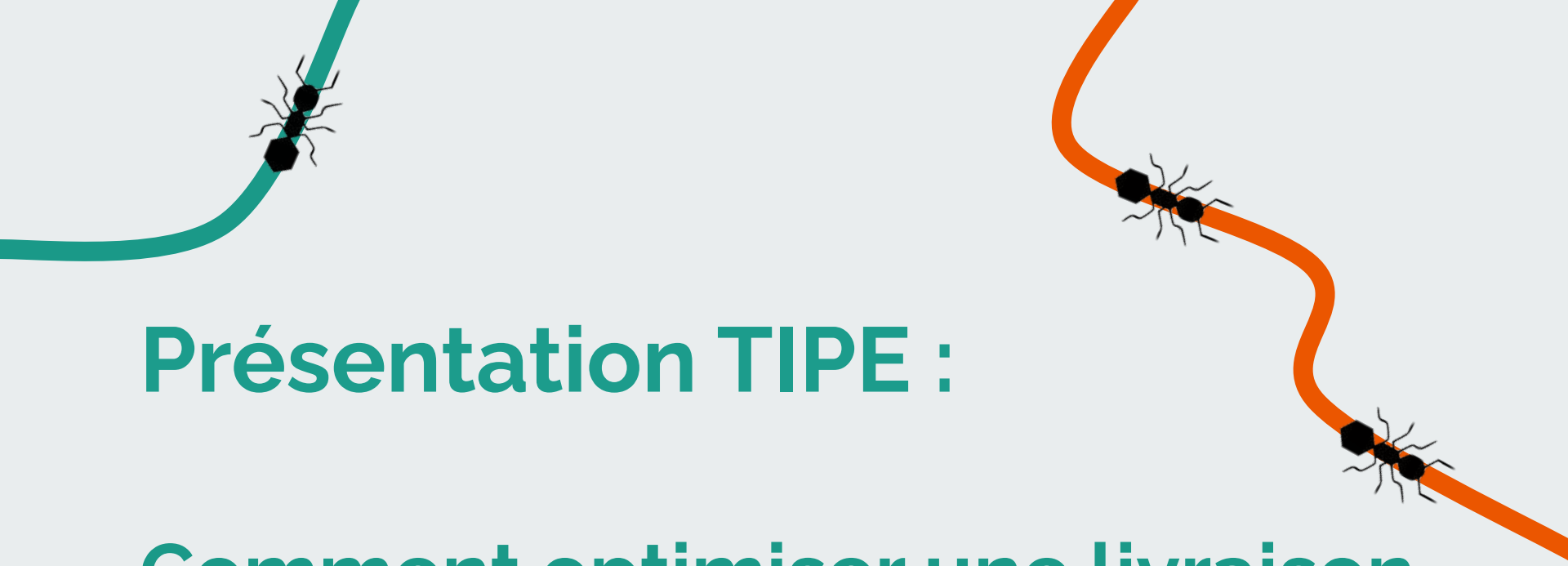


Présentation TIPE :

Comment optimiser une livraison de colis en s'inspirant du comportement des fourmis ?

Eliot Clerc

Candidat n°19655

Ancrage au thème de l'année :

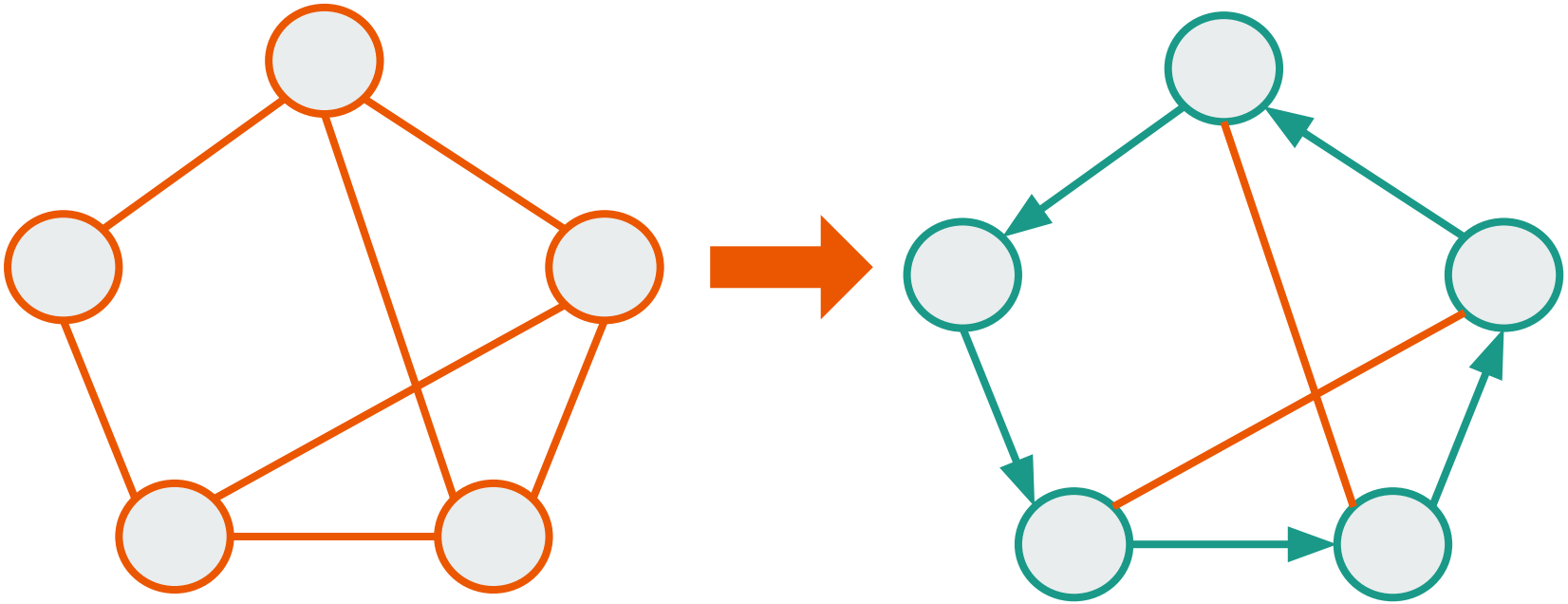
La ville

- Paris : entre **200.000** et **500.000** colis livrés par jour
- Empreinte carbone non-négligeable
- Impératif que les tournées de livraison de colis soient **les plus courtes possible**
  - Raisons **économiques**
  - Raisons **écologiques**

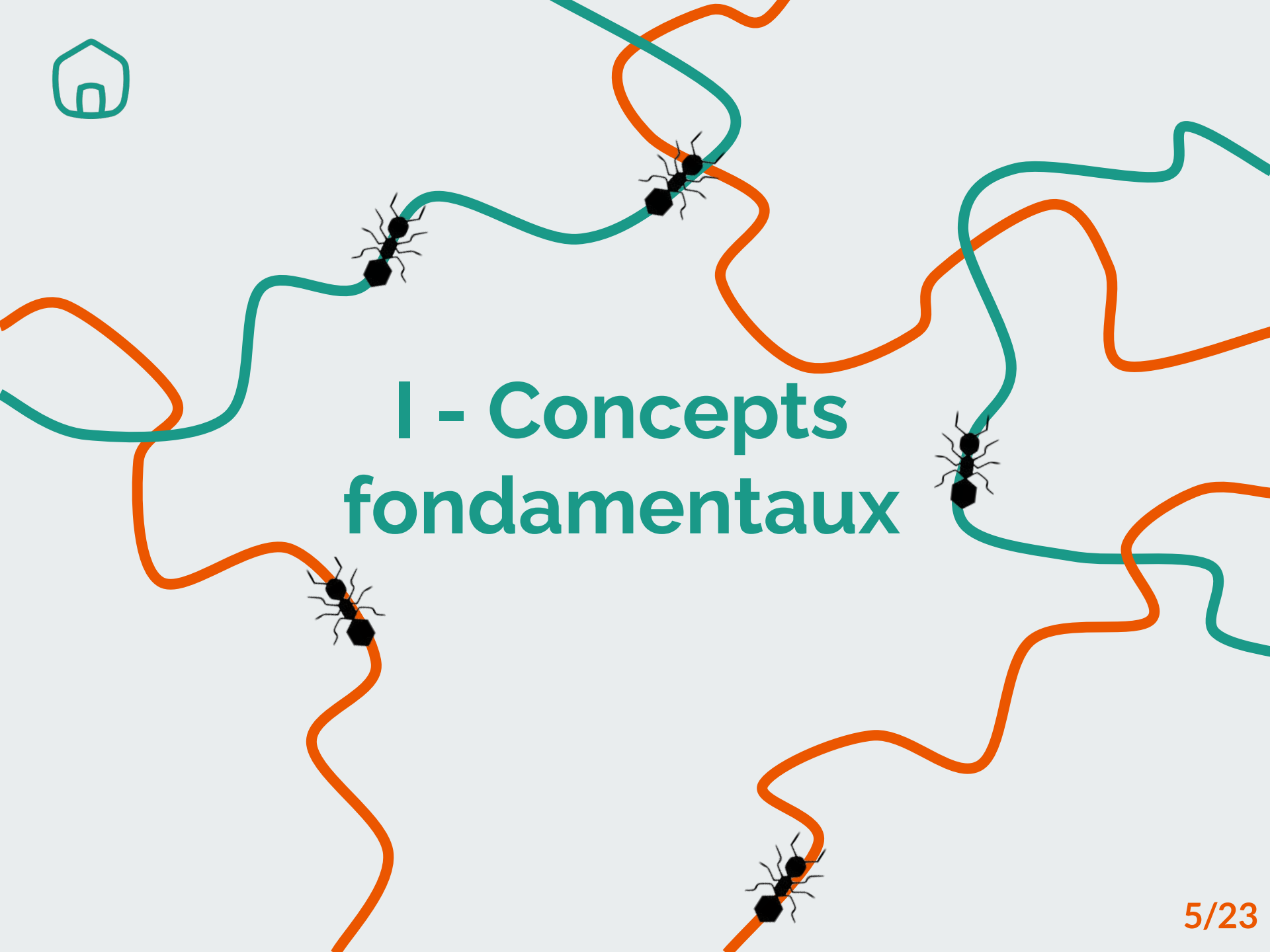
Problématique : Comment optimiser une livraison de colis en s'inspirant du comportement des fourmis ?

Objectifs

Problème du voyageur de commerce (TSP) :



Sommaire



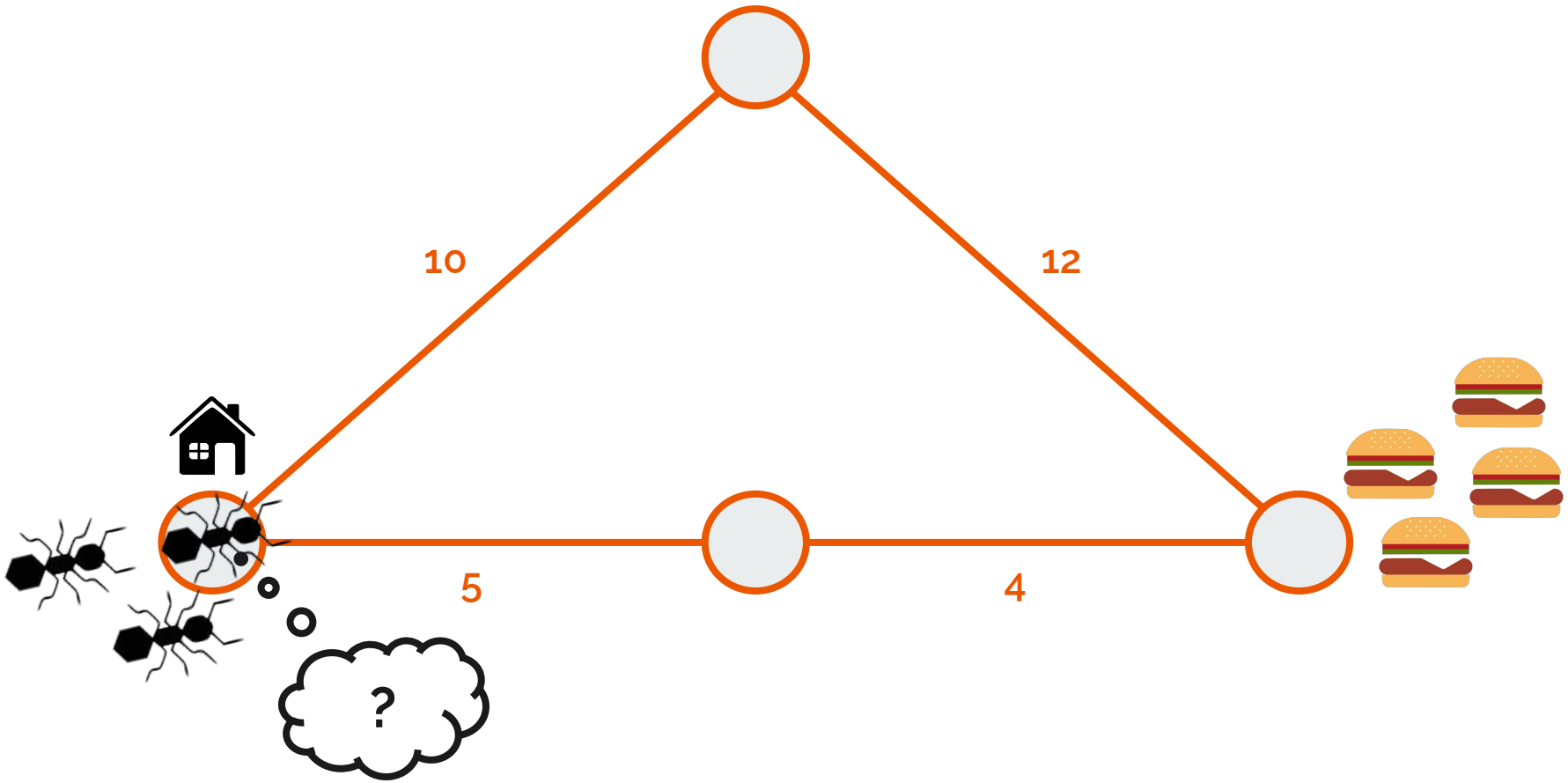
I - Concepts fondamentaux

Algorithmes d'approximation

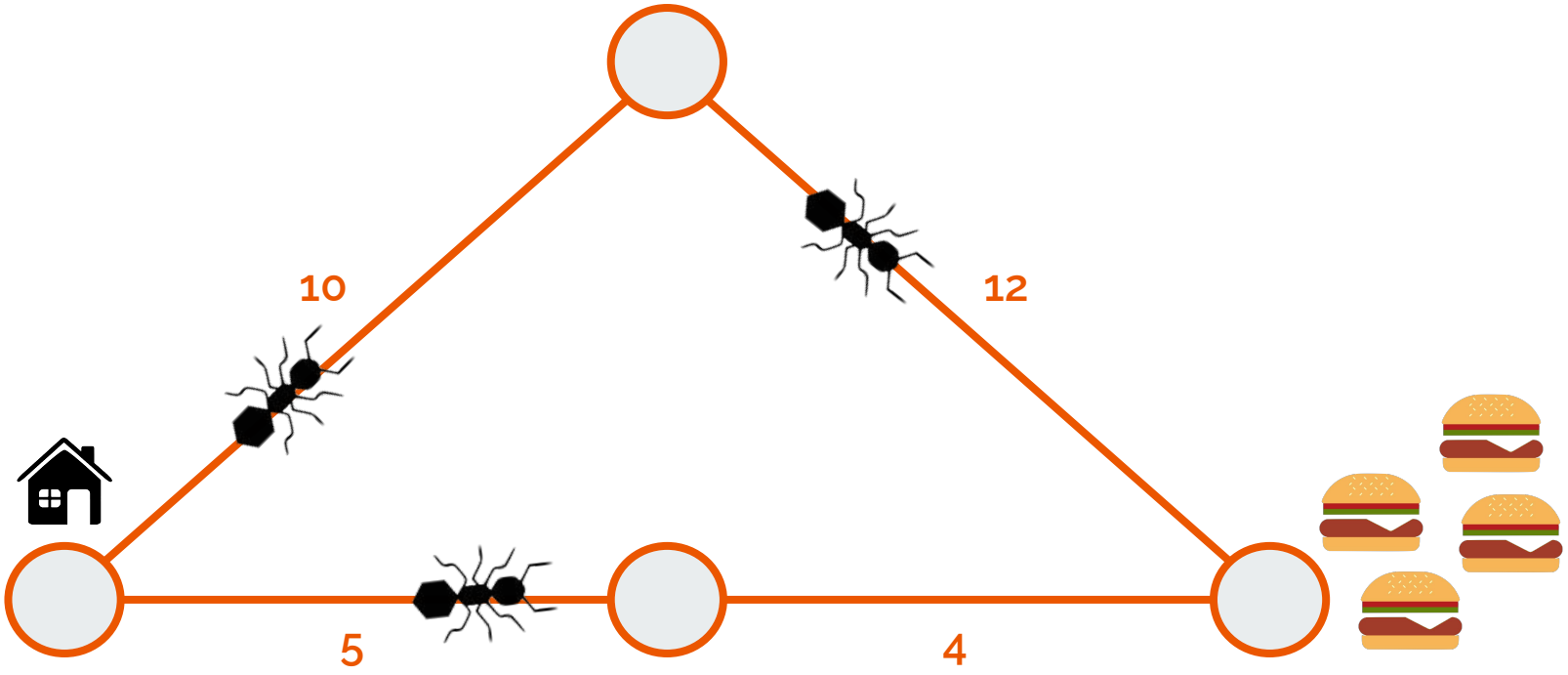
- Le problème du voyageur de commerce est un **problème NP-Difficile** donc il n'existe pas d'algorithmes de résolution en temps polynomial ⇒ On doit donc se tourner vers des **algorithmes d'approximation**

- On choisit donc une approche **biomimétique** en s'inspirant du comportement des fourmis

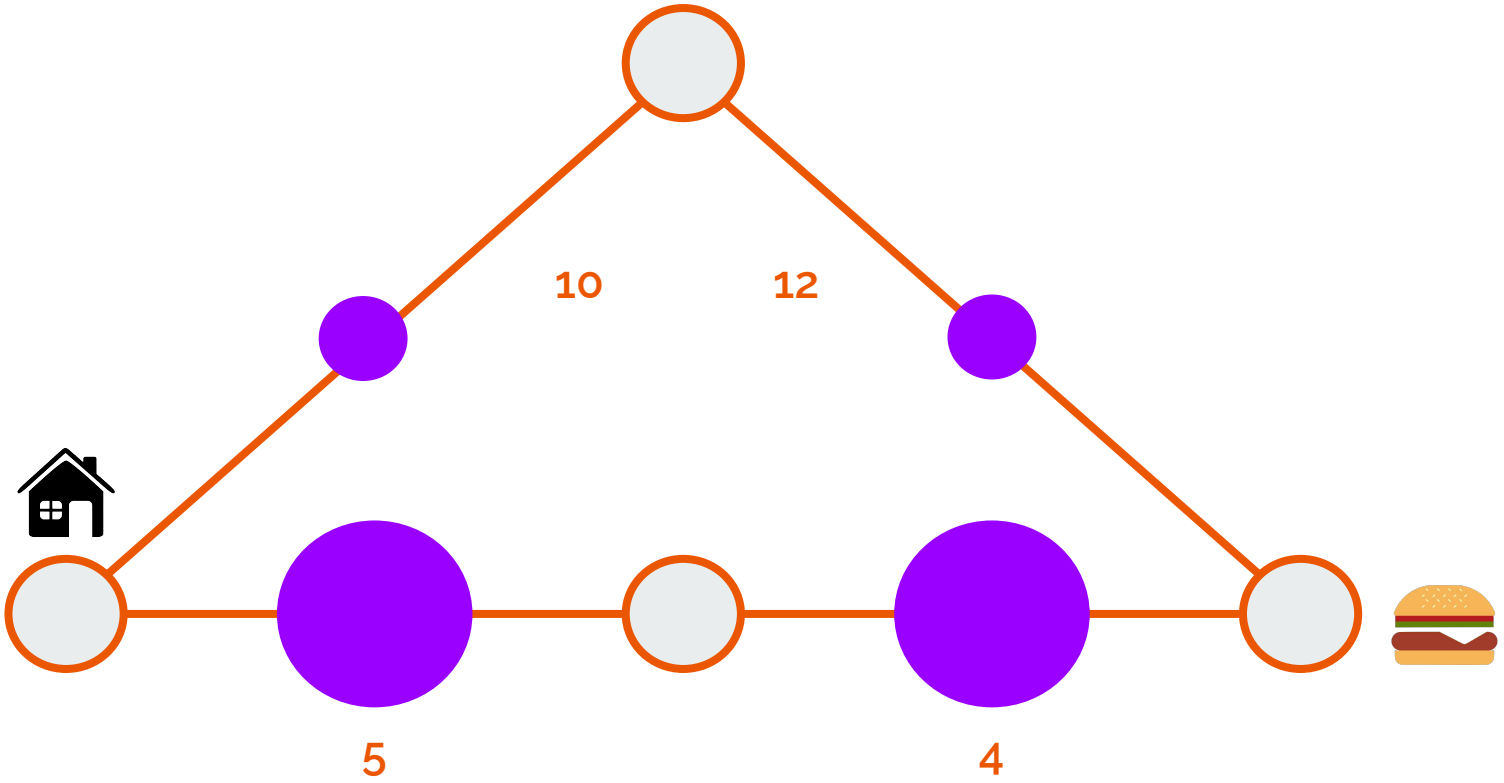
Algorithme des fourmis



Algorithme des fourmis

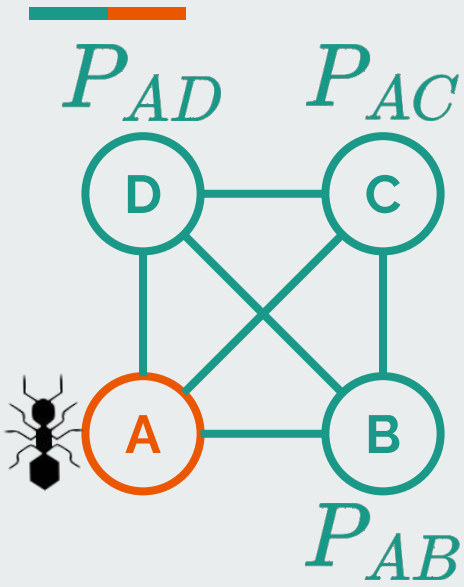


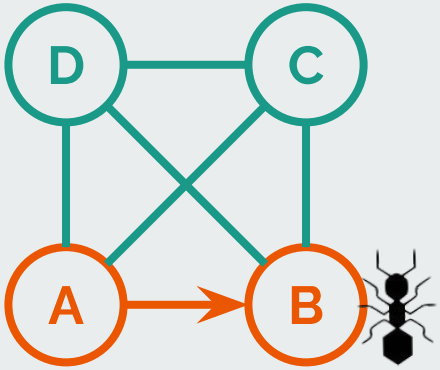
Algorithme des fourmis

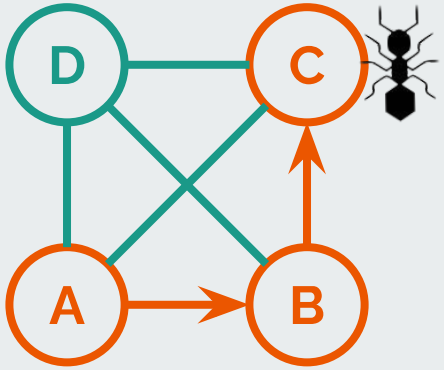


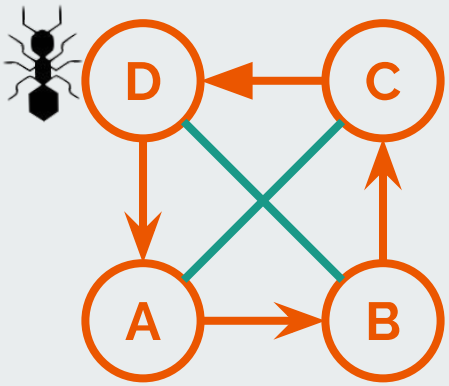
II- Construction d'une ACO

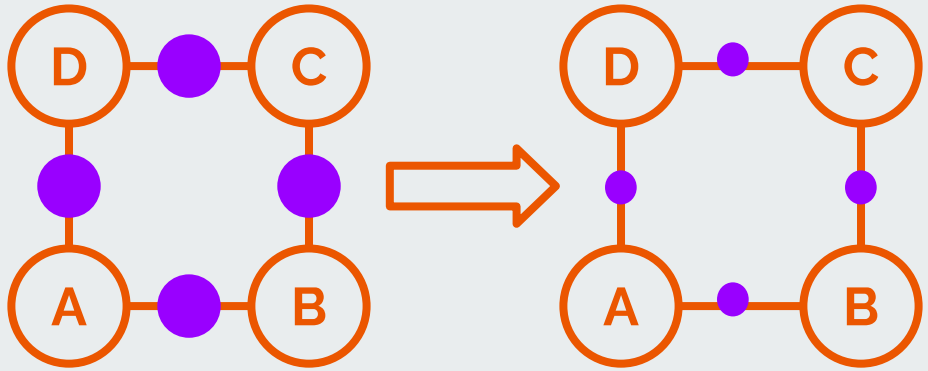
Adaptation de l'algorithme des fourmis au TSP :











Définir une Aco

$$\eta_{ij} = \frac{1}{distance_{ij}}$$

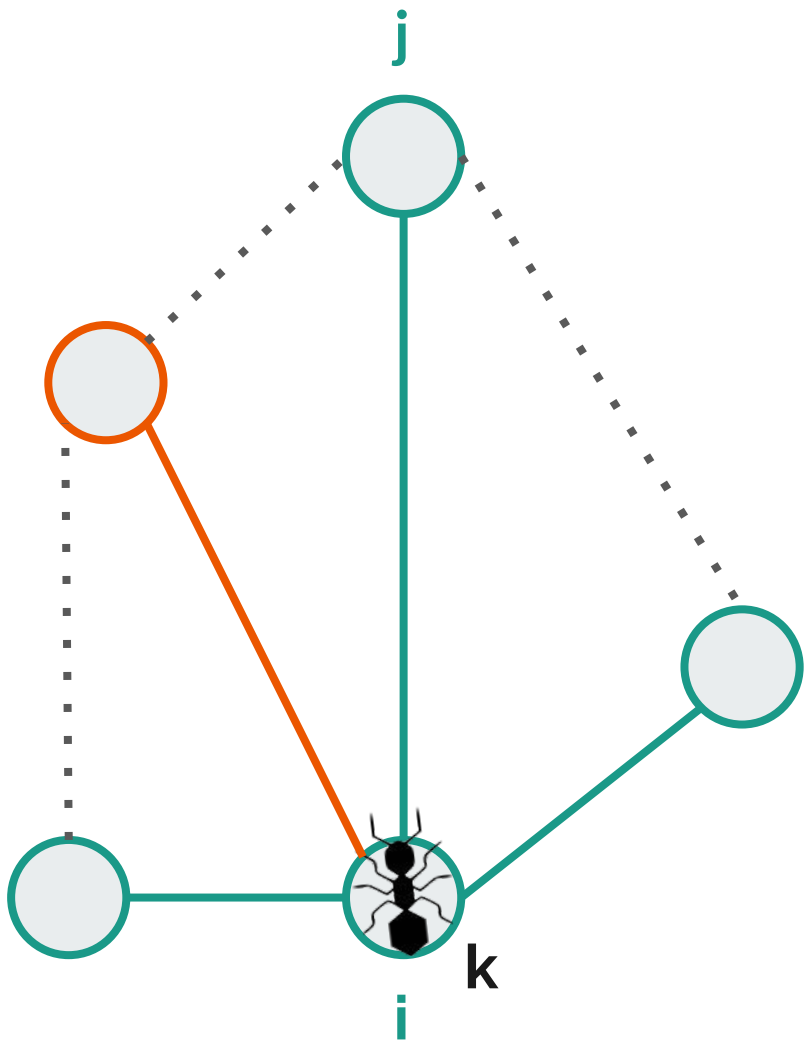
$$\tau_{ij} = pheromone_{ij}$$

$$\tau_{ij} = (1 - \rho) * \tau_{ij}$$

$$p_{ij}^{k} = \frac{(\tau_{ij})^{\alpha}(\eta_{ij})^{\beta}}{\sum_{l_k \in \text{non visité}} (\tau_{il_k})^{\alpha}(\eta_{il_k})^{\beta}}$$

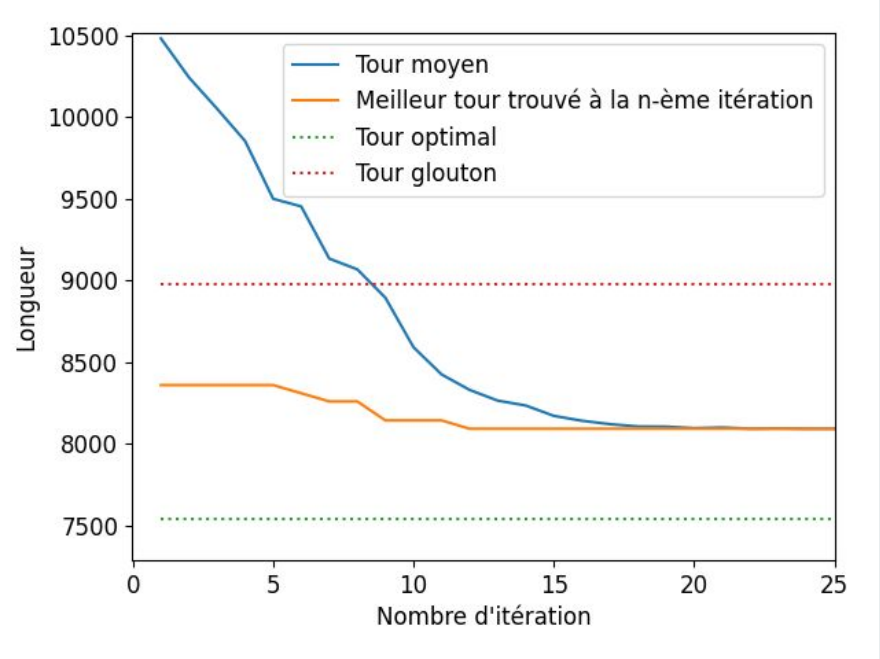
Sommet non visité par la fourmi k

Sommet déjà visité par la fourmi k

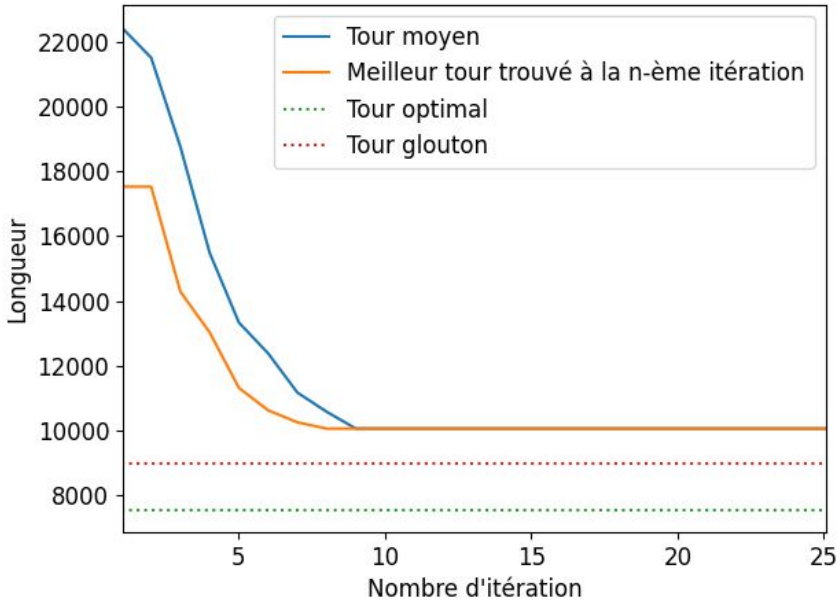


Influence de α et β sur les résultats

Instance : Berlin52 (52 sommets) de TSPLIB



Bon résultat obtenu pour α = 1 et β = 5



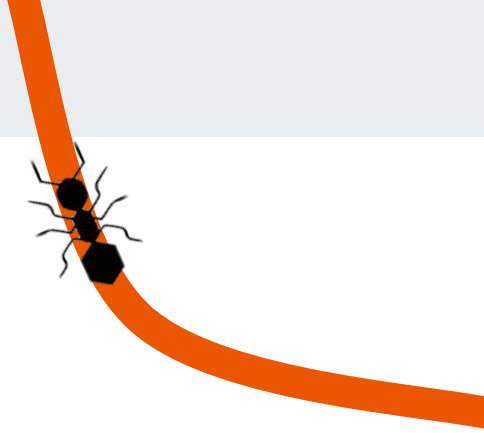
Mauvais résultat obtenu pour α = 4 et β = 1

Choix des paramètres

- ρ, nombre de fourmis, quantité de phéromone déposée et quantité de phéromone initiale
  - Thèse de référence de M. Dorigo

- Choix de α et β :
  - Contact professionnel M. Théraulaz
  - $\alpha = 1$ et $\beta \in \{2,3,4,5\}$

Pseudo-code de l'Aco



procédure Aco :

Initialisation des paramètres

pour chaque itération :

Calcul des cycles trouvés par chacune des fourmis
Trier les tours trouvés selon leur longueur

si on trouve un **nouveau meilleur cycle** :
mise à jour du minimum

Dépôt des phéromones sur les N tours les plus courts
Évaporation des phéromones

fin pour

III- Résultats

Résultats AcoV1 (100 tests de 200 itérations, instances TSPLIB)

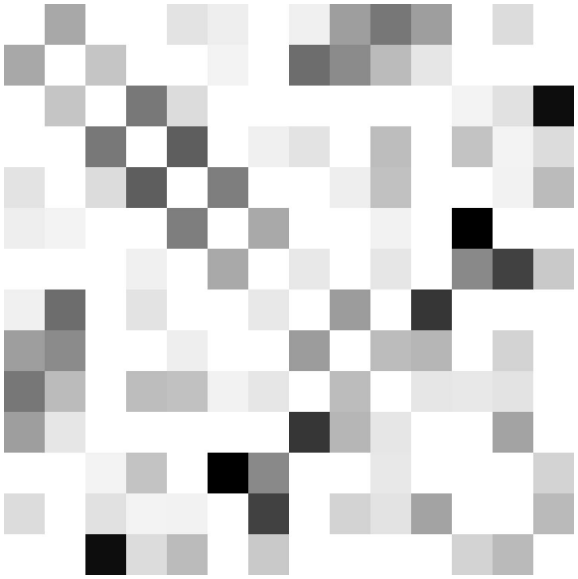
Instances	Taille	**Tour optimal**	Tour glouton	**Plus court tour trouvé**	**Facteur d'approximation** (en moyenne)	Moyenne des tours trouvés
lin105	105	**14379**	20356	**15905**	**1.18**	16929.5
kroA100	100	**21282**	27807	**24428**	**1.22**	25841.9
rat99	100	**1211**	1539	**1429**	**1.23**	1487.3
eil76	76	**538**	622	**587**	**1.18**	634.0
berlin52	52	**7542**	8980	**8155**	**1.14**	8597.6

Causes des “Mauvais” résultats

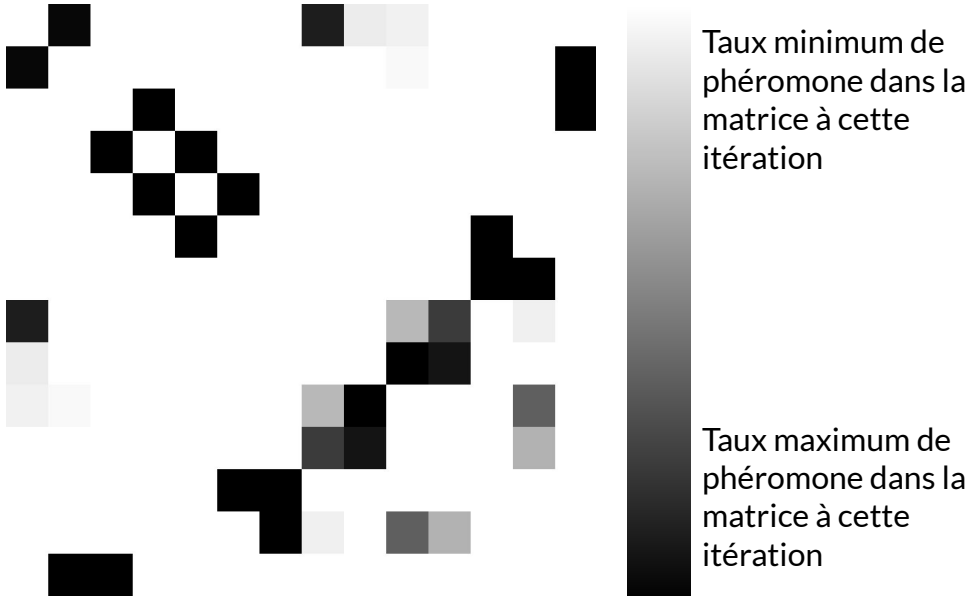
Exploration excessive

- Construction des cycles trop **aléatoire**

→ Due à : mauvais choix de α et β ou quantité initiale de phéromone trop grande



Matrice de phéromone dans le cas d’une exploration excessive (Burma14 itération 143)

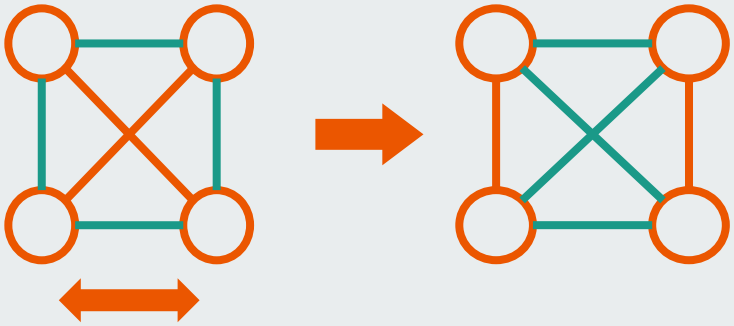


Matrice de phéromone sans ce problème (Burma14 itération 143)

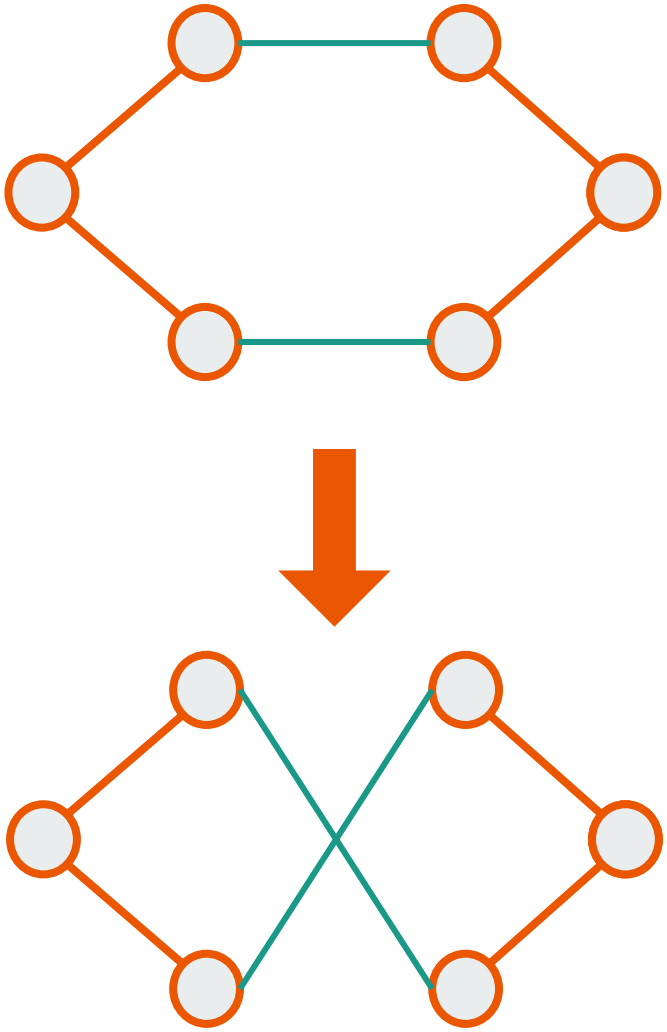
Recherche locale

But : tester de petites modifications sur les cycles trouvés au cas où elles permettraient d'en trouver un meilleur

Idée :
tester toutes les inversions de sommets 2 à 2



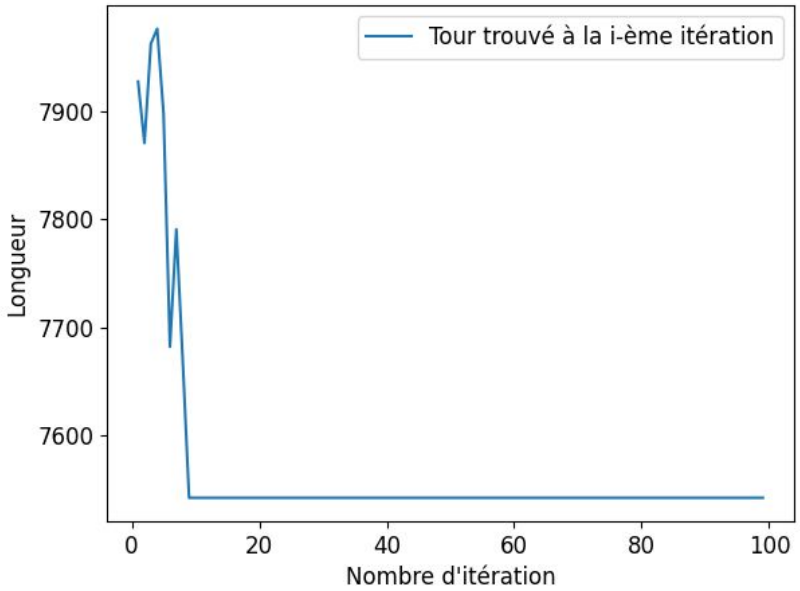
2-opt



Autres causes des “Mauvais” résultats

Convergence précoce

- Au bout de très **peu d’itérations**, et donc après peu de chemins testés, la colonie de fourmis converge ⇒ **mauvais résultats**

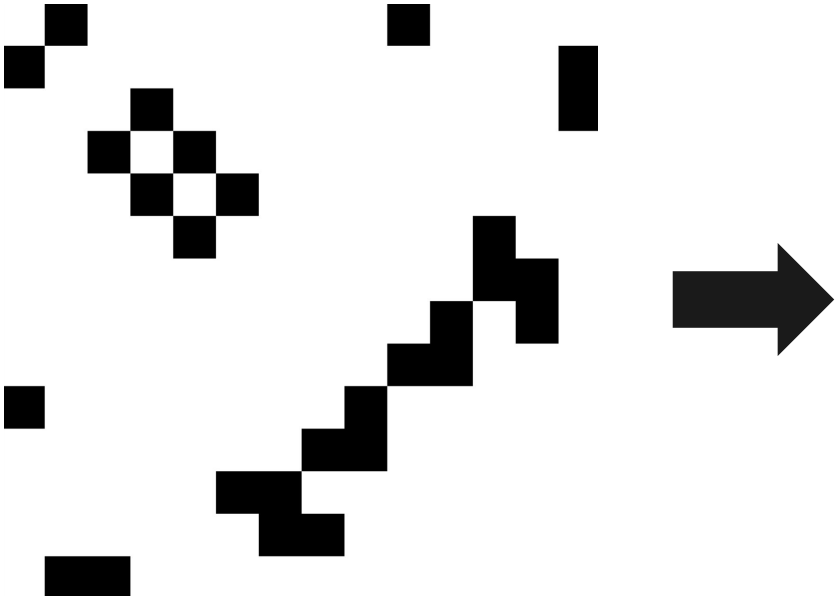


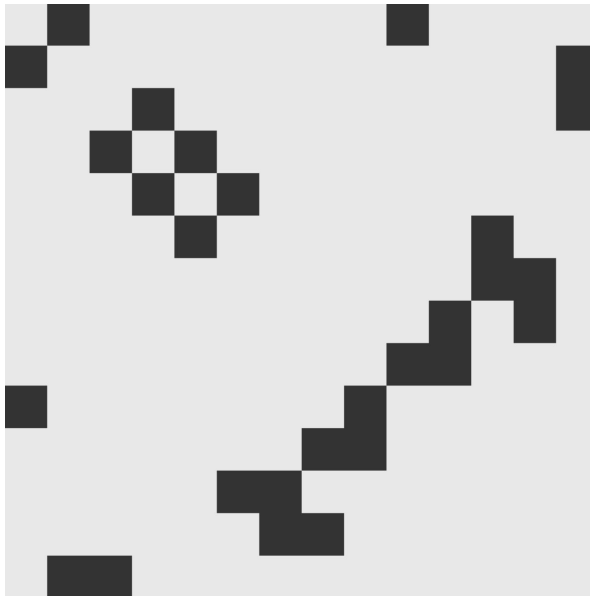
- Le phénomène “inverse” de l’exploration excessive

  ⇒ Trouver un équilibre entre **exploration** et **exploitation** de pistes prometteuses

Constante d'exploration

- L'idée est de **minorer le taux de phéromone** sur une arête
- Permet d'éviter la convergence précoce → Intérêt d'avoir plus d'itérations, car cela permet à la colonie de toujours "tester" de nouveaux chemins





Taux minimum de phéromone dans la matrice à cette itération

Taux maximum de phéromone dans la matrice à cette itération

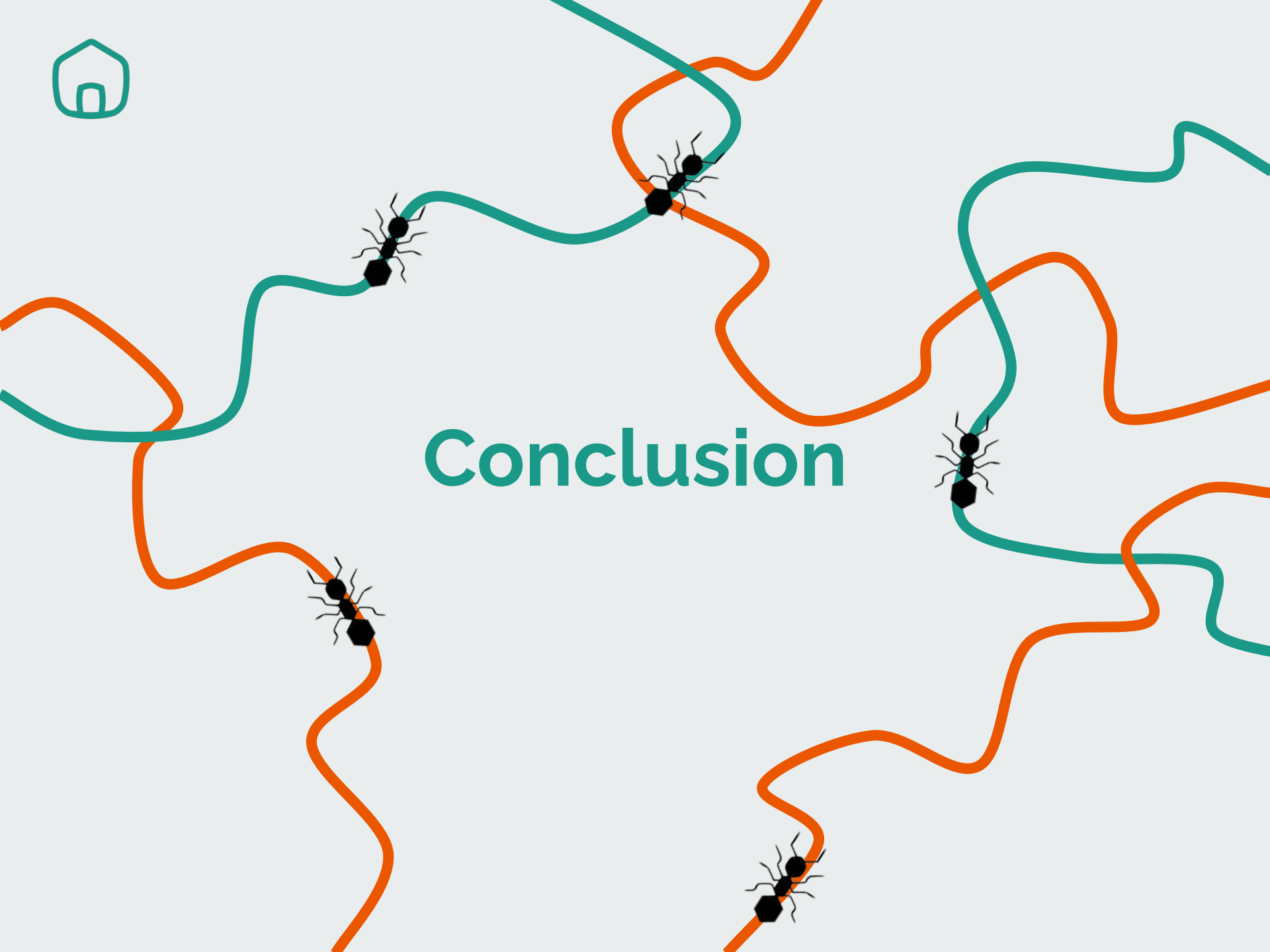
Constante d'exploration

- Celle-ci ne **dépend pas de l'instance** (longueur des chemins considérés, taux de phéromone initiaux)
  - Trop **faible** ⇒ s'avère **inutile**
  - Trop **élevée** ⇒ la recherche est trop **aléatoire**
- Solution apportée (personnel) :
  - Rendre la constante d'exploration dépendante du **taux maximum de phéromones**
  - Mais aussi du nombre de tour durant lequel, l'Aco n'a pas trouvé de nouveau plus court cycle ⇒ exploration progressive

$$C_{exploration} = 0.001 * N * \tau_{max}$$

Résultats AcoV2 (100 tests de 200 itérations)

Instances	Taille	Tour optimal	Tour glouton	Plus court tour trouvé	Moyenne des tours trouvés	Nombre de tours optimaux trouvés
lin105	105	**14379**	20356	14379	14379	**100**
kroA100	100	**21282**	27807	21282	21282	**100**
rat99	100	**1211**	1539	1211	1211	**100**
eil76	76	**538**	622	538	538	**100**
berlin52	52	**7542**	8980	7542	7542.0	**100**


Conclusion

Annexes

- Code de l'Aco
- Pseudo-code du calcul des cycles
- Tests d'autres traitements sur les phéromones
  - Réinitialisation de la matrice de phéromones
  - Constante d'exploration
- AcoV1.5
- Étude rapide de la complexité de l'Aco


Code de l'Aco

```
397 tour* ACO(int** g, int nbIterations, int nbNode){
398     int Cg = greedyTourLen(g,nbNode);
399     tour* bestTour = initEmptyTour(nbNode);
400     long double tau0 = 1 / (long double) Cg;
401     float rho = 0.1;
402     int nbAnt = nbNode;
403     int alpha = 1;
404     int nbKeep = 6;
405 
406     for(int beta = 2; beta < 6; beta++){
407         long double** pheromone = initPheromone(nbNode,tau0);
408 
409         for(int ite = 0; ite < nbIterations; ite++){
410             tour** tours = traverseGraph(g, nbNode, pheromone,nbAnt,alpha,beta);
411             qsort(tours,nbNode,sizeof(tour*), cmpTour);
412             tour** tmp = keepNBests(tours,nbAnt,nbNode,nbKeep);
413             tours = tmp;
414 
415             for(int i = 0; i < nbKeep; i++){
416                 tour* cycle = tours[i];
417                 if(cycle->length < bestTour->length){
418                     updateMax(bestTour,cycle,nbNode);
419                 }
420             }
421             updatePheromone(tours,pheromone,nbKeep,nbNode,rho);
422         }
423         freePheromone(pheromone,nbNode);
424     }
425     return bestTour;
426 }
```

Annexes

```
381  void updatePheromone(tour** tours,long double** pheromone ,int nbAnt,int nbNode,float rho){
382      for(int i = 0; i < nbAnt; i++){
383          tour* cycle = tours[i];
384          long double delta = (nbAnt - i) / cycle->length; //asRank
385
386          for(int j = 0; j < nbNode; j++){
387              pheromone[(cycle->data)[j]][(cycle->data)[(j + 1)%nbNode]] += delta; //dépot des phéromone
388          }
389      }
390      for(int k = 0; k < nbNode; k++){
391              for(int l = 0; l < nbNode; l++){
392                  pheromone[k][l] = (1 - rho)*pheromone[k][l]; //évaporation
393              }
394          }
395  }
```

```
283 tour** traverseGraph(int** g, int nbNode, long double** pheromone, int nbAnt,int alpha, int beta){
284     int nextNode = -1;
285     long double p = 0;
286     tour** tours = malloc(sizeof(tour*) * nbAnt);
287     bool* visited = malloc(sizeof(bool) * nbNode);
288     int* available = malloc(sizeof(int) * nbNode);
289     long double* probability = malloc(sizeof(double long) * nbNode);
290     initLongDoubleArrVal(probability,nbNode,0);
291     initIntArray(available,nbNode,-1);
292 
293     for(int i = 0; i < nbAnt; i++){ //pour chaque fourmi
294         int length = 0;
295         initBoolArray(visited,nbNode,false);
296         int sourceNode = rand() % nbNode;
297         int currentNode = sourceNode;   //Noeud de départ du cycle
298         int nbVisited = 0;
299         tour* cycle = initEmptyTour(nbNode);
300         (cycle->data)[nbVisited] = sourceNode;
301         visited[sourceNode] = true;
302         nbVisited++;
303 
304         for(int step = 0; step < nbNode - 1; step++){ //construction du cycle
305             int nbAvailable = 0;
306 
307             for(int node = 0; node < nbNode; node++){   //trouve les noeud pas encore visité par cette fourmi
308                 if(!visited[node] && currentNode!=node){
309                     available[nbAvailable] = node;
310                     nbAvailable++;
311                 }
312             }
313 
314             for(int i = 0; i < nbAvailable; i++){   //calcul des probabilités
315                 if(!visited[available[i]] && currentNode!=available[i]){
316                     p = proba(g,nbAvailable,available,currentNode,available[i],pheromone,alpha,beta);
317                     probability[i] = p;
318                     }
319             }
320 
321             nextNode = randomChoice(available,probability,nbAvailable); //choix du prochain noeud
322             visited[nextNode] = true;
323             length += g[currentNode][nextNode];
324             currentNode = nextNode;
325             (cycle->data)[nbVisited] = currentNode;
326             nbVisited++;
327 
328         }
329 
330         length += g[currentNode][cycle->data[0]];
331         cycle->length = length;
332         tours[i] = cycle;
333      }
334     free(probability);
335     free(visited);   //libère la mémoire aloué
336     free(available);
337     return tours;
338 }
```

Annexes

```
276  void localSearch(int** g, tour* cycle, int n, int nbIte) {
277      //implémentation de 2-opt
278      int improvement = 1;
279      int iter = 0;
280
281      while (improvement && iter < nbIte){ //on stoppe la recherche si on ne trouve pas d'amélioration
282          improvement = 0;
283
284          for (int i = 0; i < n - 1; i++) {
285              for (int k = i + 1; k < n; k++) {
286
287                  int d1 = g[cycle->data[i]][cycle->data[i + 1]];
288                  int d2 = g[cycle->data[k]][cycle->data[(k + 1) % n]];
289                  int d3 = g[cycle->data[i]][cycle->data[k]];
290                  int d4 = g[cycle->data[i + 1]][cycle->data[(k + 1) % n]];
291                  int delta = (d1 + d2) - (d3 + d4);
292
293                  if (delta > 0) { //si on obtient un meilleur cycle avec ce branchement d'arête
294                      reverse(cycle->data, i + 1, k); //on applique le branchement
295                      cycle->length = cycle->length - delta;
296                      improvement = 1;
297                  }
298              }
299          }
300          iter++;
301      }
302  }
```

```
292  void localSearch2(int** g, tour* cycle, int n, int nbIte){
293      double tmp = 0;
294      bool improved = true;
295
296      for(int ite = 0; ite < nbIte; ite++){ //on itère le processus un certain nombre de fois
297          if(!improved){
298              break;  //si on ne trouve pas de meilleur tour : on s'arrête
299          }
300
301          for(int i = 0; i < n; i++){
302              for(int j = 0; j < n; j++){
303                  swap(cycle->data,i,j);  //inversion des sommets
304                  tmp = tourLength(g, n, cycle->data);
305
306                  if(tmp < cycle->length){
307                      cycle->length = tmp;
308                  }
309                  else{
310                      improved = false;
311                      swap(cycle->data,i,j); //si on ne trouve pas mieux on remet comme avant
312                  }
313              }
314          }
315      }
316  }
```

Pseudo-code de l'Aco

```
procédure Calcul des tours :

    pour chaque fourmi :
        Choix du noeud de départ aléatoirement

        pour chaque étape :
            Calcul des probabilités de se rendre aux noeuds non visités
            Choix aléatoire du prochain noeud
            Ajout de ce noeud au cycle
            Mise à jour de la distance parcourue

        fin pour
    fin pour

    retourner les cycles des fourmis
```

Traitement sur les phéromones :

Réinitialisation de la matrice de phéromones

Principe :

- **Réinitialiser la matrice de phéromones** si la colonie a convergé (prématurément ou non) pour lui permettre de trouver d'autres potentiels meilleurs cycles

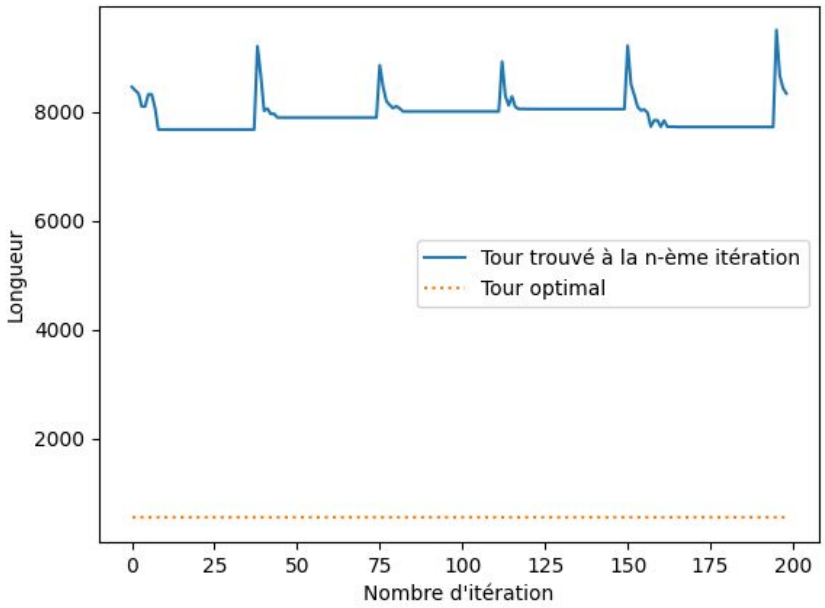
Problèmes :

- Il est assez dur de savoir quand le faire
  - Cela peut faire **perdre la recherche précédente** alors que la colonie de fourmis était proche de trouver un bon cycle
- Souvent, l'Aco cherche de nouveau vers les cycles vers lesquels elle était **bloquée**

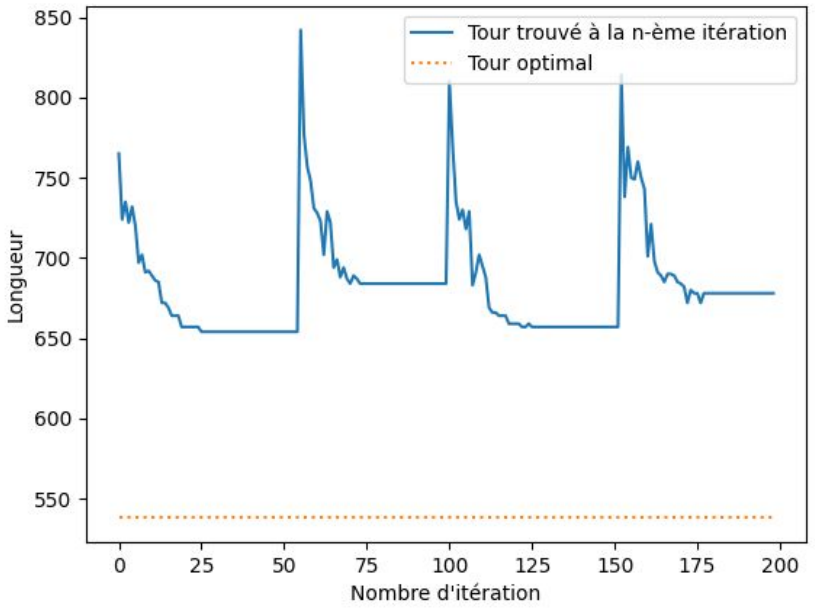
Traitement sur les phéromones :

Réinitialisation de la matrice de phéromones

➔ **Test : réinitialiser la matrice de phéromones si le meilleur tour trouvé n'a pas changé depuis plus de 30 itérations**



Instance : berlin52

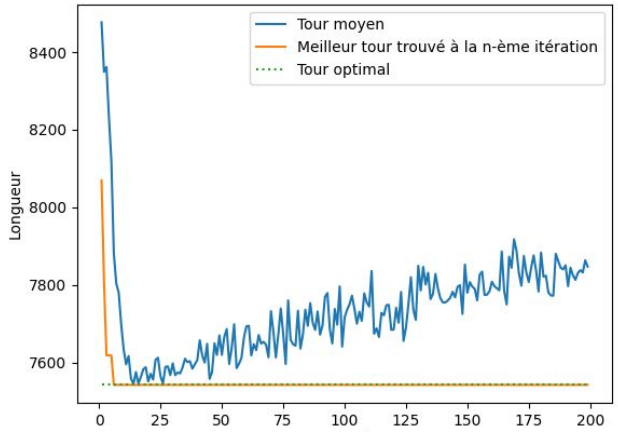


Instance : eil101

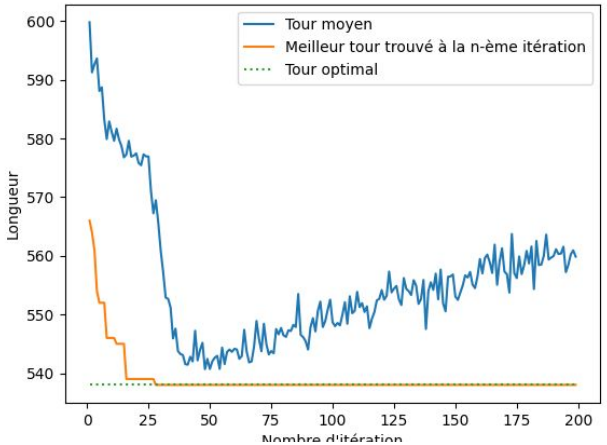
Traitement sur les phéromones :

Problème de la constante d'exploration si elle ne dépend pas du taux maximum de phéromone (uniquement progressive)

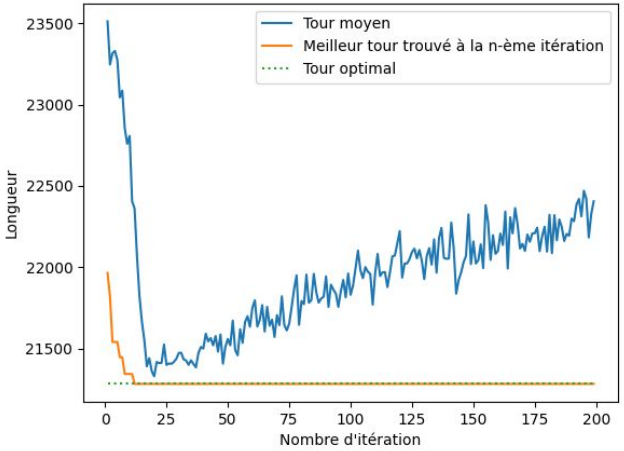
berlin52



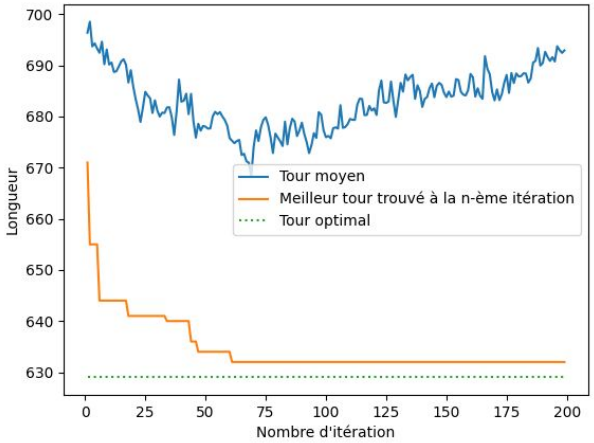
eil76



kroA100



eil101



Construction d'une AcoV1.5

- Pour éviter l'**exploration excessive** → AsRank
  - Seule les n "meilleures" fourmis déposent des phéromones
  - Elles sont **classées** et le taux de phéromone déposé dépend de leur classement

$$\tau = \frac{(n - rank)}{distance}$$

- α, β → fixer α = 1 et balayer une plage de valeurs cohérentes de β ∈ {2,3,4,5}

Étude rapide de la complexité

- L'implémentation de l'Aco choisie à une **complexité temporelle** : $O(m * n^2)$
- **m** : nombre de fourmis, **n** nombre de ville
  - Avec les thèses de références (M. Dorigo) : $O(n^3)$
- Dans la pratique, la version de l'Aco implémentée permet de traiter des instances de l'ordre de la **centaine de sommets**