  - On utilise ces solutions dans le développement de systèmes IA distribués.

- **Notre propos** : s'inspirer du comportement collectif des fourmis :
  - Leur capacité à trouver le plus court chemin jusqu'une source de nourriture ,
  - Concept imité en optimisation pour résoudre des problèmes complexes (cf. TSP),
  - Plus généralement : dans la recherche de chemins particuliers dans un graphe,
  - ...

- **Pour ce BE** : plusieurs sujets dont :
  - 1- SLAM (Cours 2, Robot disponible)
  - 2-L'implantation d'un système **multi-agents** de recherche du chemin le plus court (PCC) suivant le principe de la **Stigmergie** en colonie de fourmis (ACO) amélioré par les algorithmes **génétiques**.
  - 3- Bus Allocation Problem (BAP)
  - 4- Coloration de graphes par colonie de fourmis / PSO (Bonus en plus)
  - 5- Algorithmes Génétiques en génération d'images
  - 6- Routage et trafic (voir le pdf)

☞ Le sujet "navigation probabiliste" cherche élèves courageux !

I.2 Colonies de fourmis

Contexte : on s'inscrit dans le cadre **ACO** (*Ant Colony Optimization*) :

• L'optimisation de colonie de fourmis (ACO) étudie les systèmes artificiels qui imitent le comportement des vraies colonies de fourmis.

• Employée pour résoudre des problèmes discrets d'optimisation.

• Constitue une classe de méta-heuristiques pour des problèmes d'optimisation difficiles (NP-difficiles) et les problèmes **dynamiques** (en cours de résolution).

• Voir support et cours 1 pour "un peu d'histoire" sur ce sujet.

I.2.1 Principes

- On aura des "fourmis" (artificielles) :

  Agents de recherche qui miment le comportement des fourmis réels.

- Question posée par les éthologues :

  Comment des insectes qui ne "voient/entendent" pas trouvent le PCC entre leur nid et la nourriture ?

- Réponse : ils utilisent le milieu (environnement) pour communiquer (entre les individus).

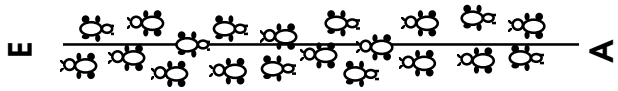
- La phéromone : une fourmi qui se déplace laisse une quantité variable de phéromone sur son chemin, qui est détectée par les prochaines fourmis leur permettant de déterminer le meilleur chemin (avec une bonne probabilité).

- Une forme de **boucle rétroactive positive** ("autocatalytic behavior") :

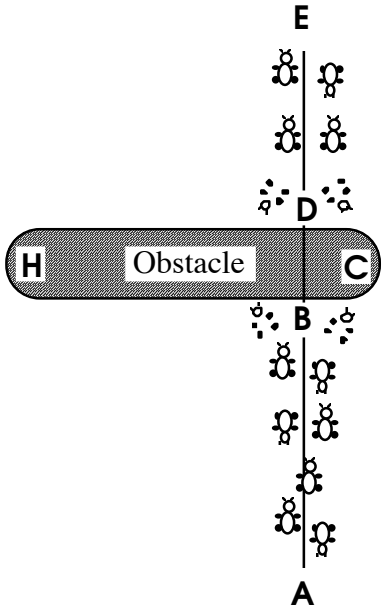
→ Plus les fourmis suivent un chemin, plus ce chemin devient attractif.

I.2.2 Exemple

- Les fourmis se déplacent entre le nid (E) et la source de la nourriture (A) sur le chemin A-E.



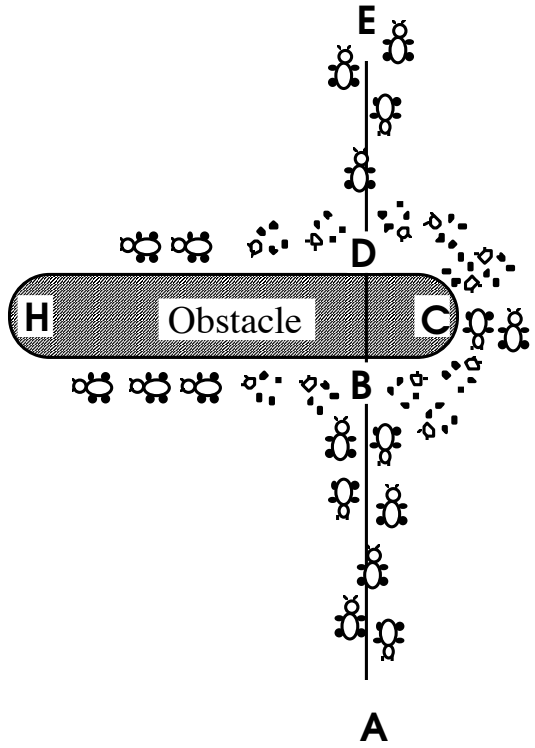
- Un obstacle coupe le chemin.
- La fourmi qui se déplace de A vers E et qui se trouve en B a 2 choix :
  - Le chemin B-C-D
  - Le chemin B-H-D
- *B-C-D* ou *B-H-D* ?
  → le choix selon **f***(intensité de la phéromone)*



- La première fourmi a une même probabilité de suivre l'un de ces deux chemins.
- Celle qui suit le chemin B-C-D arrive en premier en D ...

Puis (il faut retourner au nid) :

- Les fourmis qui retournent de E ont deux choix en D :
  - Le chemin D-C-B
  - Le chemin D-H-B
- Le chemin D-C-B aura une plus forte intensité de phéromone,
- Cette intensité est causée par :
  - La moitié des fourmis qui prennent ce chemin de retour.
  - Le nombre supérieur de fourmis qui auront suivi le chemin B-C-D et qui retournent au nid.



- Le chemin le plus court reçoit davantage de phéromone du fait du passage plus fréquent des fourmis.

I.2.3 Phéromone : dépôt et évaporation

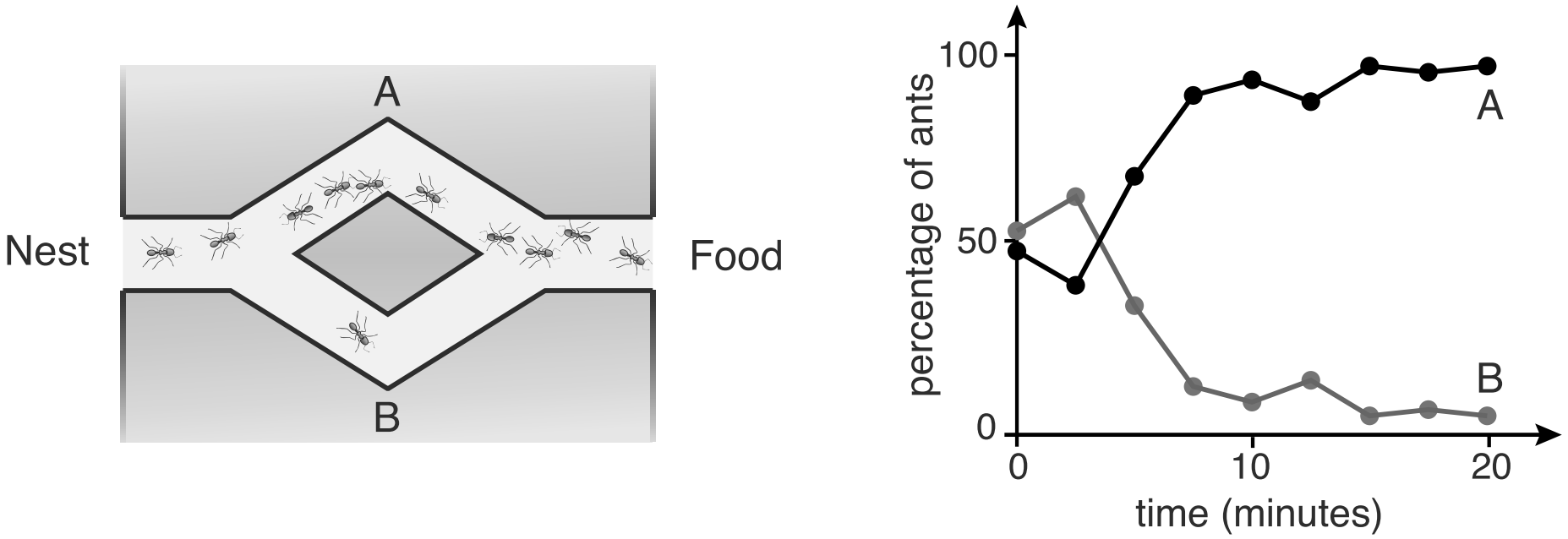


FIGURE I.1 – Dépôt et évaporation de Phéromone par les fourmis : la voie A saturée, la voie B quasi nulle

➙ On constate : moins une voie est empruntée, moins il y aura de phéromone dont la quantité tend vers 0 avec l'évaporation (dans le temps).

☞ Voir support de ce cours pour plus sur les détails du **rétrocontrôle positif**.

I.3 Stigmergie, Intelligence émergente

• Les métaphores biologiques et le modèle *stigmergique* ont été appliqués à la construction de nombreux systèmes complexes.

☞ Tous ces systèmes utilisent des agents indépendants qui interagissent dans un environnement commun pour atteindre des propriétés globales.

• Le modèle de communication indirecte rend ces systèmes **adaptatifs**, **décentralisée** et **émergentes**.

• La **stigmergie** se caractérise par les éléments suivants :

- Environnent "Open-hostile"
- Pas de contrôle Centralisé
- Pouvoir de calcul limité (convient à l'embarqué)
- Communication élémentaire, peu (ou pas) de mémoire.

Exemple : un réseau de communication :

- Adaptatif et décentralisé : tolérance des/aux pannes
- La décentralisation des moyens met le réseau davantage à l'abri d'attaques.
- Ces caractéristiques sont souhaitables pour la survie et la sécurité du système.

- Comportement "**émergent**" du système :
  - *Pros* :
    - **un comportement global** apparaît à partir des comportements des membres.
    - *Être émergent* (et nombreux) permet d'éviter d'être l'unique élément problématique en cas de panne (comme dans les systèmes traditionnels).
  - *Cons :*
    - les systèmes stigmergiques peuvent poser des problèmes de sécurité dès qu'ils fonctionnent dans un environnement ouvert et hostile.

➙ L'absence de contrôle centralisé pose parfois des problèmes :
on ne peut par exemple pas utiliser un mécanisme cryptographique centrale (parfois nécessaire)

➙ Seuls les membres possèdent une puissance de calcul limitée et individuelle

I.3.1 Optimisation par essaim de particules (PSO) : le principe

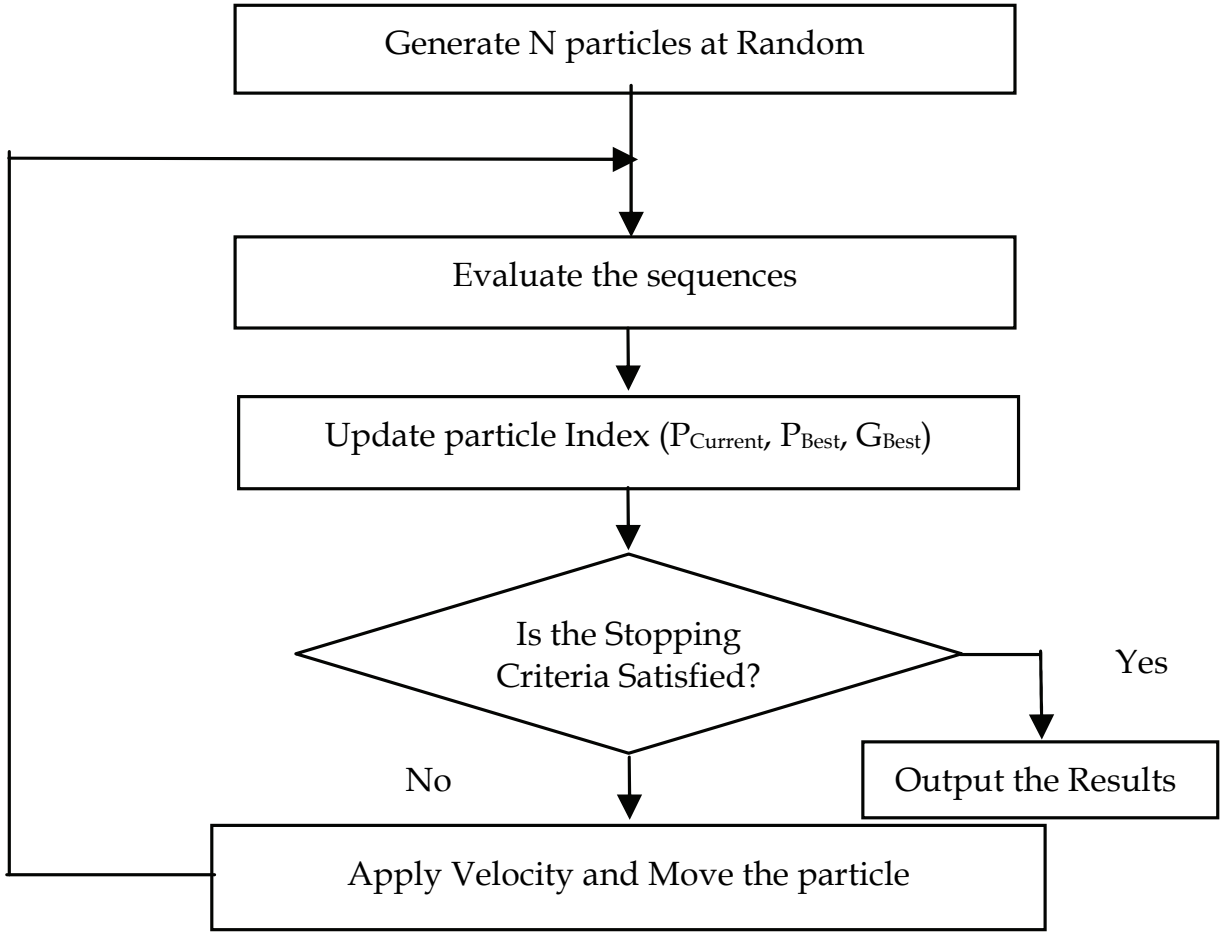


FIGURE I.2 – Algorithme de base de PSO : **pbest** : le meilleur local, **gbest** : le meilleur global

Suivant ce principe général :

Après chaque *"déplacement"*, les caractéristiques de la population (les particules) sont mises à jour via un calcul, P ; Ex. par un code de "update" PSO suivant :

```
for each particle p
  for each coordinate c          ← (p.c : la coordonnée c de la particule p)
    p.c = p.c + 2 * rand() * (pbest.c - p.c) + 2 * rand() * (gbest.c - p.c)
```

pbest : le meilleur local jusque là,

gbest : le meilleur global jusque là. ☞ "centralisation" ?

➙ P.ex : la fourmi la plus rapide au dernier voyage vs. la fourmi la plus rapide des tous les voyages.

☞ ceci n'est qu'un exemple de calcul.
On ne l'utilisera pas forcément dans le BE !

I.3.2 Quelques exemples d'application

- le problème symétrique et asymétrique de voyageur de commerce.
- le problème d'ordonnancement séquentiel.
- le problème d'affectation quadratique.
- le problème de tournées des véhicules.
- le problème d'établissement d'horaires (emploi du temps)
- les problèmes de coloration de graphes.
- les problèmes de partitionnement.
- les réseaux de télécommunications.
- le routage de circuits, Implantations parallèles, ...
- &c.

Applications particulières en Réseaux et Télécoms présentés dans la littérature :

- Network Routing Problems [DiCaro98]
- Peer-to-Peer network framework [Montresor01]
- Distributed Intrusion and Response Systems [Fenet01]
- Terrain Coverage [Koenig01],
- Modified PSO algorithm for solving planar graph coloring problem [Guangzha07]
- &c.

☞ Difficulté : adapter les données du problème au concept de "colonie de fourmis" !

☞ Voir cours 2 : exemple de réseau (AntNet).

I.4 Modélisation de la colonie de fourmis comme un système multi-agents

- Cette section explique quelques éléments du BE.
- On considère la recherche dans les graphes et le comportement alimentaire des fourmis

➙ Remarquer la similitude entre ces deux problèmes.

I.4.1 Environnement : un graphe de villes

- Les noeuds d'un graphe comme des lieux où les fourmis pourraient s'arrêter lors d'un "voyage" ;
  → On appellera ces noeuds "villes".
  → Les arêtes du graphe représentent les routes reliant ces villes.

- Cet **environnement** virtuel sera peuplé d'**agents** représentant les fourmis
  → *individus / population*

- Les fourmis essayent de trouver le meilleur itinéraire (le plus court chemin = PCC) entre 2 points situés dans cet environnement.

- L'idée principale : **mettre simplement des fourmis "en marche"** :
  - Observer/mesurer l'intensité de la phéromone déposée sur les arêtes du graphe dans le temps,
  - Appliquer des actions et .... laisser **le PCC** émerger.

I.4.2 Les agents en IA

• Un agent est une entité autonome qui interagit dans un (son) environnement.

• Nos agents (ici) ont seulement une perception partielle mais dynamique de leur environnement lequel peut être scruté (et modifié) par leurs capacités sensorielles

➙ Perception (via des capteurs) du dépôt / la détection de **phéromone**.

• Les agents peuvent effectuer des actions fondées sur la perception locale et sur leur représentation interne de (et dans) l'environnement.

• Les actions peuvent affecter l'environnement :

➛ avec effets sur l'agent lui-même ainsi que sur d'autres agents.

• On considère la description du comportement alimentaire des fourmis

➛ on cherche la **bouffe** (avec le concept d'agent à l'esprit).

☞ Dans un processus d'apprentissage, les agents peuvent être **récompensés** / **punis** ; Pas ici.

- Les agents sont capables de :
  - "marcher" (avancer) sur un **graphe**, allant d'une **ville** vers une autre ;
  - "prendre" des aliments quand ils arrivent à la **source** de nourriture ;
  - "laisser" cette nourriture lorsqu'ils reviennent au **nid**.
  - "déposer" de la phéromone sur une **route** (une arête) lors d'un voyage ;
  - "décider"/"choisir"quel sera le prochain chemin à suivre en fonction de l'intensité de phéromone sur les arêtes possibles depuis un noeud.

Le principe de l'algorithme :

- Soit une route qui lie deux villes v_1 et v_2 :
  - On fera "avancer" les fourmis pas à pas
    - ➢ un pas d'itération fera avancer chaque fourmi d'un pas
  - Suite à ce "pas", on modifie l'état de l'agent (fourmi).

➢ Voir le tableau de la page suivante pour les actions.

I.4.3 Le comportement d'un agent

• Le tableau suivant décrit les actions d'un agent en fonction de chaque Etat.

Lieu	**Actions**
Dans une ville (noeud)	Choisir l'arête suivante, déposer de la phéromone
Sur une route (arête)	Avancer un pas de plus (on ne recule jamais)
Au nid, transportant de la nourriture	Laissez les aliments
A la source de nourriture	Prendre de l'aliment et retourner au nid

• Un agent **sur un itinéraire** avance d'un pas (une étape) à chaque tour jusqu'à ce qu'il arrive dans une ville.

• Une fois **dans la ville**, il choisit un itinéraire en fonction de l'intensité de la phéromone sur les chemins disponibles.

• Si l'agent trouve la source de nourriture ; il prend la nourriture puis démarre **le voyage de retour** en suivant sa propre piste de phéromone ;

• De retour **dans le nid**, l'agent laisse la nourriture puis recommence le processus à nouveau.

☞ Notons qu'à chaque départ, les agents suivent les pistes ayant un maximum de Phéromone

➛ Mais au départ du nid, ils n'ont aucune mémoire de leur "propre" piste .

• Les biologistes pensent que la phéromone / la nourriture représente un **stimulus** (motivation, récompense).

☞ Les actions des agents nécessitent des informations locales (= une mémoire à court terme) permettant à l'agent de **reconnaître son propre chemin vers le nid**.

➛ Les classes Python / C++ nous aideront ! (hope)

• A propos du "retour par leur propre piste".

➛ Évite à la fourmi de s'enfermer sur la même arête par ex. A-B (en faisant sans cesse A-B-A-B-A-...)

I.5 La mise en oeuvre

- On simulera un environnement multi-agents en utilisant un système de tourniquet,
  - Comme un jeu où **à chaque tour**, tous les joueurs font un seul mouvement.
  - Chaque agent produira seulement l'une des actions élémentaires décrites ci-dessus à chaque tour.

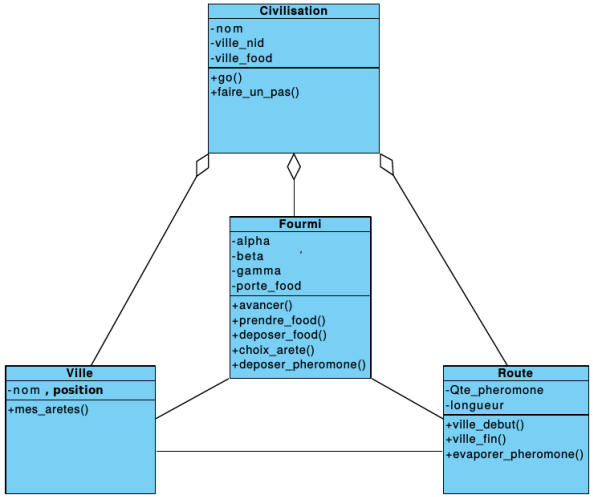


FIGURE I.3 – Une proposition possible (à compléter)

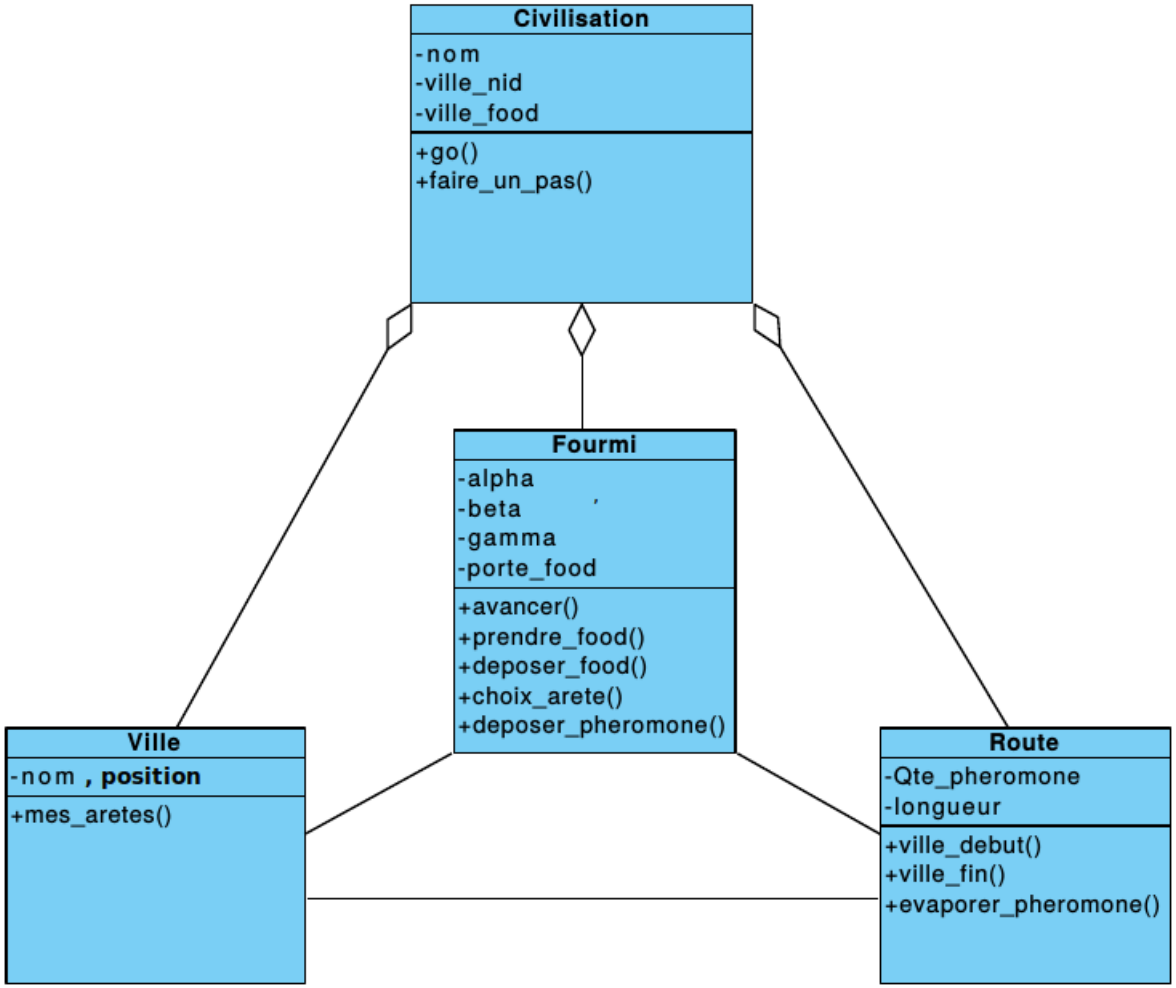


FIGURE I.4 – Une proposition possible (version de base, à compléter)

Algorithme de principe :

```
Init :
    créer le graphe (villes, arêtes) et N fourmis

faire_un_pas() :
    Si dans une ville : choisir une arête et l'emprunter
    Si sur une arête : avancer d'une unité (k mètres !)
    Si au Food : prend nouriture puis emprunter une arête de retour
    Si au Nid :
        Si porteur de nouriture alors la déposer
        Choisir une arête et l'emprunter


main() :
    Répéter :
        Pour chaque fourmi F :
            F.faire_un_pas()
            MAJ phéromone
```

- Les éléments de l'environnement sont modélisé à l'aide des classes.
  - la classe **Ant** (fourmi) représente les agents,
  - la classe **Route** représente les arêtes du graphe,
  - la **Ville** représentent les noeuds et
  - la **Civilisation** représente l'environnement.
- On peut représenter le "terrain" (l'environnement) par un échiquier.
- La *ville* (noeud) :
  - On connaît sa position (X, Y) dans l'environnement.
  - Nécessaire pour calculer la distance entre deux villes.
- L'interface graphique de l'outil est traité à part.
- Seuls les aspects essentiels seront abordés ci-après.

I.5.1 La classe Route

- Une route a besoin de "pointer" vers les deux villes qu'elle connecte.
- Une autre propriété est la longueur de la route (arête).

➙ Plus une route est longue, plus l'agent a besoin de tours (des *pas*)

- L'intensité de phéromone sur une arête est représentée dans la classe Route.
- Un point important est la volatilité de la phéromone (à simuler).

➙ La classe Route aura besoin d'une méthode pour simuler l'**évaporation**.

```
class Route :
   def __init__(self, ...) :
      self.__longueur = ...;          # float  : Longueur de la route
      self.__pheromone = ...;          # float  : Intensite de la phéromone sur la route
      self.__premiere_ville= ...;     # Villes connectées à cette route
      self.__seconde_vile= ...;

   def evaporer_Pheromone(self, ...) :     # Simulation de l'évaporation de la phéromone
      ....
   # constructeurs, interface et méthodes auxiliaires etc
```

I.5.2 La classe Ant (agent, fourmi)

• Chaque instance de la classe **Ant** doit représenter un agent individuel avec des caractéristiques singulières.

• Une caractéristique la plus importante d'une fourmi :

elle a une tendance individuelle et imprévisible (un paramètre random) à choisir un itinéraire (arête) parmi ceux disponibles.

• Le niveau de phéromone sur une route : un nombre réel.

• L'agent utilisera une méthode pour évaluer sa tendance à choisir un itinéraire en fonction de l'intensité de la phéromone.

→ Cette fonction évaluera la possibilité de choix de toutes les arrêts possibles ;

→ Puis choisira la **meilleure** parmi ces possibilités.

I.5.2.1 Dépôt de phéromone par un agent

Un exemple : une (assez bonne) variabilité du comportement des agents peut être exprimée par une fonction sinusoïdale avec (au moins) trois coefficients α, β, γ,

$$PL_{t+1} = \alpha * sin(\beta * PL_t + \gamma) \qquad PL : \textit{Pheromon Level} \text{ sur la route}$$

α, β et γ seront des propriétés de la classe **Ant** :

➛ ce sont des réels aléatoires choisis en général dans l'intervalle $[-5 \cdots +5]$.

☞ Ceci n'est qu'un exemple ! Voir plus loin.
➛ α, β et γ : voir plus loin l'algorithme Génétique.

- D'autres paramètres possibles (pour les fourmis artificielles)
  - moduler le taux de phéromone "perceptible" sur l'arête,
  - tenir compte de la longueur de l'arête ?
  - tenir compte de l'"importance" de l'arête
  - Etc.
- Ces propriétés permettront d'avoir des individus différentes dans la population

➙ Elles sont également nécessaires pour les algorithmes génétiques qui seront abordés plus loin.

☞ Parfois, une "simulation des fourmis" utilise une caractéristique inexistante chez une fourmi !

Mais sans exagération !

```
class Ant :
   def __init__(self, ...) :
      self.__alphaa = ...            # La sensibilité phéromonale de la fourmi
      self.__beta = ...
      self.__ gamma = ...
      self.__porte_nourriture= ...   # Trasporte de la nouriture ou non


def getTendance(self, PheroLevel : float) :     # tendance à choisir une  route
def prendre_nourriture(self, ...) :               # Prendre la nouriture dans la source de nouriture
def  laisser_nourriture(self, ...) :            # Laisse la nouriture (dans le nid)
def deposer_pheromone(self, ...) :           # Augmenter le niveau de la phéromone de la route
def marcher(self, ...) :                        # Avancer une étape plus

# constructeurs, interface et méthodes auxiliaires etc
```

I.5.3 La classe Civilisation (Environnement)

• La classe *Civilisation* contrôle l'ensemble de l'environnement :

➙ le processus d'évolution et de simulation.

• Deux noeuds du graphe désignent le **nid** et la **source** de nourriture.

• Au début de la simulation, on crée un nombre aléatoire d'agents dans le nid.

• A chaque tour, les agents effectuent des actions en fonction de leur position actuelle.

• Pendant l'exécution de la simulation, les itinéraires les plus utilisés verront leur niveau de phéromone augmenter

➙ après quelques temps, une solution émergera du comportement collectif de ces fourmis virtuelles.

• Si une interface graphique (cf. Qt/Tk/etc.) est créée, un dessin du graphe peut être créé.

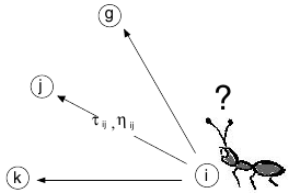
```
class Civilisation :
   def __init__(self, ...) :
      self.__src_nourriture = ...      #  La ville source de nouriture de la  Civilisation
      self.__nid = ...                 #  Le nid de la Civilisation
      self.__routes = ...              #  toutes les routes de l'Environement
      self.__villes = ...              #  toutes les villes de l'Environement
      self.__fourmis = ...             #  toutes les fourmis dans l'Environement
      self.__selectionNaturelle = ...  # les tours restants avant la prochaine sélection  (pour l'algorithme géné
 tique)

    def tourSuivant(self, ...):      #  Effectue un tour de la   simulation
      pass

    # constructeurs, interface et méthodes auxiliaires etc
```

I.6 Les règles à préférer / utiliser !

• On peut traduire les règles applicables au problème TSP avec l'optimisation "colonie de fourmis" (ACO) de différentes manières.



Règle du choix d'arête :

• Une fourmi décide de la prochaine ville en fonction du taux τ_{ij} de phéromone (sur l'arête $i-j$) et de l'heuristique η_{ij} associée à l'arête $i-j$ (ici, la ville voisine j non encore visitée). η_{ij} est souvent l'inverse de la distance $i-j$.

• Soit la fourmi k dans la ville r ; s la ville voisine de r non encore visitée par k.

➛ Ces infos sont obtenues d'une mémoire locale M_k de la fourmi k.

La fourmi k choisit d'aller à la ville s par ce calcul :

$$s = \begin{cases} \underset{s \notin M_k}{\operatorname{argmax}} \left\{ [\tau(r,s)].[\eta(r,s)]^{\beta} \right\} & Si\ q \leq q_0 \\ \underset{s \notin M_k}{\operatorname{argmax}} \left\{ P_k(r,s) \right\} & Sinon \end{cases}$$

Détails (voir $P_k(r,s)$)

∘ $\tau(r,s)$ est la quantité de phéromone sur l'arête (r,s)

∘ $\eta(r,s)$ est une fonction heuristique associée à l'arête $(r-s)$ qui est (souvent) l'inverse de la distance entre les villes r et s,

$$s = \begin{cases} \underset{s \notin M_k}{\text{argmax}} \left\{ [\tau(r,s)].[\eta(r,s)]^\beta \right\} & Si\ q \leq q_0 \\ \underset{s \notin M_k}{\text{argmax}} \left\{ P_k(r,s) \right\} & Sinon \end{cases}$$

∘ β est un paramètre qui pondère ici l'importance relative de la phéromone ainsi que la proximité $(r \leftrightarrow s)$,

∘ q est un réel aléatoire (paramètre du système) choisi uniformément sur l'intervalle $[0,1]$,

∘ $0 \leq q_0 \leq 1$ est un paramètre du système,

∘ $P_k(r,s)$ est une probabilité de choix du prochain voisin s selon la distribution ci-contre (qui favorise les arêtes plus courtes avec un niveau de phéromone plus élevé)

$$P_k(r,s) = \begin{cases} \dfrac{[\tau(r,s)]^\alpha.[\eta(r,s)]^\beta}{\sum_{s \notin M_k} [\tau(r,s)]^\alpha.[\eta(r,s)]^\beta} & Si\ s \notin M_k,\ s\ voisine\ de\ r \\ 0 & Sinon \end{cases}$$

☞ $P_k(r,s)$ est la probabilité de choisir d'aller de la ville r à s.

∘ Comme pour β, le paramètre α pondère ici le taux de phéromone.

☞ On peut éviter de calculer $P_k(r,s)$: choisir la plus grande valeur du numérateur (sauf si on a explicitement besoin de $P_k(r,s)$, P. Ex. $\{s | Seuil_1 \leq P_k(r,s) \leq Seuil_2\}$ où $Seuil_i$ sont des paramètres du système).

☞ On peut simplifier et remplacer le calcul de $P_k(r,s)$ par un tirage uniforme de s lorsque $q \leq q_0$ (ci-haut).

☞ Voir ci-après : une (3e) règle plus simple est proposée pour l'implantation .

I.6.1 Mise à jour de la phéromone

- MAJ lorsque les fourmis auront chacune construit leur trajet :

1. D'abord **l'évaporation** : une baisse générale de la quantité sur **toutes les arêtes** du graphe par un facteur constant,
2. Puis **augmentation** : on met à jour la phéromone sur les arêtes empruntées.

I.6.1.1 La règle d'évaporation

- Pour l'étape (itération) $n+1$, on se donne la règle d'évaporation suivante

$$\tau_{ij}^{n+1} \leftarrow (1-\rho)\tau_{ij}^{n} \qquad \forall (i,j) \in \ graphe$$

Où $0 \leq \rho \leq 1$ est le taux d'évaporation.

- ρ permet d'éviter de stocker une quantité illimitée de phéromone ;

Il permet également de minimiser l'effet des "mauvais choix antérieurs".

- Ces paramètres doivent permettre une évaporation *exponentielle* pendant les itérations si une arête n'est pas empruntée.

☞ Un calcul simple permet de remarquer ici que si

$$\tau_{ij}^{n+1} \leftarrow (1-\rho)\tau_{ij}^{n}$$

alors on peut généraliser pour l'itération (étape) m :

$$\tau_{ij}^{m} = \tau_{ij}^{0} \,.\, e^{m.ln(1-\rho)}$$ (i.e. l'exponentielle recherchée, sans jamais tomber à 0)

I.6.1.2 La règle d'augmentation

Une règle souvent utilisée à la suite d'une évaporation systématique à chaque étape.

- La quantité de phéromone sur les arêtes empruntées augmente via :

$$\tau_{ij}^{n+1} \leftarrow \tau_{ij}^{n} + \Delta\tau_{ij}^{k} \qquad \forall (i,j) \in T^k$$

T^k : trajet de la fourmi k voir ci-après

Où $\Delta\tau_{ij}^{k}$ est la quantité de phéromone que la fourmi k déposera sur l'arête empruntée.

- Cette quantité est définie par :

$$\Delta\tau_{ij}^{k} = \begin{cases} \frac{1}{C^k} & Si\ l'arete\ (i,j) \in T^k : \\ 0 & Sinon \end{cases}$$

Où C^k est la longueur de la tournée T^k effectuée par la fourmi k ;

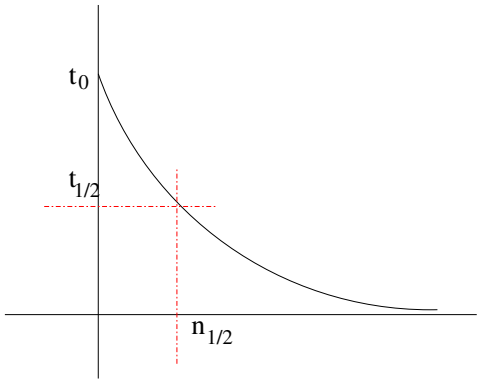
➙ C^k est simplement la somme des arêtes empruntées par la fourmi k

Remarques : vous pouvez définir une (autre) règle vous-même.

• Par exemple : décidez du temps (ou du nombre d'itérations qu'on appelle $n_{\frac{1}{2}}$) où vous voulez que la quantité de la phéromone se divise par deux ("tombe" à $\frac{1}{2}$) :

$$\frac{\tau_{ij}^{n_{\frac{1}{2}}}}{\tau_{ij}^{0}} = (1-\rho)^{n_{\frac{1}{2}}} = \frac{1}{2}$$

Où $0 \leq \rho \leq 1$ est le taux d'évaporation.

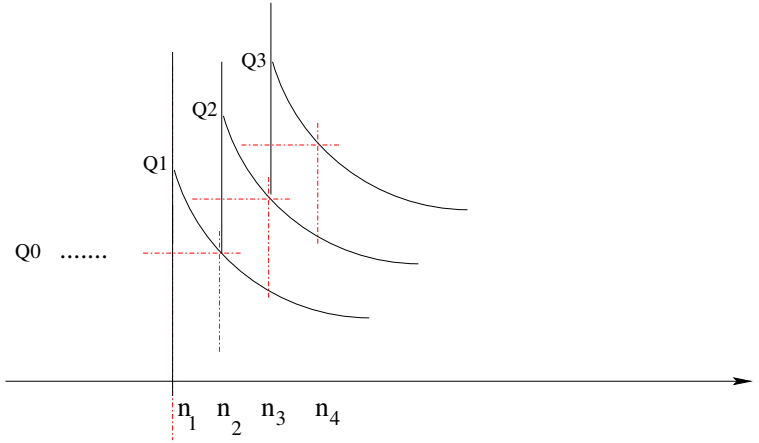


D'où le nombre d'itérations pour atteindre cette quantité :

$$n_{\frac{1}{2}} = \frac{log(\frac{1}{2})}{log(1-\rho)}$$

• A l'étape $n+1$, on aura : $\tau_{ij}^{n+1} \leftarrow \tau_{ij}^{n} + \varepsilon\ I_{ij}^{n}$
où $I_{ij}^{n} = 1$ si la fourmi k est sur l'arête ij, 0 sinon.

☞ ici, τ_{ij}^{n+1} est ce qui reste après l'évaporation.



• Il vous reste à définir
$\varepsilon = (Q_{n+1} - Q_n)$ de la figure ?

I.6.2 Initialisation des paramètres

- Au départ, on initialise les paramètres de la manière suivante :

$$\forall (i,j)\ arete\ du\ graphe,\ \tau_{ij} = \tau_0 = \frac{m}{C^{nn}}$$

Où m est le nombre de fourmis et C^{nn} est la longueur de la tournée réalisée par l'heuristique "les plus proches voisins" (en empruntant toujours la vile la plus proche).

☞ On peut utiliser toute autre règle qui permet de construire raisonnablement un tour optimisé (même avec un optimum local comme le trajet où on choisit tjs la ville la plus proche).

☞ **N.B. faire attention à ceci dans vos choix initiaux :**

◦ Si la quantité initiale de phéromone τ_0 est trop basse, les résultats seront vite biaisés par la 1ère tournée générée qui n'est en général pas une bonne solution.

◦ Si τ_0 est trop élevée, on perdra beaucoup d'itérations jusqu'à l'évaporation des phéromones donnant des valeurs "raisonnables".

I.6.3 Remarques, critiques et propositions

- Le système défini ci-dessus peut donner de bons résultats,
  - Mais un nombre aléatoire d'agents ayant des caractéristiques aléatoires ne peut pas toujours résoudre un problème donné (cf. convergence ?).
  - Une population d'agents capable de trouver le PCC dans un graphe peut ne pas être en mesure de trouver une solution dans un environnement complètement différent.

D'où les questions :

- Comment trouver les meilleurs agents et combien en faut-il pour un problème donné ?
- Comment équilibrer les agents ayant des caractéristiques différentes pour composer une bonne population ?

- **Amélioration** : vers les algorithmes génétiques : optimisation

I.7 Optimisation de la colonie de fourmis

• **Les algorithmes génétiques** : une technique d'optimisation adaptative basée sur le processus naturel d'évolution.

• Principe darwinien de la "survie du plus apte" au fil des générations consécutives.

• Les individus les plus efficaces propagent leurs caractéristiques par les gènes qui seront re-combinés dans la création de nouveaux individus.

• La population évolue en s'adaptant à leur environnement à travers la mutation et la sélection naturelle.

• Les agents sont créés au début avec des **caractéristiques aléatoires**.

Remarques sur les caractéristiques :

- ils peuvent ne pas être adaptées (à un environnement spécifique)
- cela pourrait prendre trop de temps pour obtenir un bon résultat
- ils pourraient même ne pas trouver une solution.

Les opérateurs évolutionnistes (génétiques) :

- **Sélection** : on retient les meilleurs acteurs (on élimine les autres),
- **Mutation** : souvent pour répondre à une nécessité (sinon à une perte d'individu),
- **Croisement** : progéniture prometteuse.

• Les performances du système peuvent être grandement améliorées en appliquant ces opérateurs à la population de fourmis.

• Deux types majeurs d'individus :

- ◦ les meilleurs travailleurs (apportent beaucoup, en quantité, si la quantité "transportable" varie selon les fourmis ?)
- ◦ les meilleurs explorateurs (les plus rapides ou les moins répétitifs).
- ◦ fourmis de force différente, fourmis soldats (présence d'ennemies ?)
- ◦ ... ajouter vos critères (fourmi meilleure en xx , fourmi qui perd la bouffe !, ...)

I.8 Algorithme GI : le principe

Algorithme Génétique : le Principe :

Construire une population initiale P de taille n
Évaluer P
Répéter jusqu'à l'arrêt :

- $P1 \leftarrow$ **Sélectionner** une partie de la population P (facteur $k1$)
- $P2 \leftarrow$ **Reproduire** (croisement) les individus de $P1$ (facteur $k2$)
- $P3 \leftarrow$ **Mutation** de la descendance de cette reproduction ($P2$) ($k3 \leq k2$)
- $P4 \leftarrow$ **Évaluer** chaque individu de $P1 \cup P2 \cup P3$ (et retenir n best individus)
- $P \leftarrow P4$

La méta-Heuristique AG est appliqués à :

- TSP, RNs, ML (Clustering), ...
- et à tout autre problème dont l'espace est de recherche est grand

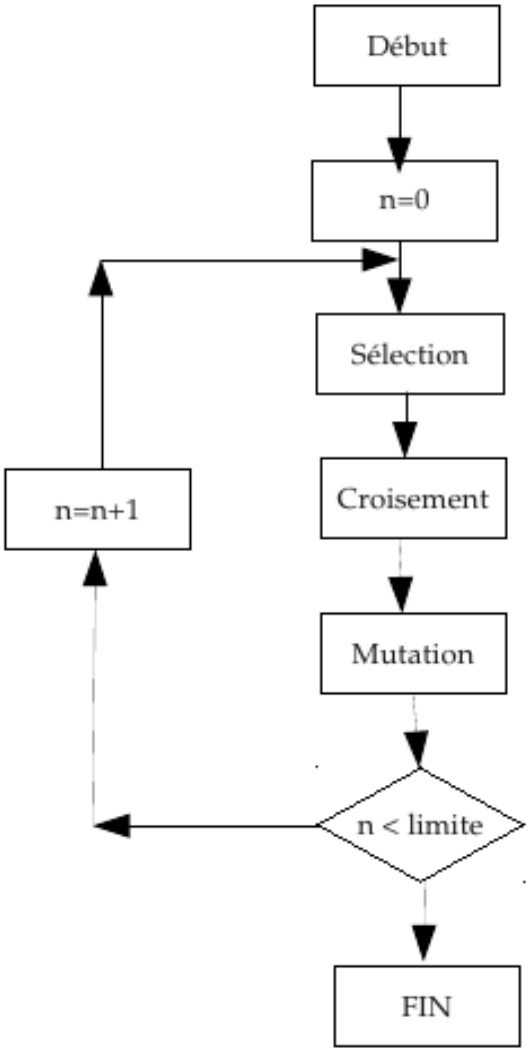
Opérateurs Évolutionnistes :

Codage et représentation : en base 2 (0/1) ou base 10 (0..9), arborescence, ... N.B. : les réels reçoivent de préférence une représentation en entier.

Sélection (reproduction) : comme la sélection naturelle, on cherche à choisir les populations prometteuses : celles qui minimisent par exemple une erreur moindre carré ou une distance p/r à une fonction **objectif** (ou une référence).
➙ décision selon une valeur "*fitness* " (e.g. Erreur Moindre Carrée)
➙ et / ou selon l'intérêt d'une population ± prometteuse (vs. l'objectif')

Croisement : échange de contenus génétiques entre couples. Souvent échange des moitiés des chaînes de bits. Dans une représentation arborescente, on peut permuter des sous-arbres dans un couple de parents et obtenir ainsi deux descendants.

Mutation : cherche à éviter les extrema locaux de la "fitness" (suite aux Sélections / Croisements). Remplacement d'un bit (aléatoire), d'un caractère, d'une sous structure, etc. ➙ Permet de rechercher dans des régions diverses de l'espace de recherche. Dans une représentation arborescente, on peut *muter* (changer) soit des nœuds terminaux (variables ou constantes dans le cas des fonctions) ou des sous arbres (ou sous expressions dans le cas de populations de fonctions).



Exemple : Voyageur de commerce (TSP).

I.8.1 Sélection

• **La concurrence (compétition)** : les individus retenus dans cet environnement peuvent

  ◦ soit recueillir une grande quantité de nourriture (les *travailleurs*)

  ◦ soit trouver des voies alternatives vers la source de nourriture (*les explorateurs*).

• La sélection naturelle leur permet de "produire" une descendance et de diffuser leurs caractéristiques.

➙ De même que pour le processus naturel, l'évolution se produit parce que la nouvelle génération des individus sera parfois "meilleure" que les parents.

• **Extinction** : les agents qui se perdent (dans l'envt.) ne peuvent pas/plus aider.

➙ P. Ex. un agent qui voyage entre 2 noeuds en utilisant toujours la même arête est inutile à la colonie. De même, il sera inutile s'il continue de tourner en rond.

➙ **Ces individus seront éliminées de la population** (ne seront pas sélectionnés).

I.8.2 Progéniture (croisement)

• **Croisement** *(crossover)* : les descendances sont créées sur la base des caractéristiques (pour nous, α, β, γ) des individus qui **réussissent** .

• Comme il y a deux types d'individus nécessaires à la colonie, deux individus seront créés à chaque cycle évolutif.

◦ L'un d'eux sera descendant des deux travailleurs les plus réussis, et

◦ L'autre sera descendant des deux meilleurs explorateurs.

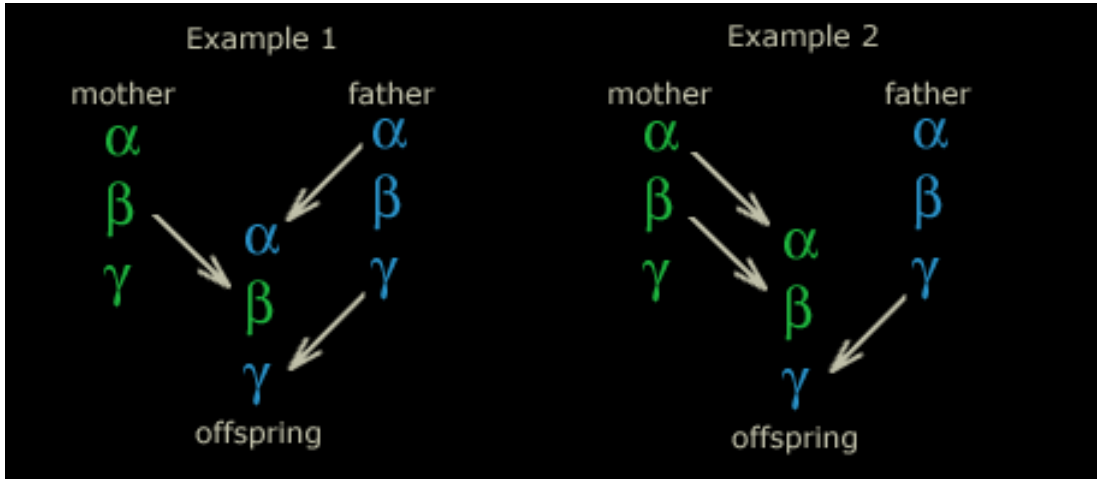


FIGURE I.5 – deux exemples de descendants créé par croisement

• Les gènes représentant les caractéristiques des parents seront combinés **au hasard** pour composer un nouveau chromosome qui donnera un nouvel individu.

• Cette combinaison est inspirée du processus biologique appelé **croisement**.

• Chaque caractéristique de l'individu viendra de l'un des parents choisi au hasard.

• La figure montre deux exemples possibles de descendants créé avec des combinaisons de croisement :

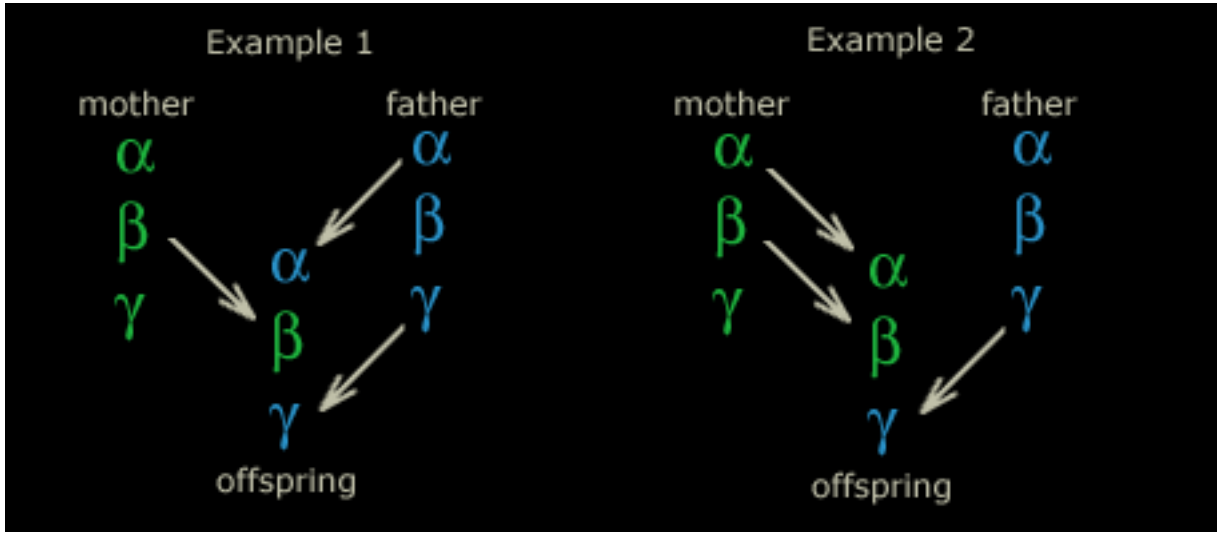


FIGURE I.6 – deux exemples de descendants créé par croisement

I.8.3 Mutation

• La mutation peut maintenir la diversité au sein de la population.

• La **mutation** va changer l'une des caractéristiques de l'individu au hasard.

☞ Après un croisement, il y a une faible probabilité qu'une mutation se produise.

I.8.4 Migration

La **migration** ajoutera un nouvel individu complètement aléatoire à la population.

• L'effet est similaire à la mutation, car elle va accroître la diversité au sein de l'environnement.

I.9 Mise en oeuvre des opérateurs génétiques dans l'exemple

I.9.1 Sélection

- Le **travailleur** le plus réussi est celui qui a recueilli davantage de nourriture.
- Un compteur sera ajoutée à la classe Ant et sera incrémenté à chaque fois que l'agent atteint le nid apportant de la nourriture.
- A chaque processus de sélection, l'agent avec la valeur supérieure de ce compteur sera considéré comme la plus réussi.
- Les agents comptent combien de fois ils ont été sur la même route (arête).
  - Les valeurs faibles de ce compteur indiquent les **explorateurs** avec succès ;
  - Les valeurs plus élevées : des agents qui se sont perdus dans l'environnement.

I.9.2 Progéniture, Croisement

• **Croisement** *(crossover, recouvrement)* : un nouveau constructeur sera ajouté à la classe Ant qui aura pour paramètres deux références (pointeurs) sur *Ant*.

➙ La nouvelle instance sera créée en combinant les caractéristiques des parents.

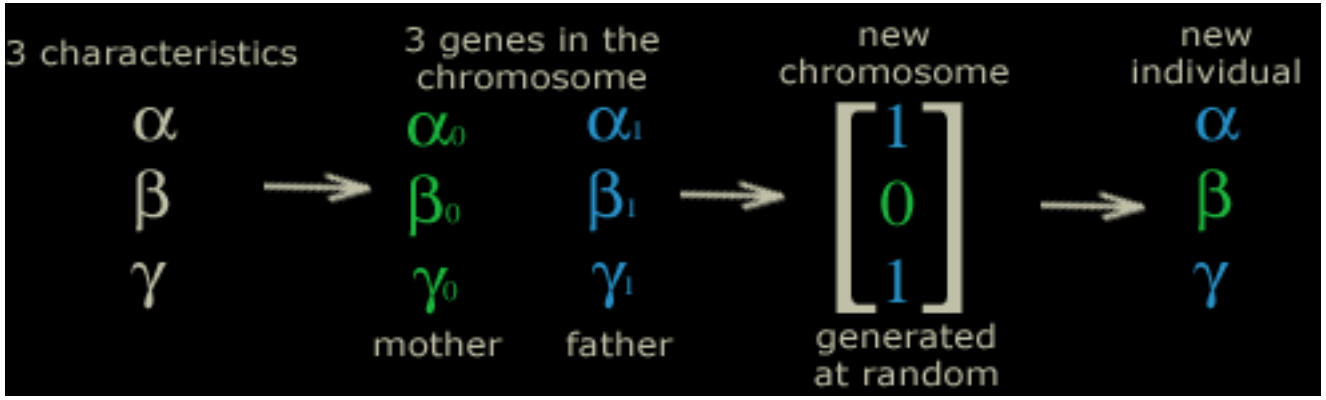


FIGURE I.7 – Implantation de CrossOver (sur la base de α, β, γ)

• Un chromosome est composé de 3 gènes représentant chacun une caractéristique.

• Le nouveau chromosome peut être rempli par des 0 ou des 1 avec la même probabilité.

• Pour chaque gène , 0 = la caractéristique sera héritée de la mère et 1 par le père.

I.9.3 Mutation

• La **mutation** est une autre méthode qui sera utilisée pour effectuer une modification.

• Elle peut être appelée après le croisement (CrossOver) mais c'est un évènement avec une faible probabilité de se produire.

I.9.4 Migration

• Un individu totalement nouveau peut être créé en attribuant une valeur aléatoire à α, β et γ.

➙ Peut être mis en oeuvre comme le constructeur par défaut de la classe *Ant*.

I.9.5 La nouvelle classe Ant

• Après l'inclusion de ces nouvelles propriétés et méthodes, la classe Ant devrait ressembler à ci-dessous (tous les attributs ne sont pas présents)

```
from dataclasses import dataclass
@dataclass
class Ant :
    alfa : float                    # La sensibilité phéromonale de la fourmi
   beta : float
   gamma : float
    porte_nourriture : bool          # Trasporte de la nouriture ou non
    qte_nouuriture_collectee : int       # Quantité de nouriture collectée par cette fourmi
    nb_fois_sur_meme_route : int        # voir ci-dessus

    def ..... des fonctions :          # Constructeur par défaut : un individu tout neuf !
   def construction (... ) :          # Constructeur
   def getTendance(PheroLevel)          # Evalue la tendance à choisir une route  (proba)
   def prendre_nourriture(... ) :       # Prendre la nouriture dans la source de nouriture
   def laisser_nourriture(... ) :       # Laisse la nouriture (dans le nid)
   def deposer_pheromone(... ) :        # Augmenter le niveau de la phéromone de la route
   def marcher(... ) :                # Avancer une étape plus
   def mutation(... ) :
   # constructeurs, interface et méthodes auxiliaires etc ...
```

I.10 Remarques

• Comme il n'y a pas de fonction de coût explicite liée au graphe, ce système peut être utilisé dans des applications avec un environnement inconnus et temps réel :

➟ par exemple pour l'exploration robotique dans un environnement dynamique.

• On note qu'aucune connaissance préalable n'est nécessaire ici car l'effet d'une fonction de coût implicite est une conséquence de la longueur des chemins (arêtes) qui ont besoin de plus de temps pour être franchis.

➟ Effet sur un pas de l'algo : toute une arrête ou une longueur l ?

• Les agents (fourmis) ont seulement une mémoire relativement limitée et le système n'est pas très affecté par la complexité du problème.

• Si l'un des agents est accidentellement perdu, le système continuera de travailler et le résultat final ne sera pas affectée.

➟ Cette propriété rend le système robuste .

- La taille de la population est contrôlée par la sélection naturelle.
  - Si peu d'agents perdus → résultats plus rapides
  - Si des agents perdus (environnement vaste), ils seront éliminés.
  - La population va naturellement se développer dans des environnements plus vastes !
- Il n'y a pas de centre de décision central.
- L'information est distribuée entre les agents et l'environnement.
- Le système est également adaptatif.

➙ Si une arête du graphe est soudainement enlevé ou même si une nouvelle est créée, le système peut rapidement se réadapter à l'environnement modifié et une nouvelle route va émerger.

• Les unités peuvent être conçues pour effectuer de simples actions locales avec un faible coût de calcul et de l'intelligence apparaitraient alors comme une conséquence de l'auto-organisation de la colonie d'individus.

• Les algorithmes génétiques pourraient être appliqués pour sélectionner et faire évoluer les meilleurs individus et le joueur pourra apprendre de nouvelles stratégies et ainsi le jeu deviendrait plus intelligent.

• Ce système combine le principe bio-inspiré multi-agents, ceux des algorithmes génétiques et le principe de la *stimergie*.

• Le système est aisément parallélisable.

• L'intelligence émergente peut être utilisée dans les jeux de stratégie temps réel.

I.11 Démo PCC

- PCC (simulation sous Windows, voir support du cours Chap2 pour les détails)

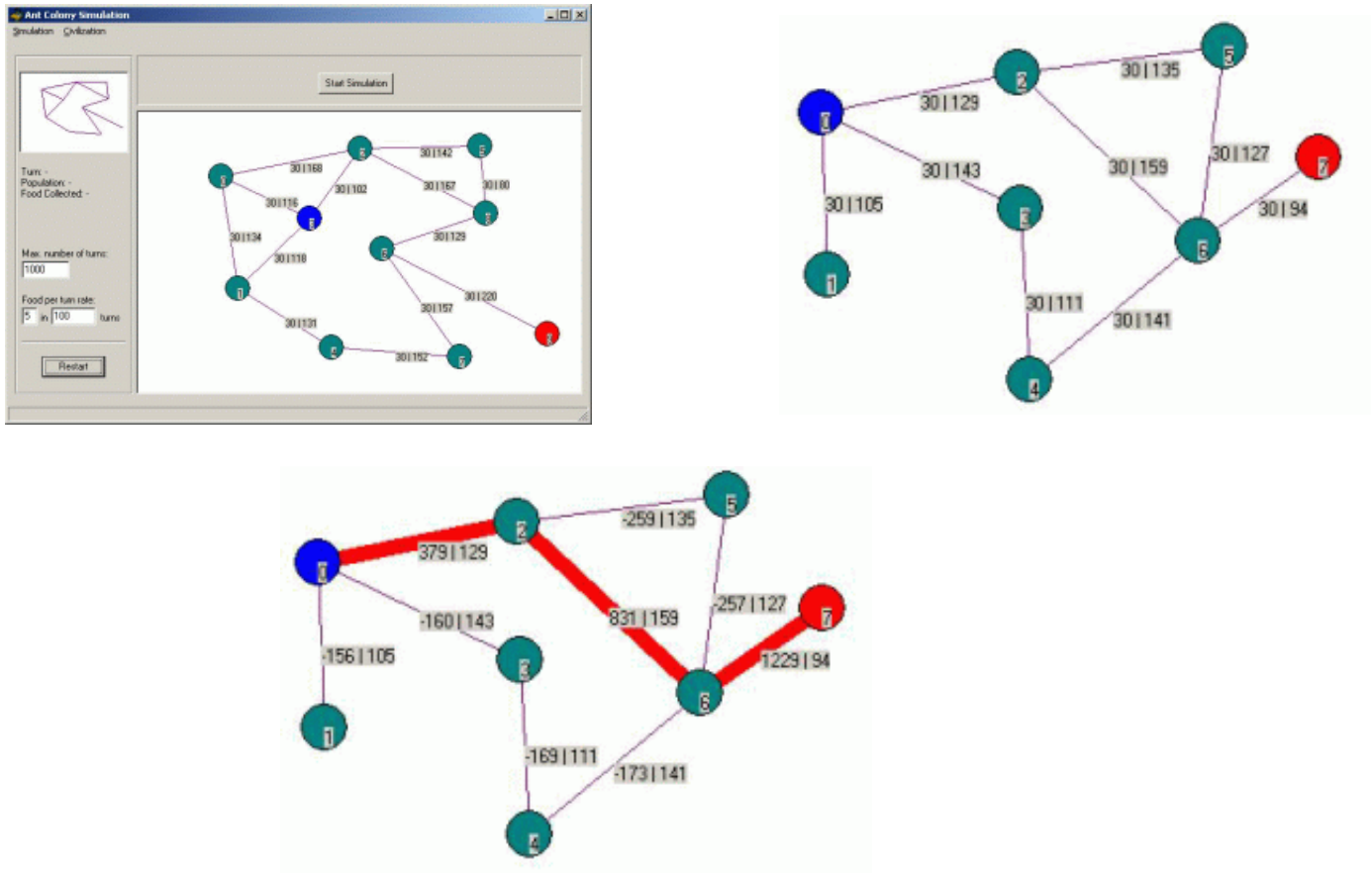


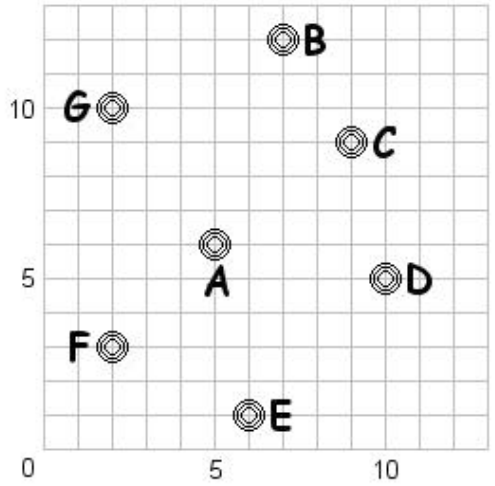
FIGURE I.8 – Meilleure route

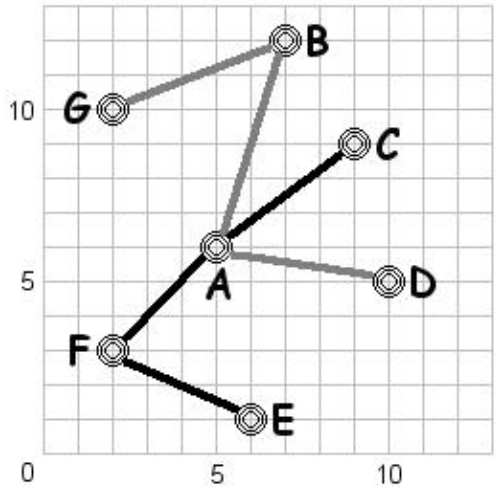
I.12 Sujet 2 : Bus Allocation Problem (BAP)

- Ce problème se compose d'un certain nombre d'arrêts et de lignes de bus.

  ➙ Chaque arrêt de bus est à un endroit précis.

- La figure suivante donne une carte avec 7 arrêts ; la solution est à droite.





- Les lignes de bus seront construites en tenant compte de plusieurs contraintes :
  - distance, vitesse de bus (temps d'attente / fluidité / correspondances)
  - nombre de passagers, capacité des bus, .. Voir plus loin.

• **Une solution à la BAP sera une collection de lignes de bus.**

• Utilisant l'emplacement de chaque arrêt de bus, nous sommes en mesure de calculer la distance euclidienne entre deux arrêts.

➙ Notez qu'il est également possible d'utiliser des routes prédéfinies à la place de la distance euclidienne.

➙ Cela pourrait signifier que la distance entre deux arrêts X à Y est différente de la distance de Y à X.

• Un voyage d'un arrêt (départ) à un autre (destination) est appelé une **transition**. Nous supposons que pour chaque transition possible, le nombre de passagers est connu.

• Un des arrêts de bus sera **l'arrêt principal** (A dans l'exemple). Cette arrêt représentera la **gare centrale**.

I.12.1 Une modélisation

• Dans le modèle suivant, toute ligne de bus devra faire appel à l'arrêt principal.

• L'arrêt principal sera associé à l'ensemble des lignes de bus mais tous les autres arrêts seront affectés à exactement une ligne de bus.

• Les bus circuleront simultanément du première arrêt au dernier et du dernier au premier.

Éléments d'une mise en oeuvre :

- On utilisera :

$D = \{d_{ij}\}$ = distance entre deux arrêts i et j.

$T = \{t_{ij}\}$ = nbr personnes en attente à l'arrêt i ayant l'arrêt j pour destination.

- On note également :

r_m : arrêt principal

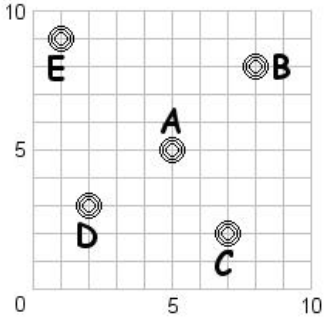
v : vitesse moyenne du bus (m/s)

f : fréquence du bus (temps/h).

c : le temps nécessaire pour changer de bus (correspondance).

- Pour simplifier, on se donne l'instance plus simple suivante avec sa matrice T des transitions

➙ Rappel : T : nbr personnes en attente à l'arrêt i ayant l'arrêt j pour destination.



$$\boldsymbol{T} = \begin{bmatrix} 0 & 7 & 10 & 10 & 7 \\ 5 & 0 & 3 & 3 & 1 \\ 9 & 4 & 0 & 7 & 5 \\ 9 & 3 & 6 & 0 & 4 \\ 7 & 4 & 4 & 3 & 0 \end{bmatrix}$$

• En utilisant une solution, la vitesse moyenne du bus et la matrice D, on peut calculer la matrice suivante :

$U = \{u_{ij}\}$ = Le temps nécessaire pour une personne d'aller de l'arrêt i à j.

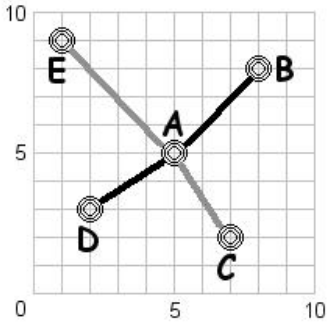
• Pour comparer les solutions, on se donne également la valeur *att* pour "average travel time" qui devra être minimisée (fonction objective) via

$$att = \frac{\sum_{x=1}^{n} \left(\frac{\sum_{y=1}^{n} u_{xy} \cdot t_{xy}}{\sum_{y=1}^{n} t_{yz}} \right)}{n}$$

• Soit la matrice des distances D pour l'instance simplifiée, sa matrice U et une solution à cet exemple (vitesse du bus : 7 m/s) :

$$\boldsymbol{D} = \begin{bmatrix} 0 & 424.26 & 360.56 & 360.56 & 565.69 \\ 424.26 & 0 & 608.28 & 781.03 & 707.11 \\ 360.56 & 608.28 & 0 & 509.90 & 921.95 \\ 360.56 & 781.03 & 509.90 & 0 & 608.28 \\ 565.69 & 707.11 & 921.95 & 608.28 & 0 \end{bmatrix}$$

$$\boldsymbol{U} = \begin{bmatrix} 0 & 112.117 & 132.32 & 112.117 & 132.32 \\ 60.6092 & 0 & 132.32 & 112.117 & 372.32 \\ 51.5079 & 352.117 & 0 & 352.117 & 132.32 \\ 51.5079 & 112.117 & 132.32 & 0 & 372.32 \\ 80.8122 & 352.117 & 132.32 & 352.117 & 0 \end{bmatrix}$$

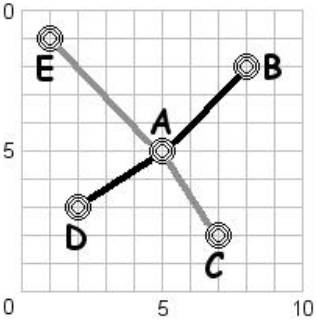


../..

Explications de la matrice U :

$$D = \begin{bmatrix} 0 & 424.26 & 360.56 & 360.56 & 565.69 \\ 424.26 & 0 & 608.28 & 781.03 & 707.11 \\ 360.56 & 608.28 & 0 & 509.90 & 921.95 \\ 360.56 & 781.03 & 509.90 & 0 & 608.28 \\ 565.69 & 707.11 & 921.95 & 608.28 & 0 \end{bmatrix}$$

$$U = \begin{bmatrix} 0 & 112.117 & 132.32 & 112.117 & 132.32 \\ 60.6092 & 0 & 132.32 & 112.117 & 372.32 \\ 51.5079 & 352.117 & 0 & 352.117 & 132.32 \\ 51.5079 & 112.117 & 132.32 & 0 & 372.32 \\ 80.8122 & 352.117 & 132.32 & 352.117 & 0 \end{bmatrix}$$



- u_{21} : (B à A) : B est alloué à la même ligne de bus que A ; et cette entrée est égale à la valeur du d_{21} (distance de B à A) divisé par 7 (vitesse du bus).

- u_{12} : (de A à B) : A est alloué à la même ligne de bus que B ; mais la personne qui attend en A (pour aller à B) doit attendre le bus qui arrive de D avant de pouvoir aller à B : cet entrée est donc est égale à u_{42} (de D à B).

- u_{43} : (D à C) : D n'est pas attribué à la même ligne de bus que C.
  - Un tel voyageur doit d'abord se rendre à A, où il peut prendre le bus de la ligne (E,C) :
    - le trajet de E à A prend 80,81 secondes et D à A prend $51,51$ secondes.
    - → Donc $51,51 + 20$ secondes (ce qui est c) est toujours inférieur à $80,81$ secondes.
  - Quand elle arrivera à C, cela lui aura pris $80.81 + 51.51$ (A à C) secondes.

- u_{54} : (E à D) : E n'est pas attribué à la même ligne que D.

➙ Encore une fois la personne se déplace d'abord à A, mais elle découvre qu'elle a raté le bus et doit attendre le prochain.

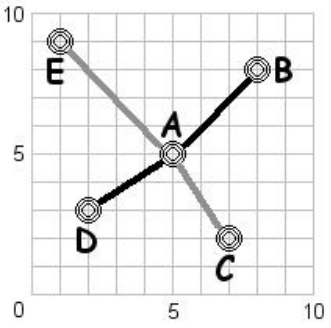
➙ Quand elle arrive à D, il lui aura pris sa 240 secondes (somme de la 1e colonne de la matrice U en tenant compte de la fréquence (3600 / f) pour les attentes).

• Utilisant la matrice U, la matrice et l'équation de *"att"*, nous sommes en mesure de calculer *att*. La valeur de *att* de cette solution est 155,491 secondes.

➙ Donc, il faut en moyenne à une personne 155,491 secondes pour arriver à destination.

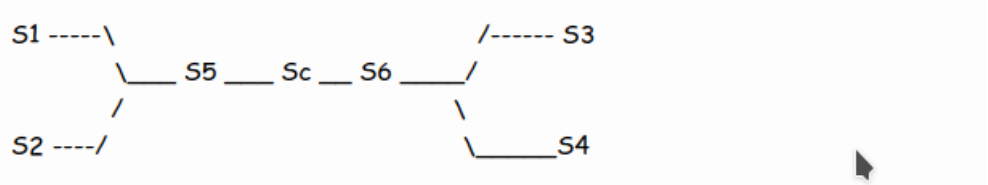
$$\boldsymbol{D} = \begin{bmatrix} 0 & 424.26 & 360.56 & 360.56 & 565.69 \\ 424.26 & 0 & 608.28 & 781.03 & 707.11 \\ 360.56 & 608.28 & 0 & 509.90 & 921.95 \\ 360.56 & 781.03 & 509.90 & 0 & 608.28 \\ 565.69 & 707.11 & 921.95 & 608.28 & 0 \end{bmatrix}$$

$$\boldsymbol{U} = \begin{bmatrix} 0 & 112.117 & 132.32 & 112.117 & 132.32 \\ 60.6092 & 0 & 132.32 & 112.117 & 372.32 \\ 51.5079 & 352.117 & 0 & 352.117 & 132.32 \\ 51.5079 & 112.117 & 132.32 & 0 & 372.32 \\ 80.8122 & 352.117 & 132.32 & 352.117 & 0 \end{bmatrix}$$



Quelques remarques et indications :

Soit une configuration d'arrêts :



Supposons qu'on décide de 2 lignes pour notre problème ; on ferait donc partir 2 colonies.

Comment faire pour départager les stops puisque les 2 colonies ne doivent pas utiliser exactement les mêmes stops, cela ferait double emploi.

Éléments de réponse :

1- faire un *clustering* des stops (cf. article BAP sur Jakarta) selon des critères géographiques/temporels/etc.

Dans ce cas, chaque ligne de bus aura ses arrêts. On peut éventuellement modifier ces 2 clusters.

P. ex pour la configuration ci-dessus (avec "Sc" : gare centrale) : on peut penser que comme pour "Sc", S5 et S6 feront partie des 2 clusters. Après tout, dans le cas réel, on a plusieurs bus qui passent par le même arrêt.

2- Faire partir 2 fourmis F1 et F2 depuis la gare centrale SC puis généraliser à deux colonies C1 C2.

Dans ce cas, la question revient au choix d'un prochain stop pour UNE fourmi.

A chaque fois que F_i veut choisir un prochain stop, elle le choisira parmi les stops non visités, ni par elle même ni par aucune autre F_j.

Option de partir de la gare centrale :

Supposons que C_1 (colonie 1) part de l'arrêt central Sc et arrive à S3. Comment faire pour ensuite couvrir les stops e.g. S2 et S5 pour C_1 ? Peut-on revenir sur ses pas ? Le bus revient-il en marche arrière ?

On peut effectivement séparer le cas de SC mais on voit que sauf dans le cas d'un graphe "complet" (où toute paire de noeuds est connectée), on risque de vouloir emprunter des routes directes (telle que la directe S3 –> S1 en faisant un détour).

On pourrait ne pas partir du centre mais choisir des points de départ aux extrêmes pour chauqe C_i.

I.13 Bonus 1 : Coloration de graphes

Coloration et colonie de fourmis : 3 catégories de méthodes :

1) Chaque fourmi est un algo constructif qui laisse une trace sur chaque paire de sommets non adjacents pour indiquer si ces sommets ont reçu la même couleur.

2) Des fourmis se promènent sur le graphe et tentent collectivement de modifier la couleur des sommets qu'elles visitent, l'objectif étant de diminuer le nombre d'arêtes conflictuelles dans une k-coloration non acceptable.

3) Les fourmis sont devenues des algorithmes de recherche locale qui laissent des traces sur l'exploration qu'elles ont faite de l'espace de recherche.

- Les algorithmes les plus récents (ceux de la troisième catégorie) rivalisent positivement avec les meilleurs algorithmes connus à ce jour.
- Voir "mise en oeuvre" ci-dessous.

I.13.1 Mise en oeuvre 1

- Soit un graphe de N noeuds et un ensemble de k couleurs, $k \leq N$.

  La 1e étape des calculs suivants a lieu en parallèle au niveau de chaque noeud.

☞ Il n'y a pas de notion de distance et toutes les fourmis envoyées arrivent en même temps à destination

- Chaque noeud a un degré D de connexions et lâchera D fourmis locales, une par voisin.
- Au départ, on active chaque noeud en parallèle avec les autres.
- Chaque noeud choisit une couleur C et envoie une fourmi par voisin pour lui dire de ne pas prendre C.

  Ce message peut être traduit par le dépôt de phéromone sur les autres couleurs ($\neq C$) du voisin : éliminer une couler chez le voisin = favoriser les autres couleurs : ../..

- Le messager porte le numéro de l'envoyeur.
- L'action du choix d'une couleur au niveau d'un noeud peut être confiée à une fourmi supplémentaire par noeud.

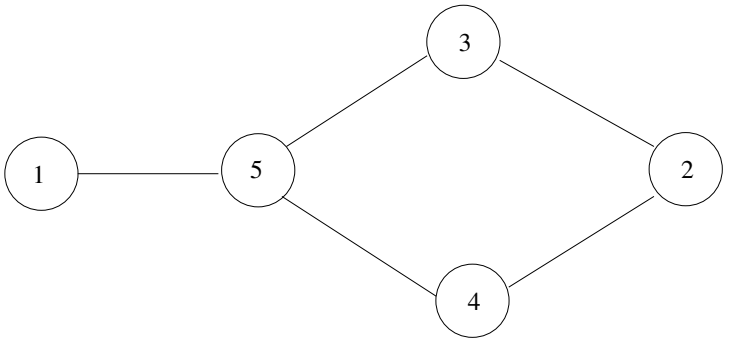
- **Itération :** une décision locale est prise au niveau de chaque noeud M :
  - Examiner les fourmis présentes dans M.

    ➙ Chacune est là pour dire : "ne prend pas telle couleur" (i.e. elle favorise les autres couleurs).

    ➙ Ordonner les noeuds selon le nombre maximum d'un même message venant des voisins différents (voir exemple ci-dessous).
  - Choisir le noeud qui maximise ce nombre (venant d'un même noeud, le dernier message annule le précédent)
  - Lui donner une couleur non contestée.
  - Refaire l'itération

I.13.2 Exemple

• Soit ce graphe avec 5 noeuds et un ensemble de $k \leq 5$ couleurs $\{R, V, B, ...\}$



• Au départ, chacun choisit la couleur R (rouge) et envoie autant de fourmis que de connexions à ses voisins.

• La table suivante résumé les actions (avec optimisation mais non parallèle) :

../..

Commentaires et décisions	1	2	3	4	5
1er tour, tous choisissent R Et envoient le message $\overline{R}_{dest}$ aux voisins	R $\overline{R}_5$	R $\overline{R}_{3,4}$	R $\overline{R}_{2,5}$	R $\overline{R}_{2,5}$	R $\overline{R}_{1,3,4}$
Comptage (maxi même $\overline{C}$ de $\neq$dest) ➙ Choix de 5 (ne prend pas R) Et envoi de 3 fourmis $\overline{V}_5$ aux voisins	1 $\overline{V}_5$	2	2 $\overline{V}_5$	2 $\overline{V}_5$	3 prend **V**
Etat messages (nb. fourmis présentes + messages)	$\overline{R}_5$, $\overline{V}_5$	$\overline{R}_{3,4}$	$\overline{R}_{2,5}$, $\overline{V}_5$	$\overline{R}_{2,5}$, $\overline{V}_5$	**V**
Annulation du message $\overline{R}_5$ (5 a pris V) ➙ Venant de 5, le nouv. messager $\overline{V}_5$ tue le préc $\overline{R}_5$	$\overline{V}_5$	$\overline{R}_{3,4}$	$\overline{R}_2$, $\overline{V}_5$	$\overline{R}_2$, $\overline{V}_5$	**V**
Max même couleur exp. différents ☞ un nouv. message du même exp. annule le préd. Choix de 2 (qui ne prend pas R) Et envoi de fourmis $\overline{V}_2$ aux voisins	1	2 **V**	1 $\overline{V}_2$	1 $\overline{V}_2$	
Bilan : ($\overline{V}_2$/$\overline{V}_5$ annule $\overline{R}_2$ /$\overline{R}_5$ car 2 et 5 ont pris V) Les 3 noeuds restants sont déconnectés et prennent R	$\overline{V}_{2,5}$	**V**	$\overline{V}_{2,5}$	$\overline{V}_{2,5}$	**V**

☞ **Optimisation** : on peut calculer en parallèle 2 noeuds non connectés.

I.13.3 Mise en oeuvre 2

L'idée de base de l'ACO est d'imiter le comportement des fourmis en trouvant le chemin le plus court entre leur nid et une source de nourriture.

Dans le cas de la coloration d'un graphe, on peut considérer chaque sommet comme une "ville" qu'une fourmi peut visiter, et les couleurs comme des "phéromones" que la fourmi laisse derrière elle.

Principe de la mise en oeuvre :

• Dans certaine versions, on attribue dès le début une couleur au hasard à chaque sommet. Cela permet de harmoniser l'algorithme de recherche : quand on arrive à un sommet, il y a une couleur (vs. quand on arrive à un sommet, il y a une couleur si une fourmi est déjà passée par là) que l'on cherche à confirmer ou non. Ignorez cette initialisation si vous n'en voyez pas pour l'instant l'intérêt !

• Faire partir plusieurs fourmis pour explorer le graphe.

Chaque fourmi choisit un sommet et lui attribue une couleur basée sur les niveaux de phéromones sur les arêtes menant aux sommets voisins.

La probabilité qu'une fourmi choisisse un sommet particulier est proportionnelle au niveau de phéromone sur les bords menant à ce sommet. Pour ce choix, appliquez les "formules" habituelles.

➛ Chaque fourmi fait une coloration complète.

• Une fois qu'une fourmis a terminé sa visite, mettez à jour les niveaux de phéromone sur les arêtes en fonction de la qualité de la solution trouvée.

Les solutions de qualité supérieure se voient attribuer des niveaux plus élevés de phéromones.

• **Pour trouver une coloration** : appliquez un algorithme de recherche locale optimal (c-à-d. un algorithme pas trop bête !) pour améliorer la qualité de la solution trouvée par les fourmis.

• Ce principe (traduit en algorithme) peut converger vers une solution quasi optimale.

Cependant, comme d'autres algorithmes méta-heuristiques, la qualité de la solution trouvée par ACO peut dépendre de l'instance du problème et des paramètres utilisés.

Il est donc important de choisir soigneusement les paramètres et de tester l'algorithme sur une gamme de graphes.

Lecture (pour mieux comprendre) :

Vous trouverez d'autres indications dans l'article fourni :

"Modified PSO algorithm for solving planar graph coloring problem"

☞ N'hésitez pas à vous documenter davantage sur le WEB.

I.14 Sujet 3 : Trafic et Routage

- Lire et implanter les articles déposés à coté de ce support.

1 ◦ "How to Mitigate Traffic Congestion Based on Improved Ant Colony Algorithm : A Case Study of a Congested Old Area of a Metropolis"

2 ◦ " Dynamic Vehicle Routing Problems with Enhanced Ant Colony Optimization"

I.15 A rendre : modalités

- Choisir un des sujets :
  - Navigation Probabiliste (SLAM, cours 2) : noté sur 20.
    - ➛ Si interface graphique (GUI) : (Tkinter/SDL/QT/...) : noté sur 25.
  - Robot + SLAM : noté sur 25.
  - Colonie de fourmis : noté sur 12. Si GUI : sur 15, ajoute des AG : noté sur 20.
  - Bus (BAP), Trafic, Routage : noté sur 17. Si GUI : noté sur 20.
  - Routage
  - Algorithmes Génétiques pour générer des Images (long) : noté sur 17 (si GUI : sur 20)
  - Bonus 1 / 2 (coloration) : 5 points Bonus
- Rendre un rapport pdf + les codes à déposer sur Moodle.
  - ➛ Délais : un mois après le 2e BE.

I.16 Quelques References

1. Marais, E. N., de Kok, W.(trans.) 1937 : The Soul of the White Ant http ://journeytoforever.org/farm_library/Marais1/whiteantToC.html
2. Maeterlinck, Maurice. 1927 : The Life of the White Ant. Allen & Unwin
3. Grassé, P.-P. 1959 : La Reconstruction du nid et les coordinations interindividuelles. La théorie de la stigmergie, Insectes Sociaux 6 : 41-84.
4. Goss. S., Aron. S., Deneubourg, J.L. and Pasteels, J.M. 1989 : Self-organized shortcuts in the Argentine ant. Naturwissenschaften 76, 579-581.
5. Benzatti, D. 2002 : Emergent Intelligence, AI Depot http ://ai-depot.com/CollectiveIntelligence/Ant.html
6. Dorigo, M. and Gambardella, L.M.. Ant Colony System : A Cooperative Learning Approach to the Traveling Salesman Problem. IEEE Transactions on Evolutionary Computation, 1(1) :53-66, 1997. http ://citeseer.nj.nec.com/article/dorigo96ant.html
7. Schoonderwoerd, R., Holland, O., Bruten, J. and Rothkrantz, L., 1996 : Ant-based load balancing in telecommunications networks, Adaptive Behavior, vol.5, No.2, .
8. Di Caro, G and Dorigo, M. 1998 : AntNet : Distributed Stigmergetic Control for Communications Networks. Journal of Artificial Intelligence Research, 9 :317–365, . http ://citeseer.nj.nec.com/dicaro98antnet.html
9. Eberhart, R. C., and Kennedy, J. 1995 : A New Optimizer Using Particles Swarm Theory, Proc. Sixth International Symposium on Micro Machine and Human Science (Nagoya, Japan), IEEE Service Center, Piscataway, NJ, 39-43.
10. Collective Robotic Intelligence Project (CRIP) - http ://www.cs.ualberta.ca/k̃ube/research.html
11. http ://www.swarm-bots.org
12. J. Hoffmeyer, The swarming body, Proc. 5th Congress of the International Association for Semiotic Studies, Berkeley, 1994. http ://www.molbio.ku.dk/MolBioPa

Table des matières