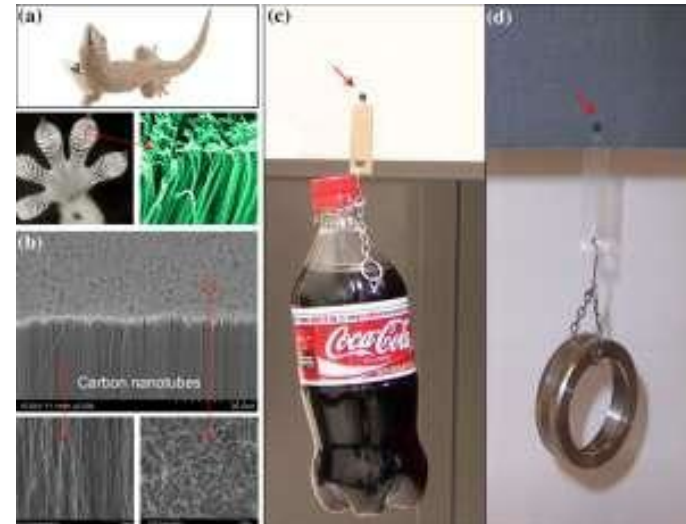


S8 - BE

Introduction aux systèmes collaboratifs
(BE Colonie de Fourmis)
Ecole Centrale de Lyon
2024-2025

Alexander Saidi

I.1 Intelligence Emergente



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

- Les **oiseaux** et les **avions** (mimétisme);
- **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** par la création de nanotubes de carbone pour collage
(ici une bouteille de 650gr sur une bande de 4x4mm, ...)

Intelligence Naturelle : ../..



- **Hydrophobie** : la **peau des requins** contient un motif spécial qui permet une circulation rapide d'eau et empêche l'encrassement. Motif copié par dépôt de la cire.
 - ➔ Utilisé dans les technologies pour les bateaux / nageurs / surfeurs / ... qui s'en inspirent !
- **Hydrophobie** : 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. ...
- **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 \Leftrightarrow structures d'ailes d'oiseaux.
- Les robots et leurs mouvements \Leftrightarrow 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, ...
- Colonies de fourmis, Colonie d'abeilles, Termitière, Etc.

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'à 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 : TSP avec la colonie de fourmis
 - 4- Bus Allocation Problem (BAP) & colonies de fourmis
 - 5- Ramassage scolaire avec la colonie de fourmis
 - 6- Coloration de graphes par colonie de fourmis / PSO (Bonus en plus)
 - 7- Algorithmes Génétiques en génération d'images
 - 8- Routage et trafic (voir le pdf)
 - 9- AntNet : simulation d'un réseau, congestion aléatoire, test de la Colonie, ...
- 👉 **Le sujet "navigation probabiliste" cherche élèves courageux !**

I.2 Colonies de fourmis

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

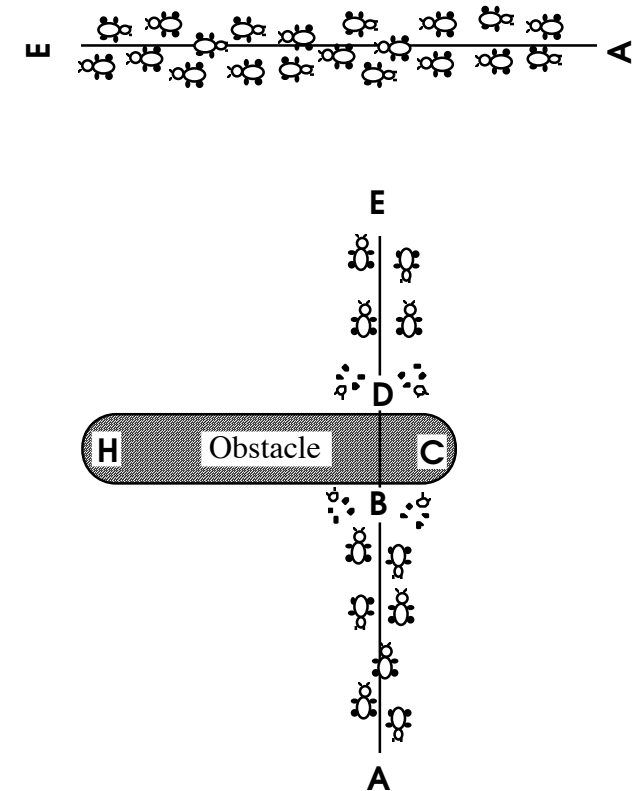
- L'optimisation de colonie de fourmis (ACO) étudie les systèmes artificiels qui imitent le comportement des vraies colonies de fourmis.
- Employée pour résoudre des problèmes discrets d'optimisation.
- Constitue une classe de méta-heuristiques pour des problèmes d'optimisation difficiles (NP-difficiles) et les problèmes **dynamiques** (en cours de résolution).
- Voir support et cours 1 pour "un peu d'histoire" sur ce sujet.

I.2.1 Rappel des 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

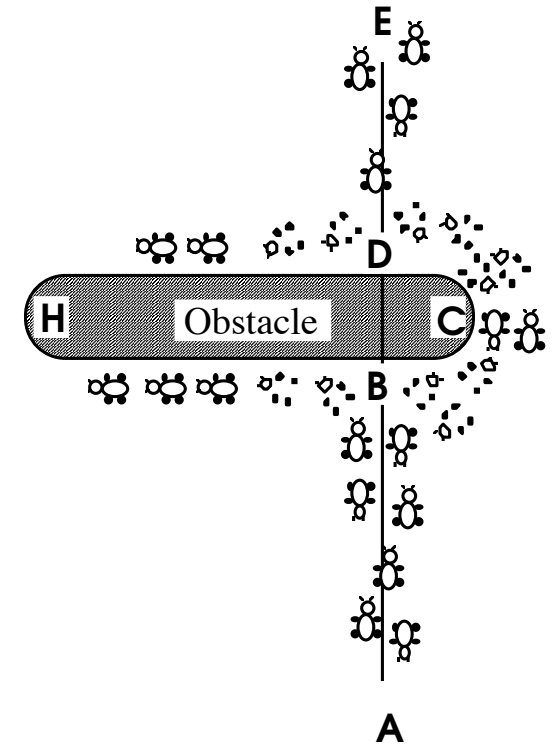
- Les fourmis se déplacent entre le nid (E) et la source de la nourriture (A) sur le chemin A-E.
- Un obstacle coupe le chemin.
- La fourmi qui se déplace de A vers E et qui se trouve en B a 2 choix :
 - Le chemin B-C-D
 - Le chemin B-H-D
- $B-C-D$ ou $B-H-D$?
 - le choix selon $f(\text{intensité de la phéromone})$



- 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

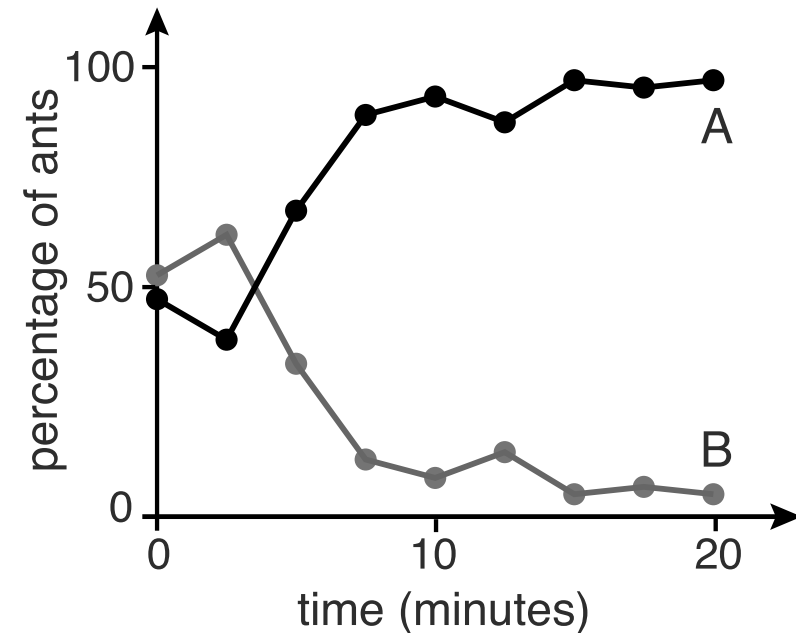
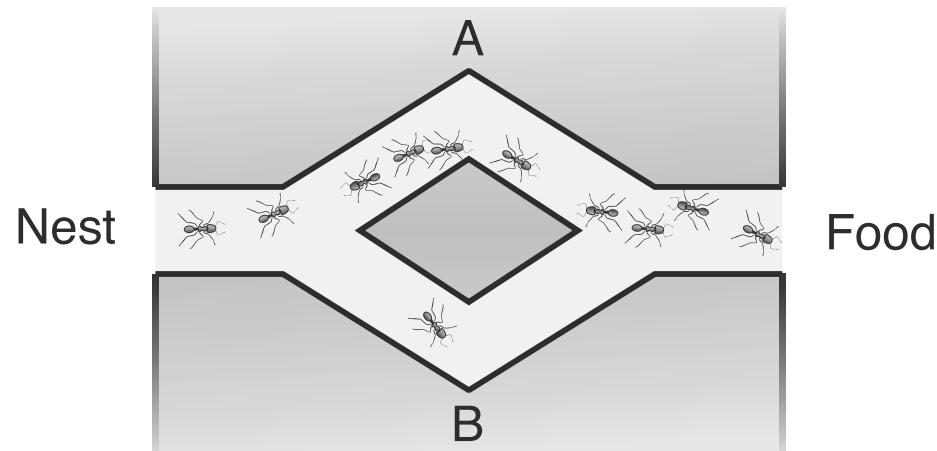


FIGURE I.1 – Dépôt et évaporation de Phéromone par les fourmis : la voie A saturée, la voie B quasi nulle

- ➔ 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 :
 - Environnement "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

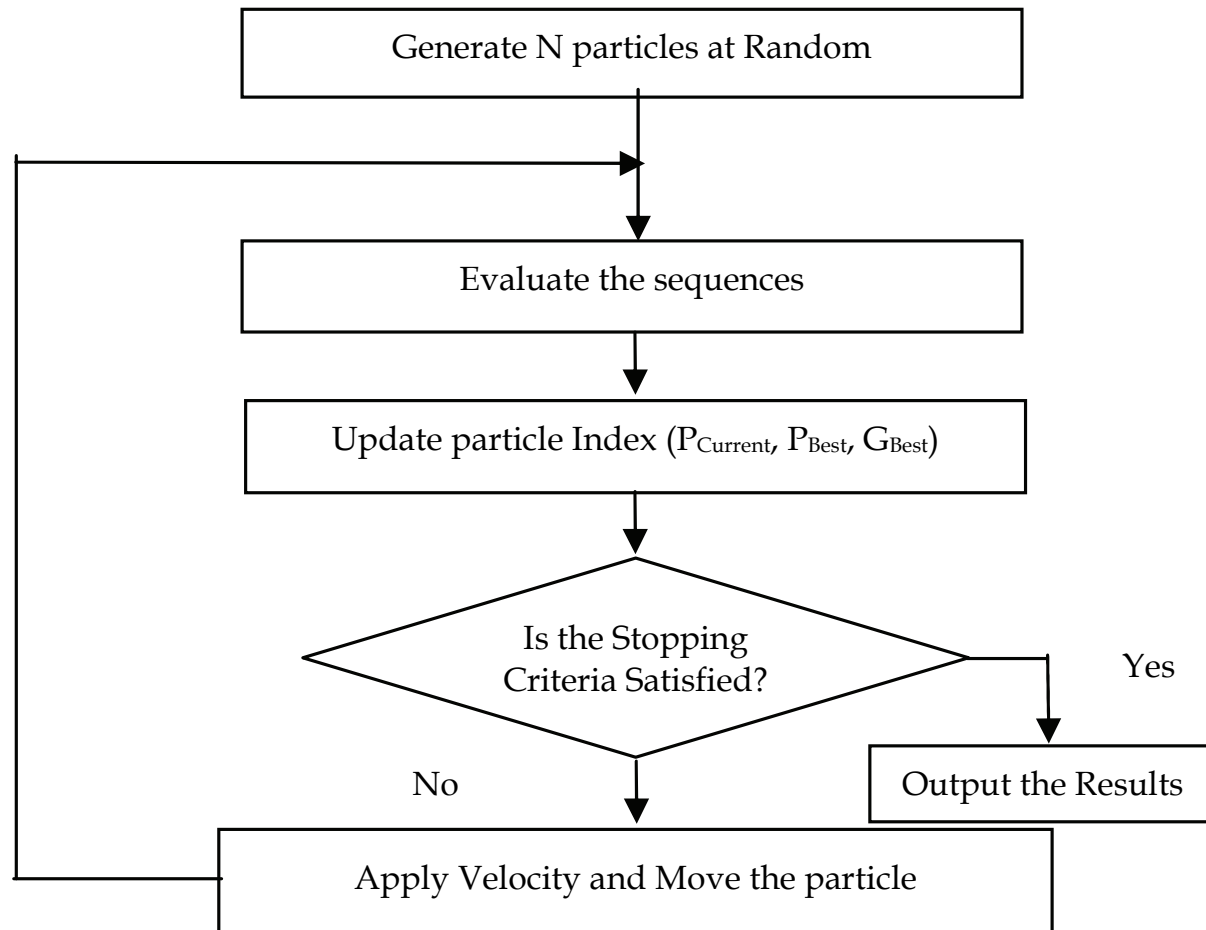


FIGURE I.2 – Algorithme de base de PSO : **pbest** : le meilleur local, **gbest** : le meilleur global

👉 Voir cours 1 : l'algorithme PSO, exemple de calcul de la **vélocité** et du **déplacement**.

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 \leftarrow (p.c : la coordonnée c de la particule p)

$p.c = p.c + 2 * \text{rand}() * (pbest.c - p.c) + 2 * \text{rand}() * (gbest.c - p.c)$

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 1 : 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 d'un pas (on ne recule jamais) ou faire un saut d'arête
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.
 - ☞ Il peut également **aller directement au noeud de destination** de cette arête.
- 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.

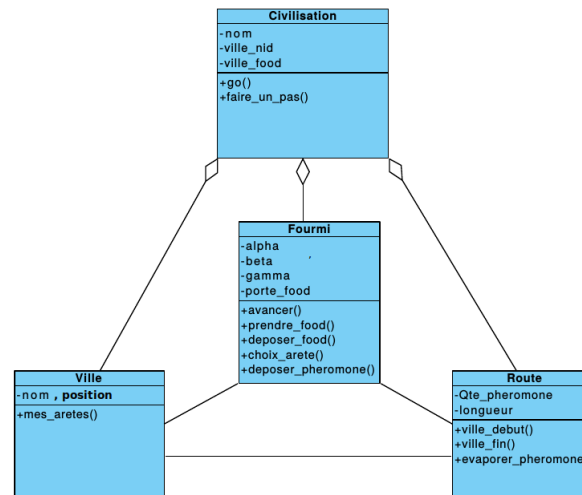


FIGURE I.3 – Une proposition possible (à compléter)

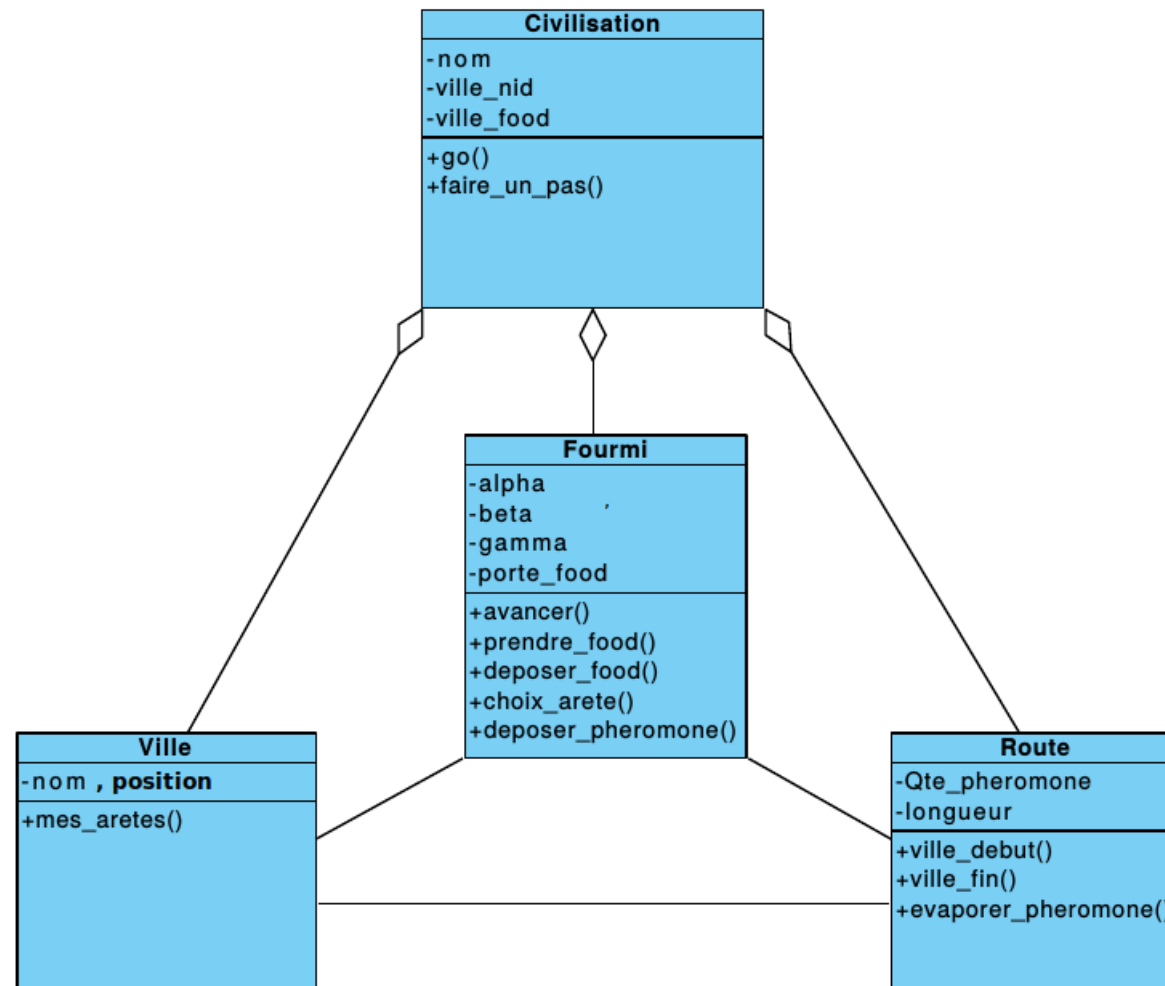


FIGURE I.4 – Une proposition possible (version de base, à compléter)

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 nourriture puis emprunter une arête de retour

Si au Nid :

Si porteur de nourriture 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 ,

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

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

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

☞ Ceci n'est qu'un exemple (ne sera pas utilisé)! 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 = ... # Transporte de la nourriture ou non  
  
    def getTendance(self, PheroLevel : float) :    # tendance à choisir une route  
    def prendre_nourriture(self, ...) :           # Prendre la nourriture dans la source de nourriture  
    def laisser_nourriture(self, ...) :           # Laisse la nourriture (dans le nid)  
    def déposer_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)

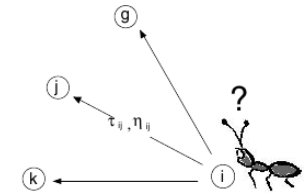
- La classe *Civilisation* contrôle l'ensemble de l'environnement :
 - le processus d'évolution et de simulation.
- Deux noeuds du graphe désignent le **nid** et la **source** de nourriture.
- Au début de la simulation, on crée un nombre aléatoire d'agents dans le nid.
- A chaque tour, les agents effectuent 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 nourriture de la Civilisation  
        self.__nid = ...              # Le nid de la Civilisation  
        self.__routes = ...           # toutes les routes de l'Environnement  
        self.__villes = ...           # toutes les villes de l'Environnement  
        self.__fourmis = ...          # toutes les fourmis dans l'Environnement  
        self.__selectionNaturelle = ... # les tours restants avant la prochaine sélection (pour l'algorithme génétique)  
  
    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 :

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

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 ,

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

- 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)

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

☞ $P_k(r, s)$ 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. $\{s \mid \text{Seuil}_1 \leq P_k(r, s) \leq \text{Seuil}_2\}$ 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

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

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 :

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

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 :

$$\tau_{ij}^{n+1} \leftarrow \tau_{ij}^n + \Delta\tau_{ij}^k \quad \forall (i, j) \in T^k \quad T^k : \text{trajet de la fourmi } k \text{ voir ci-après}$$

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

- Cette quantité est définie par :

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

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.

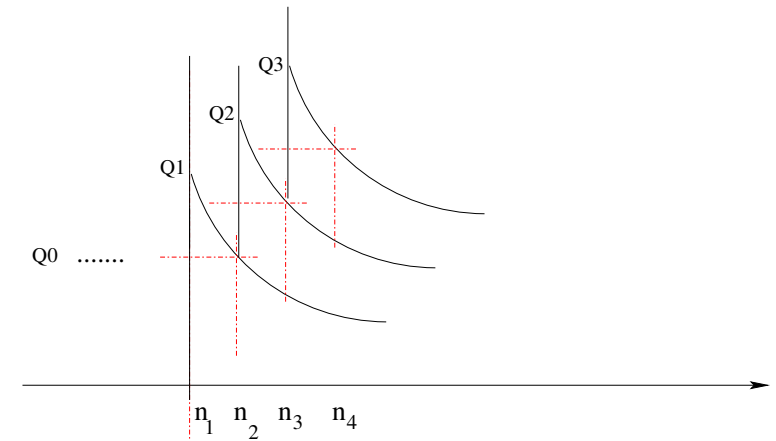
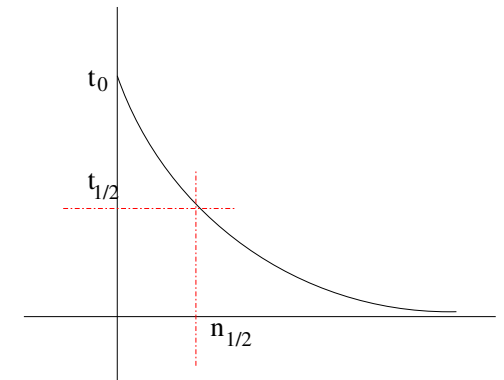
D'où le nombre d'itérations pour atteindre cette quantité :

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

- 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 :

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

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 ville 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 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.

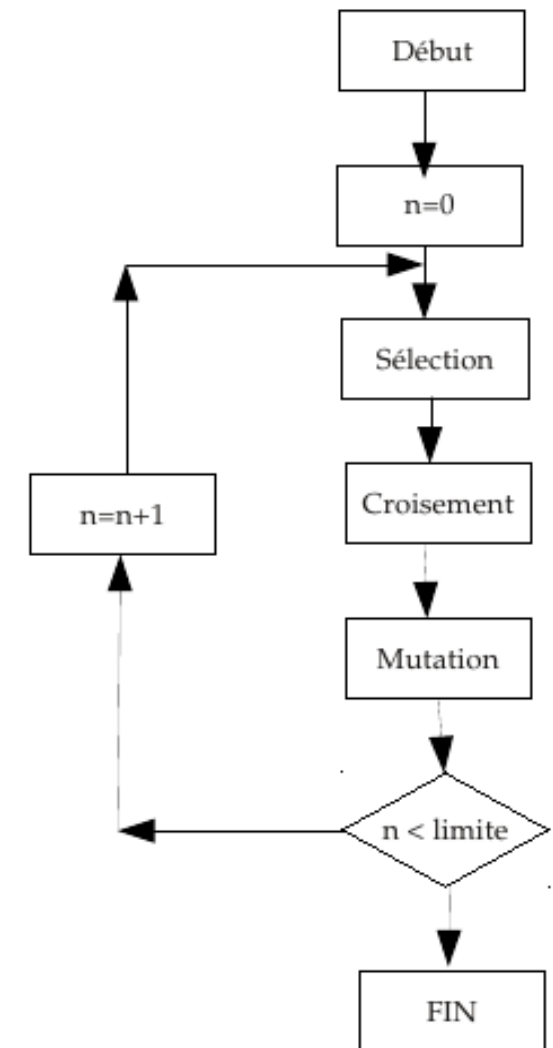
Sélection (reproduction) : comme la sélection naturelle, on cherche à choisir les populations prometteuses : celles qui minimisent par exemple une erreur moindre carré ou une distance p/r à une fonction **objectif** (ou une référence).

- décision selon une valeur "*fitness*" (e.g. Erreur Moindre Carrée)
- et / ou selon l'intérêt d'une population \pm 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.

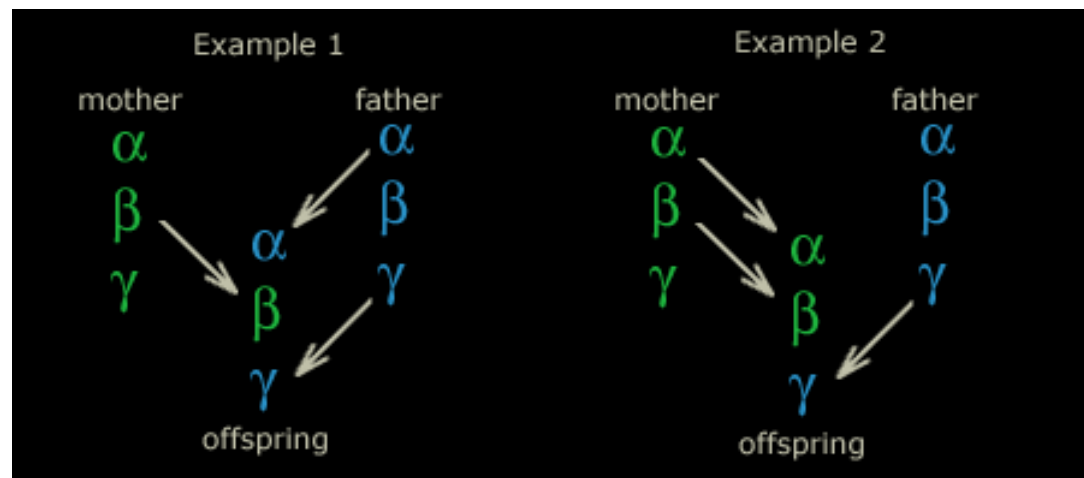


FIGURE I.5 – deux exemples de descendants créé par croisement

- 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 :

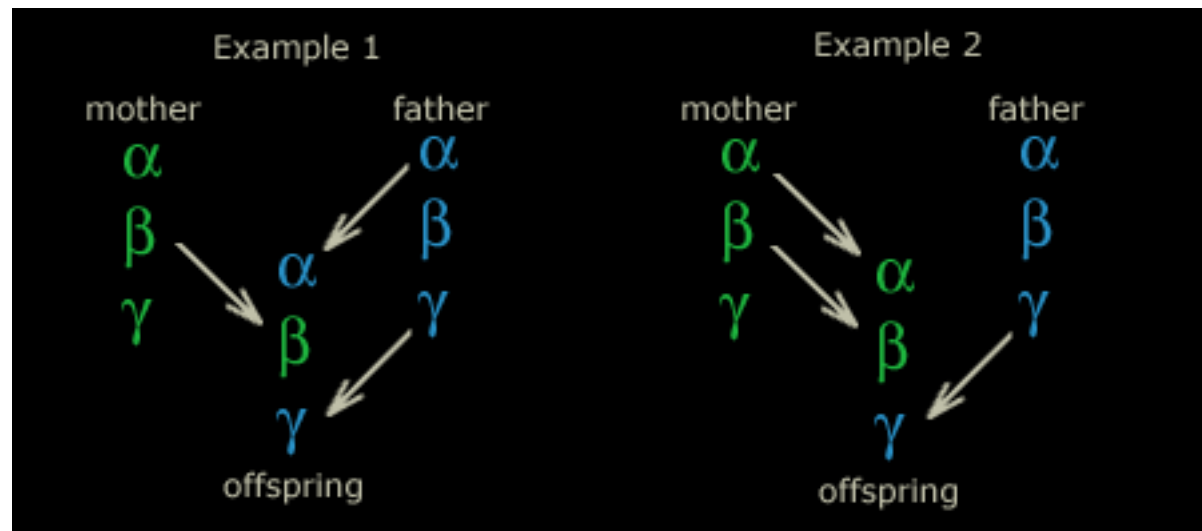


FIGURE I.6 – deux exemples de descendants créé par croisement

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 PCC

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.

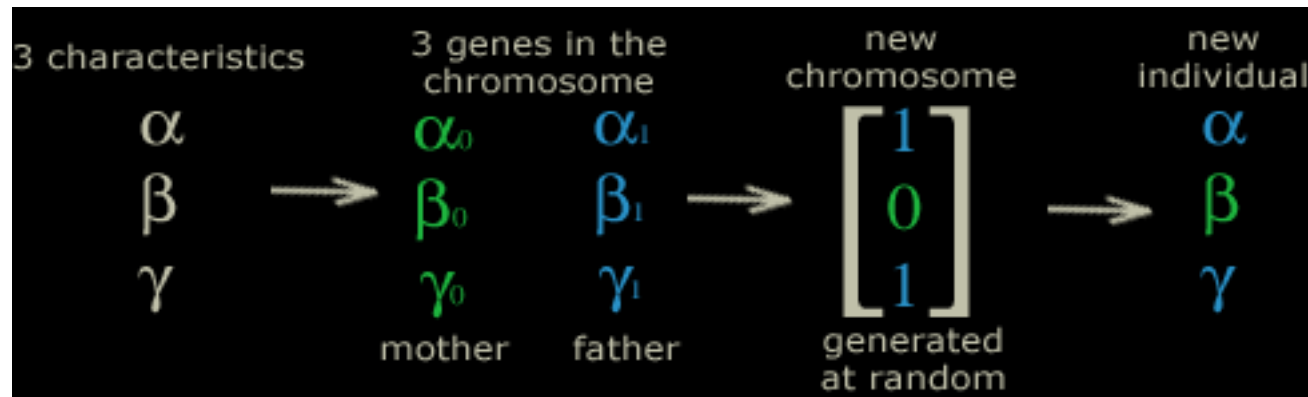


FIGURE I.7 – Implantation de CrossOver (sur la base de α, β, γ)

- 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 # Transporte de la nourriture ou non
    qte_nourriture_collectee : int # Quantité de nourriture 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) # Evaluate la tendance à choisir une route (proba)
    def prendre_nourriture(... ) : # Prendre la nourriture dans la source de nourriture
    def laisser_nourriture(... ) : # Laisse la nourriture (dans le nid)
    def déposer_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

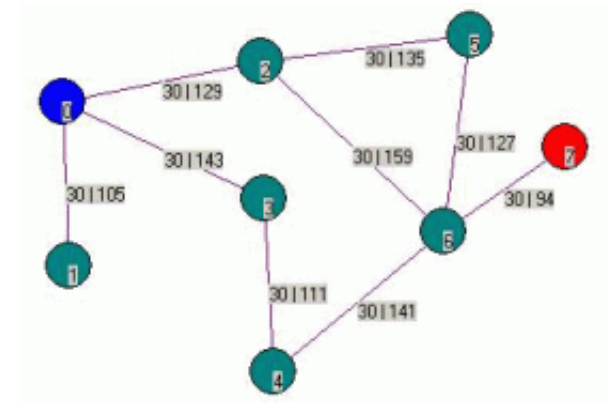
- Comme il n'y a pas de **fonction de coût explicite** liée au graphe, ce système peut être utilisé dans des applications avec un **environnement inconnus et temps réel** :
 - par exemple pour l'exploration robotique dans un environnement dynamique.
- On note qu'aucune connaissance préalable n'est nécessaire 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 (selon les recherches)
- Le système est également adaptatif et réactif.
 - 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 **algorithmes génétiques** sont 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**.

Démo PCC :

-  (ECL-S8-PhM-2024-2025)



I.12 Sujet 2 : le problème de Sac à dos

Exemple :

- Un alpiniste dispose d'un sac et d'une liste d'objets qu'il est susceptible d'emporter dans le sac.
- Chaque objet apporte un certain confort à l'alpiniste et chaque objet prend une certaine place dans le sac.
- La capacité du sac est limitée, le problème consiste alors pour l'alpiniste à maximiser son confort tout en ne dépassant pas la capacité du sac.

Un exemple (similaire) :

- le problème d'investissement où l'on dispose d'un budget fixe, de n projets, où chaque projet est identifié par un indice j , $j = 1, \dots, n$, un profit, et un coût d'investissement.
- L'**investissement optimal** peut être déterminé par la résolution d'un problème de type **sac-à-dos**.

Généralisation du problème :

- D'une façon générale, le problème du sac-à-dos consiste à remplir :
 - un sac dont la capacité est fixée,
 - avec un sous ensemble d'objets de poids (et ou volume) et
 - et de profit connus,
 - de manière à satisfaire les deux conditions suivantes :
 1. le poids cumulé du sous-ensemble d'éléments choisis ne dépasse pas la capacité du sac.
 2. maximiser le profit généré par le sous-ensemble d'éléments choisis.
- Le problème du sac-à-dos possède de nombreuses applications, en particulier dans sa version multidimensionnel (Multidimensional Knapsack Problem : MKP) qui est un problème d'optimisation combinatoire sous contraintes NP-difficile.

Plusieurs problèmes pratiques peuvent être formalisés comme un MKP.

Par exemple, le problème d'allocation des processeurs et des bases de données dans les systèmes informatiques distribués, le chargement des cargaisons, les problèmes de découpage de stock, etc.

Le MKP permet de modéliser une grande variété de problèmes où il s'agit de maximiser un profit tout en ne disposant que de ressources limitées.

Définition :

- On considère un ensemble d'objets étiquetés de 1 à n .
- Chaque objet $j \in \{1, \dots, n\}$ dispose d'un poids w_j (un entier) et d'un profit p_j .
- On dispose d'un sac dont le contenu ne doit excéder une capacité c (un entier) fixée.
- Remplir le sac en maximisant la somme des profits sous la contrainte de capacité.

Le problème de Sac à Dos a une solution en programmation Mathématique.

→ L'expression d'un Programme Mathématique ne vaut pas solution. **Il faut le meilleur ?**

Expression PLNE du problème (Programme Mathématique) :

• Plus précisément, on s'intéresse à la résolution du programme linéaire à variables binaires suivant :

$$x \in \operatorname{argmax} \sum_{1 \leq j \leq n} p_j x_j \text{ sous la contrainte } \sum_{1 \leq j \leq n} w_j x_j \leq c$$

• Le vecteur $x := (x_j, 0 \leq j \leq n) \in \{0, 1\}^n$ est une solution du problème où $x_j = 1$, si l'objet j est mis dans le sac et $x_j = 0$ sinon.

• Le problème consiste donc à choisir un sous-ensemble d'objets parmi la liste d'objets initiale afin de maximiser la fonction objectif suivante : $Z(x) = \sum_{j=1}^n p_j x_j$

• Pour éviter les cas triviaux, nous considérons l'hypothèse suivante :

$$\forall j \in \{1, n\}, w_j \leq c \text{ et } \sum_{j=1}^n w_j > c.$$

- Un algorithme **Glouton** trivial :

Entrée : Une instance d'un problème de sac à dos ;

Sortie : Une solution réalisable $x' = \{x_j\}, j \in \{1, n\}$;

1. $c' := c$;

2. Pour $j := 1$ jusqu'à n faire

 Si $w_j \leq c'$ alors $x'_j := 1; c := c' - w_j$

 Sinon $x'_j := 0$

 FinSi

FinPour

3. Sortir avec x'

- Dans cette solution, on prend au hasard tout objet x_j du poids w_j dont l'ajout au sac ne dépasse pas la capacité c du sac. Ici, le profit total n'est pas maximisé.

Algorithme ACO Ant-Knapsack (KP) :

Initialiser les traces de phéromone à τ_{max}

Répéter

Pour chaque fourmi $k = 1..nbAnts$, construire une solution S_k comme suit :

Choisir aléatoirement un premier objet $o_1 \in 1..n$

$S_k \leftarrow \{o_1\}$

$Candidates \leftarrow \{o_i \in 1..n | o_i \text{ peut être sélectionné sans violer des contraintes de ressources}\}$

Tantque $Candidates \neq \emptyset$ faire :

Choisir un objet $o_i \in Candidates$ avec la probabilité
$$p_{S_k}(o_i) = \frac{[\tau_{S_k}(o_i)]^\alpha \cdot [\eta_{S_k}(o_i)]^\beta}{\sum_{o_j \in Candidates} [\tau_{S_k}(o_j)]^\alpha \cdot [\eta_{S_k}(o_j)]^\beta}$$

$S_k \leftarrow S_k \cup \{o_i\}$

enlever de $Candidates$ chaque objet qui viole des contraintes de ressources

fin Tantque

fin pour

mettre à jour les traces de phéromone en fonction de $\{S_1, \dots, S_{nbAnts}\}$

si une trace de phéromone est inférieure à τ_{min} alors la mettre à τ_{min}

si une trace de phéromone est supérieure à τ_{max} alors la mettre à τ_{max}

fin Répéter (voir ci-dessous)

Détails de l'algorithme KP (KnapSac) :

- A chaque cycle de cet algorithme, chaque fourmi construit une solution.
- ☞ **Lorsque toutes les fourmis ont construit une solution, les traces de phéromone sont mises à jour.**
- **L'algorithme s'arrête** lorsqu'une fourmi a trouvé une solution optimale (si une valeur optimale est connue), ou lorsqu'un nombre maximal de cycles a été atteint.
- **N.B.** cet algorithme est inspiré d'un algorithme *MAX-MIN Ant System* dans lequel les traces de phéromone sont limitées à l'intervalle $[\tau_{min}, \tau_{max}]$ et sont initialisées à τ_{max} .
- Pour construire une solution, les fourmis choisissent aléatoirement (voire, relative à une probabilité) **un objet initial**, puis ajoutent itérativement des objets qui sont choisis à partir d'un ensemble *Candidats* qui contient tous les objets qui peuvent être sélectionnés sans violer de contraintes de ressources.
- A chaque étape, l'objet o_i à ajouter à la solution en cours S_k est choisi parmi l'ensemble de sommets *Candidats* **relativement à une probabilité** $p_{S_k}(o_i)$ (voir l'algorithme).
- Cette probabilité est définie proportionnellement à un facteur phéromonal $\tau_{S_k}(o_i)$ et un facteur heuristique $\eta_{S_k}(o_i)$, ces deux facteurs étant pondérés par deux paramètres α, β qui déterminent leur importance relative./..

- Le facteur phéromonal dépend de la stratégie phéromonale choisie (voir plus loin).
- Le facteur heuristique $\eta_{S_k}(o_i)$ est défini par :
 - soit $d_{S_k}(i) = b_i - \sum_{j \in S_k} r_{ij}$ la qté. restante de la ressource i lorsque la fourmi a construit la sol. S_k ,
 où $b_i, i \in 1..m$: la quantité totale disponible de la ressource i ,
 $r_{ij}, i \in 1..m, j \in 1..n$: la consommation de la ressource i par l'objet j ,
 m : nbr de ressources,
 n : nombre d'objets,
 $p_j, j \in 1..n$ le profit apporté par l'objet j ,
 - On définit le ratio : $h_{S_k}(j) = \sum_{i=1}^m \frac{r_{ij}}{d_{S_k}(i)}$
 qui représente la *dureté* de l'objet j par rapport à toutes les contraintes $i \in 1..m$ et relativement à la solution construite S_k , de sorte que **plus ce ratio est faible plus l'objet est intéressant**.
 - On intègre alors le **profit** p_j de l'objet j pour obtenir un ratio profit/ressource en définissant le facteur heuristique par : $\eta_{S_k}(j) = \frac{p_j}{h_{S_k}(j)}$

- Une solution d'un KP est un ensemble d'objets sélectionnés $S = \{o_1, \dots, o_k\}$ (un objet o_i est sélectionné si la variable de décision correspondante $x_{o_i} = 1$ dans la solution).
- **Deux points clés** : décider
 - sur quels composants (des solutions) la phéromone devra être déposée
 - comment exploiter la phéromone lors de la construction de nouvelles solutions.
- Étant donnée une solution S , il y a 3 différentes manières de déposer la phéromone :
 - 1 ◦ déposer les traces de phéromone **sur chaque objet sélectionné dans S** .
 Dans ce cas, lors de la construction des solutions suivantes, la probabilité de sélectionner chaque objet de S sera augmentée,
 - 2 ◦ déposer les traces de phéromone **sur chaque couple (o_i, o_{i+1})** (2 objets successivement ajoutés dans S).
 Dans ce cas, l'idée est d'augmenter la probabilité de choisir l'objet o_{i+1} si le dernier objet sélectionné est o_i ,
 - 3 ◦ déposer les traces de phéromone sur toutes **les paires (o_i, o_j)** (2 objets appartenant à S).
 Dans ce cas, lors de la construction d'une nouvelle solution S' , la probabilité de choisir l'objet o_i sera augmentée si o_j a déjà été sélectionné dans S
 Plus précisément, plus S contiendra de sommets appartenant déjà à S , plus les autres sommets de S auront de chances d'être sélectionnés.

Définition des composants phéromonaux

- On compare les **3 stratégies phéromonales** dans l'algorithme précédent *Ant-Knapsack* (AK) :
- Dans cet algorithme, les fourmis évoluent et déposent de la phéromone sur le graphe complet $G = (V, E)$ avec V : l'ensemble des objets.
- On aura les **trois versions différentes** en fonction des composants du graphe sur lesquels les fourmis déposent des traces de phéromone :
 - 1◦ Dans *Vertex-AK*, les fourmis **déposent la phéromone sur les sommets** V du graphe. La quantité de phéromone sur un objet $o_i \in V$ est notée $\tau(o_i)$. Cette quantité représente la "désirabilité" de choisir l'objet o_i lors de la construction d'une solution,
 - 2 ◦ Dans *Path-AK*, les fourmis **déposent de la phéromone sur les couples de sommets** sélectionnés consécutivement, i.e. sur les arcs du graphe. La quantité de phéromone sur un arc $(o_i, o_j) \in E$ est notée $\tau(o_i, o_j)$. Cette quantité représente la "désirabilité" de choisir l'objet o_i juste après avoir choisi l'objet o_j lors de la construction d'une solution.
- Il est à noter que **dans ce cas le graphe est orienté** (d'où les "arcs"),

3o Dans *Edge-AK*, les fourmis déposent la phéromone sur les **paires de sommets sélectionnés dans une même solution** (donc pas forcément deux noeuds successifs), i.e. sur les **arêtes** E du graphe.

La quantité de phéromone sur une arête $(o_i, o_j) \in E$ est notée $\tau(o_i, o_j)$.

Cette quantité représente la désirabilité de choisir un objet o_i lors de la construction d'une solution qui contient déjà l'objet o_j . Il est à noter que, dans ce cas le graphe est non orienté, et donc $\tau(o_i, o_j) = \tau(o_j, o_i)$.

Définition du facteur phéromonal

- Le facteur phéromonal $\tau_{S_k}(o_i)$ traduit l'expérience passée de la colonie. Il est utilisé dans la probabilité de transition des fourmis pour les guider vers des zones de l'espace de recherche prometteuses.

Ce facteur phéromonal dépend de la quantité de phéromone déposée sur les composants phéromonaux du graphe :

- Dans *Vertex-AK*, il dépend de la quantité déposée sur l'objet candidat, i.e. $\tau_{S_k}(o_i) = \tau(o_i)$
- Dans *Path-AK*, il dépend de la quantité présente sur l'arc connectant le dernier objet ajouté à la solution partielle S_k et l'objet candidat o_i , i.e. si le dernier objet ajouté dans S_k est o_j , $\tau_{S_k}(o_i) = \tau(o_i, o_j)$
- Dans *Edge-AK*, il dépend de la quantité déposée sur les arêtes connectant l'objet candidat o_i avec

les différents objets présents dans la solution partielle S_k , i.e. $\tau_{S_k}(o_i) = \sum_{o_j \in S_k} \tau(o_i, o_j)$

- **N.B.** : ce facteur phéromonal peut être calculé de façon incrémentale :

lorsque le premier objet o_1 a été sélectionné, le facteur phéromonal $\tau_{S_k}(o_j)$ est initialisé à $\tau(o_1, o_j)$ pour chaque objet candidat o_j ;

→ ensuite, à chaque fois qu'un nouvel objet o_i est ajouté à la solution courante S_k , le facteur phéromonal $\tau_{S_k}(o_j)$ de chaque objet candidat o_j est incrémenté de $\tau(o_i, o_j)$.

Mise à jour de la phéromone

- Quand toutes les fourmis auront fini la construction de leurs solutions, les traces de phéromone sont mises à jour.
- MAJ en deux étapes :
 - étape 1 : pour simuler l'évaporation, toutes les traces de phéromone sont diminuées en multipliant chaque composant phéromonal par un ratio de persistance $(1 - \rho)$ avec $0 \leq \rho \leq 1$. Notons que dans *Vertex-AK*, les composants phéromonaux sont associés aux objets de sorte que l'étape d'évaporation s'effectue en $O(n)$ tandis que dans *Path-AK* et *Edge-AK*, les composants phéromonaux sont associés aux couples d'objets de sorte que l'étape d'évaporation s'effectue en $O(n^2)$.

- étape 2 : la meilleure fourmi du cycle dépose de la phéromone.

Plus précisément, soit $S_k \in \{S_1, \dots, S_{nbAnts}\}$ la meilleure solution (celle ayant un profit maximal, les ex-aequo étant départagés aléatoirement) construite durant le cycle courant et S_{best} la meilleure solution construite depuis le l'exécution (y compris le cycle courant).

La quantité de phéromone déposée par la fourmi k est inversement proportionnelle à la différence de profit entre S_k et S_{best} , i.e. elle est égale à $\frac{1}{1 + \text{profit}(S_{best}) - \text{profit}(S_k)}$.

Cette quantité de phéromone est déposée sur les composants phéromonaux de S_k , i.e.

- dans *Vertex-AK*, elle est déposée sur chaque sommet de S_k ,
- dans *Path-AK*, elle est déposée sur les arcs du chemin visité par la fourmi k lors de la construction de S_k , i.e. sur tout couple de sommets (o_i, o_j) tel que o_j a été ajouté dans S_k juste après o_i .
- dans *Edge-AK*, elle est déposée sur chaque arête reliant deux sommets différents de S_k , i.e. sur la clique définie par S_k .

Comme pour la première étape, la complexité en temps de cette deuxième étape dépend de la version considérée : pour *Vertex-AK* et *Path-AK* elle s'effectue en $O(|S_k|)$ tandis que pour *Edge-AK* elle s'effectue en $O(|S_k|^2)$.

Influence des paramètres α et ρ sur la résolution :

- Quand on résout un problème d'optimisation combinatoire avec une approche heuristique, il s'agit de trouver un bon compromis entre deux objectifs relativement duaux :
 - 1- **intensifier la recherche** autour des zones de l'espace de recherche les plus prometteuses, qui sont généralement proches des meilleures solutions trouvées ;
 - 2- **diversifier la recherche** et favoriser l'exploration afin de découvrir de nouvelles et si possible meilleures zones de l'espace de recherche.
- Le comportement des fourmis par rapport à cette dualité entre intensification et diversification peut être influencé en modifiant les valeurs des paramètres.
- En particulier, la diversification peut être augmentée soit en **diminuant** la valeur du poids du **facteur phéromonal α** (de sorte que les fourmis deviennent moins sensibles aux traces phéromonales), soit en diminuant le taux d'évaporation ρ (de sorte que la phéromone s'évapore plus doucement et les écarts d'une trace à l'autre évoluent plus doucement).
- Lorsque la capacité d'exploration des fourmis est ainsi augmentée, on trouve généralement de meilleures solutions, mais en contrepartie ces solutions sont plus longues à trouver.

- On constate que cette influence des paramètres α et ρ sur le comportement des fourmis est **identique pour les trois versions proposées de *Ant-Knapsack***.

- Pour *EdgeAK* et *Vertex-AK* : quand on augmente l'intensification, en choisissant des valeurs telles que $\alpha = 2, \rho = 0.02$, *Edge-AK* trouve rapidement de bonnes solutions mais stagne plus vite sur des solutions légèrement moins bonnes.

- A l'inverse, en choisissant des valeurs pour α et ρ qui favorisent l'exploration, telles que $\alpha = 1, \rho = 0.01$, les fourmis trouvent de meilleures solutions en fin d'exécution, mais elles ont besoin de plus de cycles pour converger sur ces solutions.

- N.B. : lorsque la **phéromone n'est pas utilisée du tout, i.e. quand $\alpha = 0, \rho = 0$** , de sorte que l'algorithme se comporte comme un **algorithme glouton classique**, les solutions trouvées sont très nettement moins bonnes que lorsque la phéromone est utilisée.

I.13 Sujet 3 : TSP

- Pour un ensemble de n villes, le problème TSP (*Traveler Sales Person* = Voyageur de commerce) consiste à trouver un circuit fermé (aller-retour) de longueur minimale qui visite chaque ville une seule fois.

Distance :

- Avec d_{ij} : longueur du chemin entre les villes i et j , dans le cas du TSP, on a :

$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \quad \text{définit la distance euclidienne entre } i \text{ et } j.$$

- Le problème TSP est spécifié par un graphe (V, E) , où V (*Vertex* = *noeud*) de taille $|V| = N$ est l'ensemble des villes et E (*edge*) est l'ensemble des arêtes entre les villes,

→ Dans le cas du TSP euclidien, on considère un graphe fortement connexe.

- Si $b_i(t)$ ($i = 1, \dots, n$) est le nombre de fourmis dans la ville i au temps t

$$\rightarrow m = \sum_{i=1}^n b_i(t) \text{ le nombre total des fourmis.}$$

- Chaque fourmi est un simple **agent** avec les caractéristiques suivantes :
 - Elle choisit la ville suivante avec une probabilité qui est fonction de la distance ("visible") de la ville et la quantité de phéromone présente sur l'arête empruntée.
 - Pour éviter à la fourmi de tourner en rond, les villes déjà visitées par une fourmi sont marquées (dans une liste **tabou**, par une "recherche avec tabou", voir +loin).
 - **Quand une fourmi finit un tour**, elle dépose de la phéromone sur chaque arête visitée (i,j) .
 - On remarque donc une différence par rapport au problème du "plus court chemin" où le dépôt a lieu sur chaque arête empruntée.
 - C'est une *politique* alternative au moment du dépôt (V. un autre variant + loin).
 - Au temps t (ou étape t), chaque fourmi choisit la ville suivante où elle sera en $t+1$
 - **donc à $t+1$, la fourmi aura parcouru l'arête.**
- On appellera une **itération** de l'algorithme les m mouvements effectués par les m fourmis dans l'intervalle $(t, t+1)$.

- Notons $\tau_{ij}(t)$ l'intensité de la phéromone (aka. *trail*) sur l'arête (i,j) en temps t .
- Après n itérations de l'algorithme (appelé un *cycle*), chaque fourmi a complété un tour.
→ Le nbr de villes étant fixe, un tour de chacune des fourmis comprend n pas d'itération.

MAJ de la phéromone (le nombre de noeuds $|V|$ est identique pour toutes les fourmis) :

- Puisque **les fourmis font toutes un tour en même temps**, à ce point-là (fin de tour), l'intensité de la phéromone est mise à jour (update, indép de n) par :

$$\tau_{ij}(t + n) = \rho \cdot \tau_{ij}(t) + \Delta_{\tau_{ij}} \quad (\text{I.1})$$

où ρ est un coefficient tel que $(1 - \rho)$ représente l'**évaporation** de la phéromone entre t et $t+n$, avec

$$\Delta_{\tau_{ij}} = \sum_{k=1}^m \Delta_{\tau_{ij}}^k \quad (\text{I.2})$$

où $\Delta_{\tau_{ij}}^k$ est la **quantité de phéromone par unité de longueur** (des arêtes, p. ex. mètre) déposée sur l'arête (i,j) par la k -ème fourmi entre t et $t+n$.

- Cette quantité est donnée par :

$$\Delta_{\tau_{ij}}^k = \begin{cases} \frac{Q}{L_k} & \text{si le } k\text{-ème fourmi utilise l'arête } (i,j) \text{ dans son "tour" (entre les temps } t \text{ and } t+n) \\ 0 & \text{sinon} \end{cases} \quad (\text{I.3})$$

où Q est une constante (paramètre de l'algorithme) et L_k est la longueur de la tournée (du tour) de la k -ème fourmi ;

Le coefficient $\rho < 1$ permet d'éviter une accumulation illimitée de phéromone.

- Rappel : $\tau_{ij}(t)$ est l'intensité de la phéromone (aka. *trail*) sur l'arête (i,j) en temps t .
 - L'intensité $\tau_{ij}(0)$ de la phéromone au temps 0 est fixée arbitrairement.
 - Pour chaque arête, (i,j) , on fixera $\tau_{ij}(0) = c$, où c est une petite constante positive.
- 👉 Notons que si $|V|$ est identique pour toute fourmi, la somme des longueurs des arêtes ne l'est pas !

Choix d'arête (transition d'une ville à une autre) :

- On appelle "visibilité" η_{ij} la fraction $1/d_{ij}$ (l'inverse de la distance entre 2 villes i, j).
 ➡ Cette quantité n'est pas modifiée pendant le fonctionnement de l'algorithme.
 ➔ La "visibilité" est également appelée "l'heuristique" associée à l'arête ij (cf. le BE).
- La probabilité de transition de la ville i à la ville j pour la k -ème fourmi est définie par :

$$P_{ij}^k(t) = \begin{cases} \frac{[\tau_{ij}(t)]^\alpha \cdot [\eta_{ij}]^\beta}{\sum_{k \in allowed_k} [\tau_{ik}(t)]^\alpha \cdot [\eta_{ik}]^\beta} & \text{Si } j \in allowed_k \\ 0 & \text{Sinon} \end{cases} \quad (I.4)$$

où $allowed_k = \{V - tabou_k\}$ (toutes les villes sauf les villes prohibées, voir ci-après),

➔ $allowed_k$ = les villes non encore visitées par la fourmi k

α et β sont des paramètres qui contrôlent importance relative de la phéromone vs. la "visibilité".

Remarque :

- La probabilité de la transition $P_{ij}^k(t)$ ci-dessus est **un compromis** entre :
 - la **visibilité** (qui, si seule, nous encouragerait à préférer les villes les plus proches et nous mènerait à une stratégie *greedy*) et
 - l'**intensité** de la phéromone en temps t (qui, si seule, nous dirait que sur l'arête (i,j) , il y a eu beaucoup de trafic et donc elle est préférable ; ce qui conduirait à implanter un processus appelé *autocatalytique*) :
 - ➔ c-à-d. une réaction (chimique) dont le catalyseur est un des produits de la réaction.

👉 Remarques sur "tabou" :

- 1- L'expression $\{V - tabou_k\}$ devient plus claire quand vous aurez regardé la section ?? page ??.
- 2- Pour simplifier, disons que nous enregistrons les villes par où une fourmi est déjà passée (concept utilisé dans le cas du sujet "plus court chemin" où la fourmi est capable de reconnaître sa propre Phéromone pour ne pas tourner en rond).
 - ➔ Simplement, ici, **on ne refusera pas systématiquement de reprendre une ville déjà visitée!**
- 3- Comparé à la recherche taboue "habituelle", on a ici un algo avec une heuristique constructive.

Tout en étant dans le cadre général de la recherche "taboue", les différences entre l'approche utilisée ici et les algos "habituels" de recherche taboue sont :

 - ici toute fonction d'aspiration est absente ;
 - les éléments enregistrés dans la liste taboue sont ici différents ;
 - on ne fait pas de permutation (comme dans le cas de recherche taboue générale) ;
 - présence de noeuds dans l'algorithme présent, ...

I.13.1 Algorithmme

L'**algorithme** (dit *cycle-fourmi*) se décrit comme suit :

- Au temps zéro (initialisation), les fourmis sont positionnées sur différentes villes et on initialise les valeurs $\tau_{ij}(0)$ comme l'intensité sur les bords des arêtes.
- Le premier élément de la liste taboue de chaque fourmi est sa ville de départ.
- Par la suite, avec les paramètres α et β (cf. formule 4), chaque fourmi se déplace de la ville i à la ville j en choisissant la ville suivante avec une probabilité qui est une fonction de deux mesures :
 - 1 ◦ L'intensité de la Phéromone (a.k.a. *trail*) $\tau_{ij}(t)$ qui donne des informations sur le nombre de fourmis qui ont choisi cette même arête (i, j) dans le passé ;
 - 2 ◦ La visibilité η_{ij} qui dit que plus une ville est proche, plus elle est choisie.
- N.B.** : si on a $\alpha = 0$, la quantité de phéromone n'est plus déterminante et on sera dans un algorithme *glouton* stochastique avec plusieurs points de départ.
- Après n itérations, toutes les fourmis ont terminé une tournée et leurs listes de taboues seront pleines ;
- A ce stade pour chaque fourmi k , la valeur de L_k est calculée et les valeurs $\Delta_{\tau_{ij}}^k$ sont MAJ selon la formule (3).

- A la fin du cycle, la route la plus courte trouvée par les fourmis (i.e., $\underset{k}{\operatorname{argmin}} L_k, k = 1..m$) est sauvegardée et toutes les listes taboues sont vidées.

- **Ce processus est répété jusqu'à atteindre :**

- un nbr maximum d'itérations (paramètre),
- un nbr prédéfini de cycles NC_{MAX} ,
- toutes les fourmis font la même tournée !

Appelons ce dernier cas *stagnation* ou l'algorithme ne trouve plus de solution alternative et on arrête de chercher d'autres solutions.

L'algorithme de principe :

1. Initialisations :

$t \leftarrow 0$ { t compteur du temps / incrément}

$NC \leftarrow 0$ { NC compteur de cycles}

Pour chaque arête (i,j) Faire :

$\tau_{ij}(t) = c$ (la valeur initiale de phéromone)

$\Delta\tau_{ij} \leftarrow 0$

Placer les m fourmis sur les N noeuds (villes)

2. $s \leftarrow 1$ { s est l'indice dans la liste taboue }

Pour chaque fourmi $k \leftarrow 1..m$ Faire : (donc pour chaque fourmi)

Placer la ville de départ de la k -ème fourmi dans $tabou_k(s)$ (pour éviter de la re-choisir!)

3. Tantque la liste taboue n'est pas pleine faire : (cette étape sera répétée $(n-1)$ fois)

$s \leftarrow s + 1$

Pour chaque fourmi $k \leftarrow 1..m$ Faire :

Choisir la prochaine ville j avec la probabilité $P_{ij}^k(t)$ donnée par l'équation (4)

{la k -ème fourmi est actuellement dans la ville $i = tabou_k(s-1)$ au temps t }

Déplacer la k -ème fourmi dans la ville j (qu'elle vient de choisir)

Insérer la ville j dans la liste $tabou_k(s)$

4. Pour chaque fourmi $k \leftarrow 1 .. m$ Faire :

Déplacer la k -ème fourmi de la liste $tabou_k(n)$ vers $tabou_k(1)$

Calculer la longueur L_k de la tournée décrite par $tabou_k$

MAJ le trajet le plus court (so far!)

Pour chaque arête (i,j) Faire :

Pour chaque fourmi $k \leftarrow 1 .. m$ Faire :

$$\Delta_{\tau_{ij}}^k = \begin{cases} \frac{Q}{L_k} & \text{if } (i,j) \in \text{tour described by } tabou_k \\ 0 & \text{sinon} \end{cases}$$

$$\Delta_{\tau_{ij}} \leftarrow \Delta_{\tau_{ij}} + \Delta_{\tau_{ij}}^k$$

5. Pour chaque fourmi arête (i,j) Faire : calculer $\tau_{ij}(t+n)$ selon l'équation $\tau_{ij}(t+n) = \rho \cdot \tau_{ij}(t) + \Delta_{\tau_{ij}}$

$$t \leftarrow t + n$$

$$NC \leftarrow NC + 1$$

Pour toute arête (i,j) : $\Delta_{\tau_{ij}} \leftarrow 0$

☞ Dans la littérature, il y a de multiples variants de règles de MAJ de phéromone (en TSP).

☞ Critères d'arrêt ou continuation :

6. Si $(NC < NC_{MAX})$ ET (pas en état de *stagnation*) [la situation où les trajets sont les mêmes pour toute fourmi]

Alors

Vider toutes les listes taboues

Aller à l'étape 2

Sinon

Afficher "shortest tour" (final)

Fin.

I.13.2 La complexité de l'algorithme

La complexité de cet algorithme est $O(NC.n^2.m)$ s'il est arrêté après NC cycles.

Plus précisément, l'étape 1 est $O(n^2 + m)$, étape 2 $O(m)$, étape 3 $O(n^2.m)$, étape 4 $O(n^2.m)$, étape 5 $O(n^2)$, et l'étape 6 est $O(n.m)$,

☞ Il a été montré que cet algorithme est $O(NC.n^3)$

I.13.3 Deux solutions alternatives au TSP

- Deux solutions alternatives au TSP (à expérimenter) :

Elles diffèrent dans la façon dont la quantité de phéromone est mise à jour.

- algorithme *ant-densité* : MAJ progressive de la phéromone
- algorithme *ant-quantité* : MAJ progressive mais pondérée de la phéromone
- Dans ces deux modèles, chaque fourmi met à jour la quantité de phéromone à chaque étape, sans attendre la fin de la visite (point remarqué plus haut).
- Dans le modèle *ant-densité*, une quantité Q de phéromone est laissée sur le bord (i, j) chaque fois qu'une fourmi

Expérimentations :

- Les paramètres importants qui affectent directement ou indirectement le calcul de la probabilité dans la formule (4) :
 - α : l'importance relative de la phéromone (intensité), $\alpha \geq 0$
 - β : l'importance relative de la distance d_{ij} ("visibilité"), $\beta \geq 0$
 - ρ : la persistance de la phéromone, $0 \leq \rho < 1$ ($1 - \rho$ est l'évaporation de la phéromone);
 - Q : la quantité constante liée à la quantité de la phéromone déposée par les fourmis.
- **Le nombre m de fourmis est fixé égal au nombre N de villes.**
- Conseil : pour rechercher les meilleures paramètres, faire varier un paramètre tout en **fixant** tous les autres.
 - ➔ Mieux : répéter p. ex. en dix simulations pour chaque paramètre puis choisir la moyenne.

Les valeurs par défaut des paramètres : $\alpha = 1, \beta = 1, \rho = 0.5, Q = 100$.

- Dans chaque expérience, une seule des valeurs doit changer, sauf pour α et β , qui doivent être testés sur différents ensembles de valeurs :

$$\alpha \in \{0, 0.5, 1, 2, 5\}, \beta \in \{0, 1, 2, 5\}, \rho \in \{0, 3, 0.5, 0.7, 0.9, 0.999\} \text{ et } Q \in \{1, 100, 10000\}.$$

- Dans tous les tests, fixer $NC_{MAX} = 5000$ cycles puis faire des moyennes sur dix essais.
- Le paramètre Q se montre d'intérêt négligeable dans les tests.

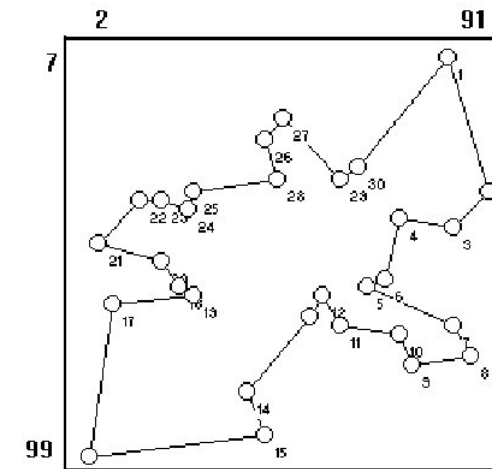
Exemple de résultats :

- Exemple de résultats présentés dans le tableau suivant.



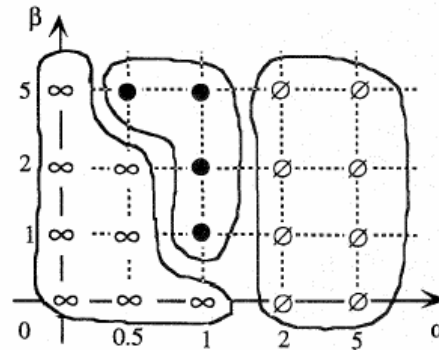
- Le graphe testé comporte 30 villes ($N = 30$) :

A droite, une exécution avec un algorithme Greedy + GA (Thanks to Victor V Miagkikh) :



Détails des bons résultats de l'algorithme ant-cycle :

- Différentes combinaisons de paramètres α et β avec $NC_MAX = 2500$ sont montrés dans le grid-search suivant où chaque couple de paramètre a été exécuté 10 fois puis moyenné.



- Situations repérées avec \circ dans la grille : mauvaises solutions et *stagnation* pour des valeurs élevées de α .
- Mauvaises solutions et pas de stagnation pour des valeurs basses de α : peu d'importance accordée à la phéromone (situations repérées avec ∞)
- Bonnes solutions : (voire très bonnes) pour α et β dans la région du centre de la figure (repérées par \bullet). On peut y lire les couples ayant donnés ces résultats.

- Ces résultats montrent le rôle important de α où la quantité de la phéromone et le fait de choisir les arêtes empruntées par les autres fourmis dans le passée.
- L'algorithme ant-cycle s'est montré efficace pour différents types de graphes avec une vitesse assez rapide.
- Par contre, il ne montre pas de situation de *stagnation* et continue à chercher d'autres solutions.
- Dans cet algorithme (ant-cycle), les paramètres optimaux se montrent peu sensibles à la taille du graphe.

I.14 Sujet 4 : Bus Allocation Problem (BAP)

☞ La lecture de la section suivante donnant une solution algorithmique au problème BAP peut aider à mieux comprendre la description de la solution avec les colonies de fourmis.

La solution au problème de BAP (Bus Allocation Problem) tient compte d'une liste de lignes de bus et chaque ligne de bus se compose d'une liste d'arrêts (de bus).

Il semble donc naturel d'essayer de résoudre ce BAP avec de multiples colonies de fourmis (*Multiple Ant Colony Systems : MACS*).

On construit le MACS de manière à ce que chaque ligne de bus soit représentée par une colonie de fourmis (Ant Colony System : ACS) séparé.

Cela signifie que la quantité de phéromone de chaque ACS sera mise à jour séparément.

Si nous avons une instance du BAP composé de n arrêts de bus et m lignes de bus, nous initialisons un MACS composé de m ACS.

Chaque ACS sera composé d'un même nombre de fourmis r .

A la fin de chaque itération, r solutions (chacune composée de m lignes de bus) auront été construites. Car chaque fourmi construit une solution complète contenant tous les stops.

➔ Chaque ACS a une règle de transition différente. Il utilise une règle LOCALE de MAJ de phéromone

différente ainsi qu'une règle différente globale (*global update rule*).

→ De tel systèmes peuvent traiter plus efficacement le problème du TSP.

I.14.1 Génération de solutions

Toutes les k -ième fourmis de chacune des m colonies construiront une solution ensemble.

Soit J_k la liste contenant tous arrêt de bus qui doivent encore être visités pour toutes les k -ième fourmis de chaque colonie.

S_k désigne la solution formée par toutes les k -ième fourmis de toutes les colonies.

L_k désigne le **att** (Average Travel Time, voir ci-dessous) de la solution formée par toutes les k -ième fourmis de toutes les colonies.

S_{bg} désigne la solution globale la mieux trouvée et L_{gb} désigne l'**att** de la meilleure solution globale trouvée.

Nous appellerons c_{ij} la visibilité du l'arête (i, j) qui est définie comme l'inverse du temps en secondes pour aller de l'arrêt de bus i à l'arrêt de bus j .

Étant donné un BAP, nous pouvons calculer les valeurs de visibilité en prenant l'inverse du temps nécessaire à un bus pour parcourir le distances (euclidiennes) entre tous les arrêts de bus.

Le prochain arrêt de bus où se déplacer est toujours décidé par les équations 3.3 et 3.4.

Une mise à jour locale est toujours effectuée en utilisant l'équation 3.5 et une mise à jour globale est toujours effectuée à l'aide de l'équation 3.6.

$$j = \begin{cases} \arg \max_{h \in J_k} \{[\tau_{ih}^l] \cdot [\eta_{ih}^l]^\beta\} & \text{if } q \leq q_0 \\ S & \text{otherwise} \end{cases}$$

FIGURE I.9 – équation 3.3 (OK) : choix du prochain arrêt de bus j : exploitation (cas 1) et exploration (cas 2 : choix de S)

$$p_{ij}^{kl} = \begin{cases} \frac{[\tau_{ij}^l] \cdot [\eta_{ij}^l]^\beta}{\sum_{h \in J_k} [\tau_{ih}^l] \cdot [\eta_{ih}^l]^\beta} & \text{if } j \in J_k \\ 0 & \text{otherwise} \end{cases}$$

FIGURE I.10 – Equation 3.4(OK)

$$\tau_{ij}^l = (1 - \rho) \cdot \tau_{ij}^l + \rho \cdot \tau_0$$

FIGURE I.11 – équation 3.5 : update locale de phéromone

$$\tau_{ij}^l = (1 - \alpha) \cdot \tau_{ij}^l + \alpha \cdot \Delta\tau_{ij}^l$$

where $\Delta\tau_{ij}^l = \begin{cases} (L_{gb})^{-1} & \text{if edge (i,j) } \in \text{ colony } l \text{ of } S_{gb} \\ 0 & \text{otherwise} \end{cases}$

FIGURE I.12 – équation 3.6 : update global de phéromone

I.14.2 Evaluation des solutions

Contrairement au problème TSP où la durée d'une tournée est immédiatement apparente, l'att d'un BAP doit être calculé.

Afin de pouvoir calculer l'att, nous avons besoin de la matrice T (transition matrice) et matrice U (temps nécessaire pour aller d'un arrêt de bus à un autre en utilisant une solution).

La matrice U peut être calculée à l'aide d'une solution, v (vitesse moyenne du bus) et de la matrice D (vitesse euclidienne distances).

La figure 3.7 ci-dessous contient le pseudo-code pour calculer la matrice U .

En utilisant la matrice T , la matrice U et l'équation 2.1, on peut calculer **att**.


```

For all buslines  $bl_x$  do
  For all busstops  $i \in bl_x$  do
    For all buslines  $bl_y$  do
      If  $bl_x = bl_y$ 
        For all busstops  $j \in bl_y$  do
          If  $i \neq j$  :
             $u_{ij} = \text{time from } i \text{ to } j \text{ using } bl_x$ 
      Else if  $i \neq \text{mainbusstop}$  :
         $\text{timetomain} = \text{time from } i \text{ to mainbusstop using } bl_x$ 
         $\text{timechangebus} = \text{timetomain mod } (3600 / f)$ 
        For all busstops  $j \in bl_y$  do
          If  $j \neq \text{mainbusstop}$ 
             $\text{arrival} = \text{time from start } (bl_y) \text{ to mainbusstop using } bl_y$ 
            If  $\text{timechangebus} + c > \text{arrival}$ 
               $u_{ij} = (3600/f) + \text{arrival} + \text{time from mainbusstop to } j \text{ using } bl_y$ 
            Else
               $u_{ij} = \text{arrival} + \text{time from mainbusstop to } j \text{ using } bl_y$ 

```

Étant donné une solution optimale d'un BAP, l'ordre relatif des arrêts de bus ainsi que la position absolue des arrêts de bus sont importants (cf. TSP).

Compte tenu de cette propriété, il peut être intéressant de étudier si, étant donné un BAP, certains arrêts de bus seraient mieux pour démarrer une ligne de bus que d'autres.

Nous avons développé une extension d'un MACS qui peut tenir compte du fait que certains arrêts de bus seront de meilleurs arrêts de bus pour démarrer une ligne de bus que d'autres.

Au lieu de choisir l'arrêt de bus pour démarrer une ligne de bus au hasard, nous avons donné à

chaque arrêt de bus un niveau de phéromone. Cette phéromone influencera la probabilité qu'un arrêt de bus soit choisi comme arrêt de départ.

Comme pour les niveaux de phéromones des arêtes, les niveaux de phéromones des arrêts de bus sont séparés pour chaque colonie. Si un arrêt de bus a un niveau élevé de phéromones dans une colonie, la probabilité que cet arrêt de bus soit choisi comme l'arrêt de bus de départ de cette colonie sera grande.

La fourmi k d'ACS l choisira arrêt de bus i comme arrêt de départ de sa ligne de bus selon la distribution de probabilité de équation 3.9. Le niveau de phéromone à l'arrêt de bus i est représenté par τ_i .

$$p_{kl} = \begin{cases} \frac{\tau_i}{\sum_{j \in J_k} \tau_j} & \text{if } i \in J_k \\ 0 & \text{otherwise} \end{cases}$$

FIGURE I.13 – équation 3.9 : Choix de l'arrêt de bus initial

Chaque fois qu'une fourmi a sélectionné un arrêt de bus comme arrêt de départ de sa ligne de bus, elle effectuer une mise à jour locale des phéromones sur cet arrêt de bus en utilisant l'équation 3.10. Cela rend le dernier arrêt de bus sélectionné un peu moins souhaitable comme arrêt de bus de

départ pour les autres fourmis du même AEC.

Le paramètre ρ ($0 \dots 1$) représente le taux d'évaporation. Après chaque itération, une mise à jour globale est effectuée à l'aide de l'équation 3.11. Le paramètre ρ détermine le taux d'évaporation.

$$\tau_{ij}^l = (1 - \rho) \cdot \tau_{ij}^l + \rho \cdot \tau_0$$

FIGURE I.14 – équation 3.10 : MAJ local de phéromone

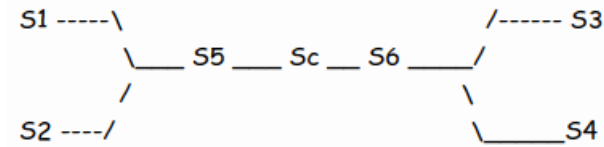
$$\tau_{ij}^l = (1 - \alpha) \cdot \tau_{ij}^l + \alpha \cdot \Delta\tau_{ij}^l$$

where $\Delta\tau_{ij}^l = \begin{cases} (L_{gb})^{-1} & \text{if edge (i,j) } \in \text{ colony } l \text{ of } S_{gb} \\ 0 & \text{otherwise} \end{cases}$

FIGURE I.15 – équation 3.11 : MAJ globale de phéromone

Quelques remarques et indications :

Soit une configuration d'arrêts (SC est l'arrêt central) :



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

Comment faire pour répartir les stops puisque les 2 colonies ne doivent pas utiliser exactement les mêmes stops.

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 :

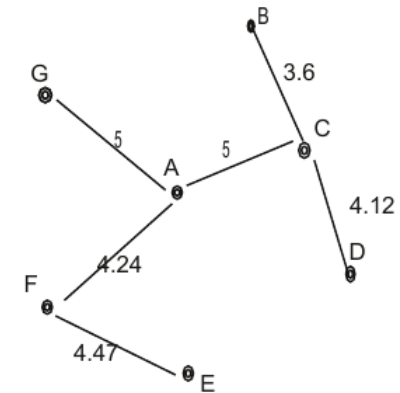
Supposons que C_1 (colonie 1) part de l'arrêt central Sc et arrive à $S3$. Comment faire pour ensuite couvrir les stops (p. ex.) $S2$ et $S5$ pour C_1 ? Peut-on revenir sur ses pas ? Le bus revient-il en marche arrière ?

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 $S3 \rightarrow S1$ en faisant un détour).

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

3- Construire un MST puis en déduire deux (sous) arbres (= 2 lignes) en veillant à ce que les deux passent par Sc .

Le MST aura été construit par les colonies de fourmis !



Indications :

Le graphe de la ville du début du sujet (avec 7 arrêts) par la méthode *Kruskal* donne le MST ci-dessus. Les valeurs sur les arêtes sont les distances euclidiennes.

On note une somme totale des arrêts = 27 dans ce MST (vs. 37 pour la solution proposée dans le sujet pour le même graphe)

Il faut maintenant décider comment répartir les 2 lignes de bus et y partager les arrêts (l'arrêt A sera en commun). Une solution peut être : B-C-A-G et D-C-A-F-E. L'arrêt C sera également en commun mais rien ne s'y oppose.

→ Ensuite, il faudra recalculer la matrice U et le temps moyen d'attente d'un trajet quelconque pour pouvoir comparer avec la solution proposée.

I.15 Une solution algorithmique au problème BAP

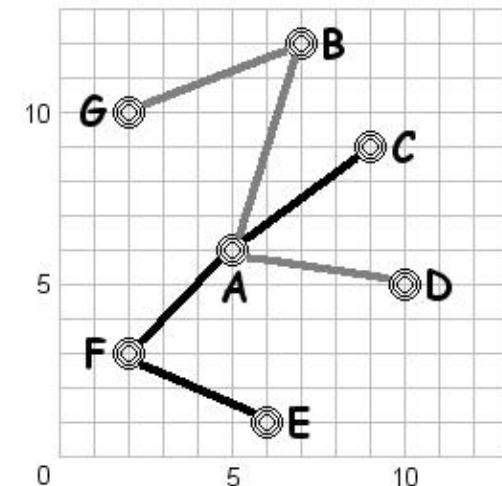
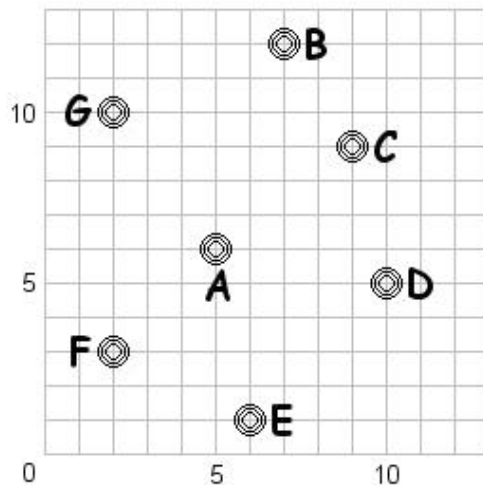
Le modèle présenté ci-dessous n'est pas une solution avec la colonie de fourmis. Par contre, ces descriptions seront très utiles et serviront de base pour comprendre la solution avec des colonies de fourmis présentée dans la section précédente.

Ci-dessous, nous définissons le problème BAP avec une proposition de résolution (optimisation) sans colonie de fourmis.

Les indications pour la mise en oeuvre d'une colonie de fourmis sont données dans la section précédente.

I.15.1 Description

- 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.

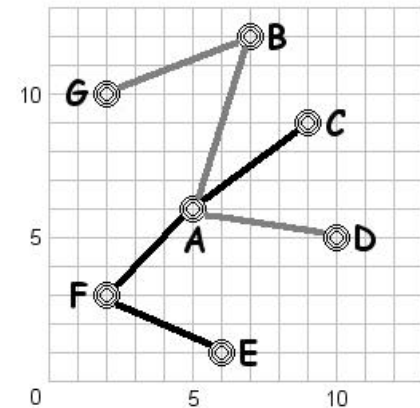


- Les lignes de bus seront construites en tenant compte de plusieurs contraintes :
 - distance, vitesse de bus (temps d'attente / fluidité / correspondances)
 - nombre de passagers, capacité des bus, fréquences de bus, .. Voir plus loin.

- 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.15.2 Définition (pour 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.
 - La fréquence des bus est le nombre de fois par heure où tous les bus commencent à circuler est une paramètre fixe pour tous les bus.
-
- On suppose que les bus ont une capacité illimitée.
 - Pour l'exemple précédent, initialement, 4 bus commencent à circuler en même temps depuis les arêtes G, D, C et E (les 4 extrémités des deux lignes).



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

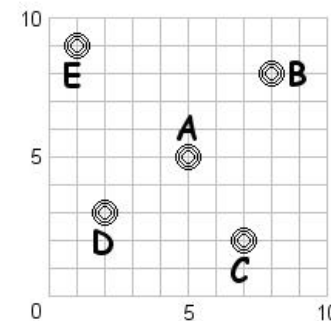
v : vitesse moyenne du bus (m/s), ici, $v = 7 \text{ m./sec}$

f : fréquence du bus (nb. fois/h), ici $f = 15$ fois par heure (toutes les 240 sec.)

c : le temps nécessaire pour changer de bus (correspondance), ici 20 sec..

- 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.



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

- 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" que devra être minimisée (fonction objective).

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

I.15.2.1 Calcul de ATT (et exemple)

- Pour calculer att , on utilise une matrice qui contient tous les temps de voyage pour toute transition possible d'un arrêt à un autre. Ces temps contiennent également les temps d'attente d'arrivée de bus.

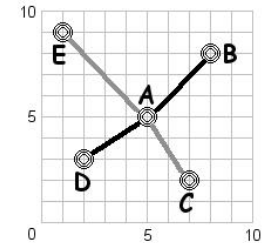
En suite, on multiplie chaque temps de voyage par le nombre de passagers qui font cette transition pour obtenir les temps de voyage cumulatifs puis, on divise chaque temps cumulatif pour un arrêt de départ par le nombre de voyageurs ayant cet arrêt comme départ pour obtenir un temps moyen.

Si on additionne tous ces temps moyens et que l'on le divise par le nombre total d'arrêts, on aura calculé att .

- 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) :

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

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



Explications de la matrice U (la matrice T est rappelée) :

- 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).

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

- 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) :

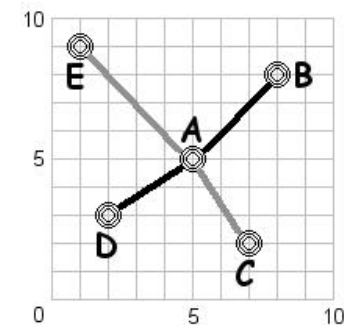
- le trajet de E à A prend 80,81 secondes et D à A prend 51,51 secondes.

- Donc $51,51 + 20$ secondes (le param. c) est inférieur à 80,81 secondes.

- Quand il 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, cela aura pris $240 + 60.61$ (B à A) + 51.51 (A à D) = 352.117 sec. Les 240 sec. viennent de $\frac{3600}{f}, f = 15$.
- Utilisant la matrice U , la matrice T et l'équation de "att", nous sommes en mesure de calculer att . La valeur moyenne de att de cette solution est 155,491 secondes.
- Donc, il faut en moyenne à une personne 155,491 secondes pour arriver à destination.

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

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



I.16 Sujet 5 : Ramassage Bus Scolaire

Solution Colonie de fourmis : construction de **DBS** (*Dynamic chain Bus Stop*)

- On construit une route (ligne) r_k
- On y ajoute un arrêt i en l'insérant à toutes les places de la route en construction
- On y insère ainsi des conflits dans la route en construction qui seront résolus avec l'ajout d'autres arrêts
- Il est parfois impossible d'ajouter un arrêt i sans conflit mais après un certain temps, on peut trouver des places plus avantageuses.
- Dans la méthode constructive on construit un itinéraire r_k en insérant des arrêts dans toutes les positions, par déplacement des éléments déjà placés tout en préservant globalement contraintes du problème. On crée ainsi une chaîne dynamique (DBS).
- Pour la méthode de construction dynamique, la logique de la construction de la solution est inversée.
- À chaque étape, une fourmi doit d'abord choisir au hasard une route r_k . Ensuite, la meilleure position *BESTPOS* dans lequel un arrêt de bus i de cet itinéraire sera inséré est sélectionnée selon une règle pseudo-aléatoire de transition modifiée selon la formule ci-dessous. F
- Formellement, si au cours de la t -ième itération, la k ième fourmi est située à l'arrêt de bus i , le prochain arrêt de bus j est choisi en fonction de la distribution de probabilité sur l'ensemble des arrêts de bus non visités

$$p_{ij}(t) == \begin{cases} \frac{[PQ_{ij}]^{\alpha} \cdot [VIS_{ij}]^{\beta}}{\sum_{fourmi \in (n-DCC)} [PQ_{ij}]^{\alpha} \cdot [VIS_{ij}]^{\beta}} & \text{Si } j \in (n - DCC) \\ 0 & \text{sinon} \end{cases} \quad (I.5)$$

Où $(n - DBSC)$: nbr d'arrêts total - les arrêts déjà choisis

La fourmi examine le choix d'aller de l'arrêt i à j en examinant la quantité de phéromone PQ_{ij} associée au coefficient α .

Également, on a la "visibilité" VIS_{ij} pour le même arrêt i à j associé au coefficient de β .

Dans ce cas, on définit $VIS_{ij} = 1/d_{ij}$.

Il est important de choisir l'arrêt i selon la meilleure position *BESTPOS* .

L'ensemble des choix valides pour une fourmi est $S = (n - DBS)$. Avec une probabilité $0 \leq p_0 \leq 1$ (un paramètre de l'algorithme), la fourmi a choisi l'arrêt de bus $i \in S$ qui maximise $[PQ_{ij}]^{\alpha} \cdot [VIS_{ij}]^{\beta}$.

MAJ de la phéromone : la règle de mise à jour sera

$$PQ_{ij} = (1 - \rho)PQ_{ij} + \rho PQ_{ij}(0)$$

Avec $0 \leq \rho \leq 1$ le coefficient d'évaporation $((1 - \rho))$ et $PQ_{ij}(0)$ la quantité initiale de phéromone

I.16.1 Recherche locale avec voisinage variable

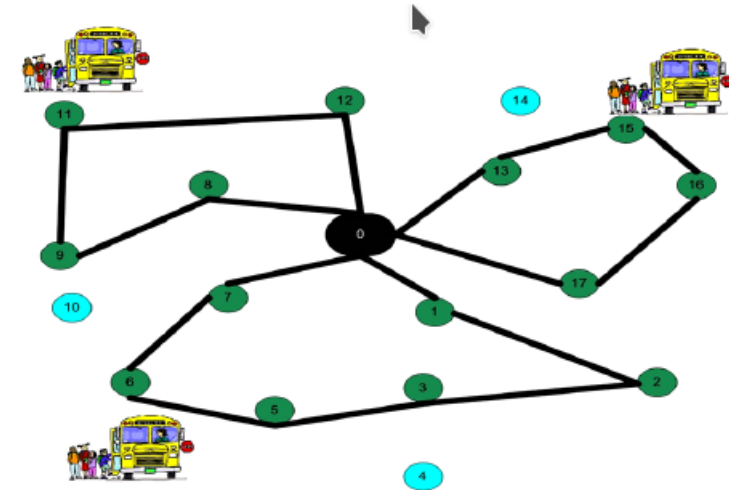
- A compléter

I.17 Sujet 5 : Ramassage Bus Scolaire

☞ Ci-dessous, une solution qui n'est pas une solution Colonie de fourmis.

Les paramètres du problème :

- C_{ij} : Le coût du voyage empruntant l'arc (i, j)
- K : Nombre de bus
- Q : Capacité des bus
- V : Ensemble de tous les arrêts (stops potentiels)
- A : Ensemble des arcs entre les arrêts
- S : Ensemble des étudiants
- S_{li} : variables booléenne indiquant si l'étudiant l peut marcher jsq'à l'arrêt i ou non.



Variables de Décision :

x_{ijk} = Le nombre de fois où le bus k traverse les arcs de i à j (un ou plusieurs)

y_{ik} = 1 si le bus k s'arrête à l'arrêt i ; 0 sinon

z_{ilk} = 1 si l'étudiant l monte dans le bus k à l'arrêt i ; 0 sinon

Une idée d'une solution :

Soit 10 arrêts de bus et 3 bus.

Une solution possible (pas pour la figure ci-dessus. V. + bas) :

- Route 1 : arrêts 1, 5, 6, 9
- Route 2 : arrêts 2, 4, 8
- Route 3 : arrêts 3, 7, 10

1	2	3	4	5	6	7	8	9	10
V1	V2	V3	V2	V1	V1	V3	V2	V1	V3

Une première idée : algorithme du plus proche voisin

- 1 : Commencer avec n'importe quel noeud comme le début d'une ligne (de bus)
- 2 : Trouver un noeud (non encore visité) qui est le plus proche du dernier noeud ajouté
- 3 : Trouver le plus grand arrêt de bus avec le plus grand nombre de personnes ;
l'ajouter au trajet
- 4 : Répéter l'étape 2 jsq'à l'ajout de tous les noeuds
- 5 : Rejoindre le premier et le dernier noeuds du trajet

- On utilisera une colonies de fourmis (ACO) ...



Définitions (habituelles) :

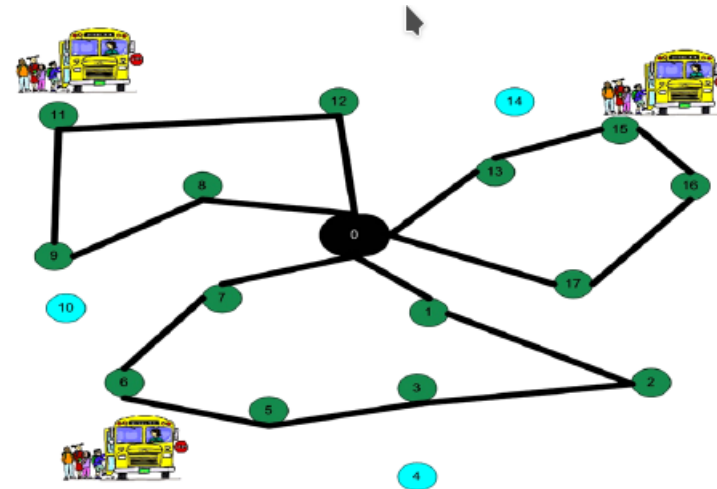
Le problème d'allocation de lignes de Bus :

- $G = (V, A)$: un graphe orienté avec $V = \{1, \dots, n\}$ ensemble de noeuds et $A = \{(i, j), 1 \leq i, j \leq n\}$ ensemble des arcs.
- Les Bus sont alloués de manière centralisée (arrêt central = l'École).
Ils ramassent les étudiants depuis n arrêts et les conduisent à l'École.
- Le nombre d'étudiants qui attendent à un arrêt i est $q_i, q_i > 0, i = 1..n$.
- La capacité de chaque Bus est de Q étudiants et on a $q_i < Q$.
- **L'objectif est de réduire le coût en réduisant le nombre de Bus nécessaires (nbr de lignes)**
- Ce problème a modèle en Programmation Mathématique connu sous le noms *BRP : Bus Routing problem* (modèle omis ici).

Pour la figure ci-contre, l'instance du BRP a 17 arrêts et 3 Bus.

Une solution possible :

- Route 1 (Bus No 1) : arrêts 1, 2, 3, 5, 6, 7 ;
- Route 2 : arrêts 8, 9, 11, 12 ;
- Route 3 : arrêts 13, 15, 16, 17



I.18 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.18.1 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

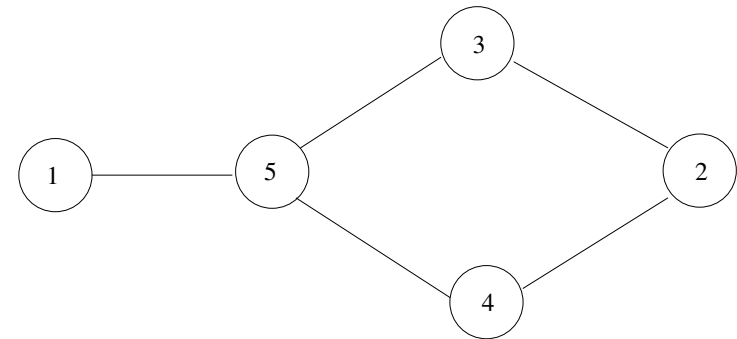
- Chaque noeud a un degré D de connexions et lâchera D fourmis locales, une par voisin.
- Au départ, on active chaque noeud en parallèle avec les autres.
- Chaque noeud choisit une couleur C et envoie une fourmi par voisin pour lui **dire de ne pas prendre** C .

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

- Le messenger 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.18.2 Exemple

- Soit ce graphe avec 5 noeuds et un ensemble de $k \leq 5$ couleurs $\{R, V, B, \dots\}$
- 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 \bar{R}_{dest} aux voisins	R \bar{R}_5	R $\bar{R}_{3,4}$	R $\bar{R}_{2,5}$	R $\bar{R}_{2,5}$	R $\bar{R}_{1,3,4}$
Comptage (maxi même \bar{C} de $\neq dest$) → Choix de 5 (ne prend pas R) Et envoi de 3 fourmis \bar{V}_5 aux voisins	1 \bar{V}_5	2	2 \bar{V}_5	2 \bar{V}_5	3 prend V
Etat messages (nb. fourmis présentes + messages)	\bar{R}_5, \bar{V}_5	$\bar{R}_{3,4}$	$\bar{R}_{2,5}, \bar{V}_5$	$\bar{R}_{2,5}, \bar{V}_5$	V
Annulation du message \bar{R}_5 (5 a pris V) → Venant de 5, le nouv. messenger \bar{V}_5 tue le préc \bar{R}_5	\bar{V}_5	$\bar{R}_{3,4}$	\bar{R}_2, \bar{V}_5	\bar{R}_2, \bar{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 \bar{V}_2 aux voisins	1	2 V	1 \bar{V}_2	1 \bar{V}_2	
Bilan : (\bar{V}_2/\bar{V}_5 annule \bar{R}_2 / \bar{R}_5 car 2 et 5 ont pris V) Les 3 noeuds restants sont déconnectés et prennent R	$\bar{V}_{2,5}$	V	$\bar{V}_{2,5}$	$\bar{V}_{2,5}$	V

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

I.18.3 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 "p" 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.19 A rendre : modalités

- Pour 23-24, mettre ce mail que j'ai envoyé en mars 2023 aux élèves pour clarifier. Les barèmes sont à la page svte.
- 1. "Plus court chemin" par les colonies de fourmis : détaillé dans le pdf du BE + les ressources sur Moodle ; Barème détaillé dans le pdf du BE (page 76).
- 2. "Allocation de lignes de BUS" (BAP) : voir le pdf du BE + les ressources pdf (articles) sur Moodle ; Barème détaillé dans le pdf du BE (page 76).
- 3. "Voyageur de commerce" (TSP) : noté sur 20 (+Bonus si interface graphique). Détails complets dans le pdf du cours 1, page 54.
- 4. "Sac à dos" : noté sur 15 (problème assez simple). Détails complets dans le pdf du cours 1, page 76.
- 5. "Routage / Trafic" : noté sur 17. Si interface graphique, noté sur 20. les ressources pdf (articles) sur Moodle
- 6. "Coloration de graphes" par colonie de fourmis : si rendu avec un 2e sujet choisi ci-dessus : bonus de 5 points. Si rendu seul (c-à-d. seulement ce problème codé) : noté sur 10,
 - ☞ Un code (aide à clarifier) pour ce problème est déposé sur Moodle. Ce code N'EST PAS UNE SOLUTION "à la colonie de fourmis" de ce problème mais une solution pour la coloration PSO (particle swarm Optimization) qui n'est pas tout à fait la même chose que par l'algorithme ACO (Ant Colony Optimization). Le code fourni est un exemple de PSO. Ne me le rendez pas comme travail !
- 7. Sujet Clustering / MST avec colonie de fourmis : mm genre que "Coloration de graphes"
- 8. ... (les autres sujets)

- 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.20 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/~kuba/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

I.1	Intelligence Emergente	1
I.2	Colonies de fourmis	6
I.2.1	Rappel des Principes	7
I.2.2	Exemple	8
I.2.3	Phéromone : dépôt et évaporation	10
I.3	Stigmergie, Intelligence émergente	11
I.3.1	Optimisation par essaim de particules (PSO) : le principe	14
I.3.2	Quelques exemples d'application	16
I.4	Modélisation de la colonie de fourmis comme un système multi-agents	18
I.4.1	Environnement : un graphe de villes	19
I.4.2	Les agents en IA	20
I.4.3	Le comportement d'un agent	22
I.5	La mise en oeuvre	24
I.5.1	La classe Route	28
I.5.2	La classe Ant (agent, fourmi)	29
I.5.2.1	Dépôt de phéromone par un agent	30
I.5.3	La classe Civilisation (Environnement)	33
I.6	Les règles à préférer / utiliser !	35
I.6.1	Mise à jour de la phéromone	37
I.6.1.1	La règle d'évaporation	38

I.6.1.2	La règle d'augmentation	39
I.6.2	Initialisation des paramètres	41
I.6.3	Remarques, critiques et propositions	42
I.7	Optimisation de la colonie de fourmis	43
I.8	Algorithme GI : le principe	45
I.8.1	Sélection	47
I.8.2	Progéniture (croisement)	48
I.8.3	Mutation	50
I.8.4	Migration	50
I.9	Mise en oeuvre des opérateurs génétiques dans PCC	51
I.9.1	Sélection	51
I.9.2	Progéniture, Croisement	52
I.9.3	Mutation	53
I.9.4	Migration	53
I.9.5	La nouvelle classe Ant	54
I.10	Remarques	55
I.11	Démo PCC	58
I.12	Sujet 2 : le problème de Sac à dos	59
I.13	Sujet 3 : TSP	74
I.13.1	Algorithme	81
I.13.2	La complexité de l'algorithme	85
I.13.3	Deux solutions alternatives au TSP	86
I.14	Sujet 4 : Bus Allocation Problem (BAP)	92
I.14.1	Génération de solutions	93
I.14.2	Evaluation des solutions	95
I.15	Une solution algorithmique au problème BAP	102
I.15.1	Description	103
I.15.2	Définition (pour une modélisation)	105

I.15.2.1 Calcul de ATT (et exemple) 107

I.16 Sujet 5 : Ramassage Bus Scolaire 110

 I.16.1 Recherche locale avec voisinage variable 112

I.17 Sujet 5 : Ramassage Bus Scolaire 113

I.18 Bonus 1 : Coloration de graphes 117

 I.18.1 Mise en oeuvre 1 118

 I.18.2 Exemple 120

 I.18.3 Mise en oeuvre 2 122

I.19 A rendre : modalités 124

I.20 Quelques References 126