Comin Clément Jlassi Ghassen

IAD Rapport de projet

Ce projet porte sur un sujet de recherche de Marco Dorigo et Luca Maria Gambardella publié en 1997. Il présente plusieurs améliorations possibles à l'algorithme de colonie de fourmis utilisé pour résoudre notamment le problème du voyageur de commerce et compare les performance de cette implémentation par rapport à celles d'autres algorithmes utilisés.

Nous commencerons par présenter le système de colonie de fourmis, puis les modifications apportés par cette étude. Nous présenterons par la suite brièvement notre propre implémentation puis discuterons du choix des paramètres ainsi que des résultats obtenus.

Présentation du problème

On se place sur un graphe ponderé complet, orienté ou non. Un des problèmes classiques que l'on peut chercher à résoudre est celui du voyageur de commerce. Il s'agit de trouver un circuit dans le graphe qui visite tout les sommets et dont le cout soit minimale. Ce problème est NP-difficile, l'interet est ici d'améliorer la solution déjà apportée par un système d'agents distribués et de fournir une solution pratique à ce problème dont les applications sont nombreuses (transport, logistique, routage de packets IP).

L'algorithme de la colonie de fourmis s'inspire de la nature pour mettre en place, de manière itérative, un chemin qui tend à se rapprocher de la solution optimale.

Dans ce modèle les agents sont des fourmis qui marchent sur les arrêtes du graphe et la connaissance partagé prend la forme d'un coefficient propre à chaque arrête et que l'on appelle phéromone. La décision prise par chaque agent d'emprunter ou non cette arrête lors de son circuit dépend de la quantité de phéromone ainsi que du poids de l'arrête dans le graphe ou son temps de parcours. Le poids de l'arrête qui peut être vue ici comme une information heuristique mais non suffisante pour déterminer le meilleur chemin.

On favorise ainsi les chemins courts ayants déjà été déjà visité aux itérations précedantes, et qui sont donc susceptibles d'apartenir à la solution optimale.

A la fin de chaque itération les agents vont déposer une certaine quantité de pheromone. La quantité de phéromone deposée est inversement proportionelle à la longueure du chemin, de sorte que les arrêtes d'un chemin court auront un probabilité plus grande d'être choisit lors de l'itération suivante.

On ajoute à cela une diminution constante de la quantité de phéromone pour toute les arrêtes, celles qui sont peu utilisés seront abandonnés car elles n'appartiennent pas au plus court chemin.

Au fil des itérations les fourmis vont explorer différents chemins dans le graphe en favorisant ceux qui semblent optimaux et après suffisament d'itérations certains chemins se démarqueront par la quantité de pheromone importante qui se trouve sur leurs arrêtes. Si le système converge tout les agents adopteront le même chemin, et si ce chemin correspond à la solution optimale du problème alors l'algorithme aura été concluant.

La convergence de l'algorithme vers une solution et le nombre d'itérations nécessaire pour y parvenir dépend de plusieurs variables, notamment le nombre de fourmis, la loi de probabilité utilisé lors du choix de l'arrête à empunter ou le taux de décroissance des phéromones.

Présentation de l'ACS

L'algorithme de la colonie de fourmis fut proposé pour la première fois en 1992, et l'ACS correspond à une version amelioré. Il modifie notamment la règle de transition qui correspond au choix de l'arrête à emprunter, mais également les variations des taux de phéromone sur les arrêtes.

Régle de transition

On définit une constante q0 dans [0:1] qui va permettre de choisir, lors de chaque transition entre deux sommets, d'opter pour une stratégie d'exploration du graphe ou de confirmation des chemins déjà connue. On simule pour cela q, une variable aléatoire suivant une loi uniforme dans [0:1].

Si $q>q0$: On opte pour une stratégie dite "biased exploration", qui correspond exactement à la règle de transition utilisé dans la version classique de la colonie de fourmis.

Supposons que la fourmis k se trouve sur le sommet s_1, soit $J_k(s_1)$ l'ensemble des sommets du graphe n'ayant pas encore été visité à cette étape, et atteignable à partir de s_1, alors pour tout $s_2 \in J_k(s_1)$ on pose:

$$p_k(s_1,s_2)=\frac{\tau(s_1,s_2)\times[\eta(s_1,s_2)]*\beta}{\sum_{u\in J_k(s_1)}\tau(s_1,u)\times[\eta(s_1,u)]*\beta}$$

Ainsi la probabilité que la fourmis choisisse d'emprunter l'arrête $s_1 - s_2$ et propotionnelle à la quantité $\tau(s_1,s_2)\times[\eta(s_1,s_2)]*\beta$ que l'on appelle désirabilité de l'arrête et que nous noterons $\delta(s_1,s_2)$ dans la suite de ce apport.

La variable τ correspond à la quantité de phéromone au moment du calcul et η est l'inverse du poids de l'arrête. Le coefficient β permet donc d'ajuster l'importance relative entre l'information heuristique (poids de l'arrête) et l'information partagé entre les agents.

Cette loi de probabilité guide le choix de l'agent vers les chemins pour lesquels δ est fort mais permet également, bien avec qu'avec une probabilité faible, de choisir une arrête peu attrayante. On peut donc assimiler cette régle de transition à une exploration du graphe, mais guidée par la connaissance partagé des agents .

Si $q\leq q0$: On opte pour une stratégie d'exploitation des résultat déjà obtenues en choisissant parmis les arrêtes menant à un sommet pas encore visité celle pour laquelle δ est maximal. On choisit dans ce cas de manière arbitraire de réutiliser un chemin déjà connue afin d'affiner notre choix sur celui-ci. Utiliser fréquement l'exploitation devrait permettre d'obtenir une convergence plus rapide.

Régle de modification des taux de phéromone

Les taux de phéromone du graphe doivent être modifiés à la fin de chaque itération afin de faire évoluer la connaissance distribué portant sur les chemins du graphe.

La modification comprend une décroissante systématique d'un taux α pour toute les arrêtes du graphe. Cela peut être assimilé à l'évaporation naturelle des phéromones et permet d'oublier progressivement les chemins qui ne sont plus explorés.

Alors que l'implémentation originelle prenait en compte les circuits réalisés par toute les fourmis, l'ACS ne conserve que celui présentant le meilleur temps de parcours.
On va incrémenter la quantité τ des arrêtes appartenants au meilleur tour par $\alpha \times \eta_{gb}$ où η_{gb} est l'inverse du poids total de ce circuit. La modification globale va ainsi mettre en avant la solution optimale trouvé à la fin de l'itération.

A cette modification globale l'ACS propose de rajouter une modification locale qui s'applique à chaque transition dans le graphe. Le chemin emprunté verra sa phéromone decrémentée d'un taux γ avant d'être augmenté de $\gamma \times \Delta\tau(s_{1,} s_2)$.

L'étude propose trois expressions pour Δτ :

$\gamma \times max(s_1, z)$ pour $z \in J_k(s_1)$, formule inspirée du Q-learning et qui est bien adapté dans un contexte d'apprentissage par renforcement.

Δτ constant fixé à τ_0 la quantité initiale de phéromone ou bien nulle, afin de désactiver l'effet de cette mise à jour locale.

Cette modification locale, bien que couteuse en temps de calcul, amène un aspect dynamique à l'apprentissage. Les arrêtes emprunté par une fourmis deviendront un peu moins désirables, poussant les autres fourmis à explorer des chemins différents. Ces différents chemins seront ensuite récompensés ou pénalisés lors de la modification globale en fin de tour.

Implémentation de l'algorithme

Notre implémentation utilise Netlogo afin de simuler et de visualiser le déroulement de l'algorithme. Le graphe utilisé est non-orienté, ponderé et complet afin de faciliter la recherche d'un circuit.

Les arrêtes possèdent deux attributs : *len* et *phe* qui correspondent respectivement au poids, initialisé de manière aléatoire, et à la quantité de phéromone, on ne s'interresse ici qu'au cas symétrique du TSP.

L'interface graphique permet de choisir divers coefficients du problèmes ainsi que des constantes utilisés dans les équations. On peut notamment choisir le nombre de sommets et le nombre d'agents.

Le paramètre $\tau 0$ est fixé au moment de l'initialisation : un premier calcul heuristique est mené en utilisant la méthode du plus proche voisin afin d'obtenir une estimation L du coût d'un chemin efficace. On fixe alors la valeure de $\tau 0$ à $(n*L)^{-1}$ comme le recommande l'étude.

Pour la mise à jour locale, nous avons choisi de fixer $\Delta\tau$ à $\tau 0$ afin de ne pas ralentir les calcul, mais l'expression utilisé pour Ant-Q est mis en commentaire dans la fonction *global_update*.

Le bouton *setup* genère un graphe de poids aléatoires et calcul un premier circuit de manière heurisitque avant le le colorer en rouge, un graphique permet également de suivre l'évolution des coûts des circuit à chaque itérations ainsi que le minimum globale.

Le bouton *go* permet de lancer la simulation : Les fourmis vont appliquer l'algorithme pour construire une nouvelle solution en la colorant en rouge. Le meilleur circuit est conservé dans la variable *best_circuit* et on arrête la simulation au bout de *cond* simulations. On considère alors que l'on a atteint la convergence de la méthode et on affiche le meilleur circuit trouvé et son coût.

Simulations

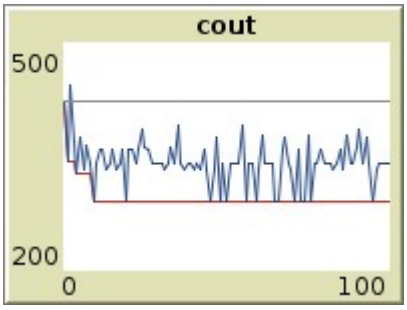
Afin de mesurer les perfomances du système que nous avons implementé nous utiliserons un graphe qui contient les évolutions des variables *iter_len* et *cost,* représentant respectivement le coût du meilleur chemin trouvé à la fin d'une itération et celui du meilleur chemin trouvé depuis le début de la simulation.

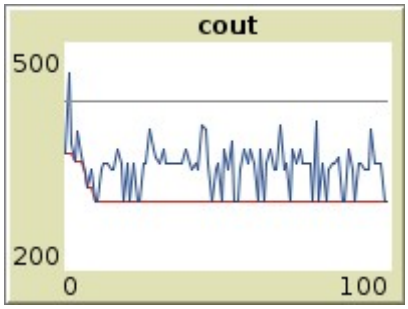
iter_len sera tracé en bleu et *cost* en rouge, on fera également apparaître la constante *heur* qui correspond à la solution trouvé par l'heuristique du plus proche voisin en gris.

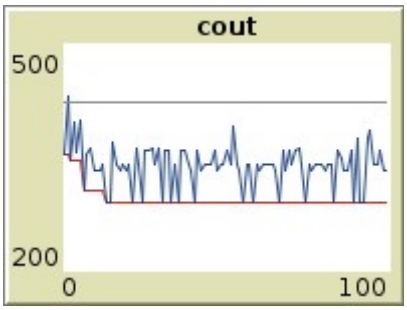
Pour toute les simulations suivantes, le nombre de sommets sera fixé à 40, le nombre de fourmis à 10 et les arrêtes prendront des poids aléatoires entre 1 et 100. La valeure de $\tau 0$ sera determiné par le résultat de l'heuristique et on menera la simulation sur 100 itérations.

Simulations en utilisant les paramètres recommandés :

on aura donc $\beta=0.9$, $\alpha=0.1$, $\rho=0.1$. Les simulations sont menés sur le même graphe.



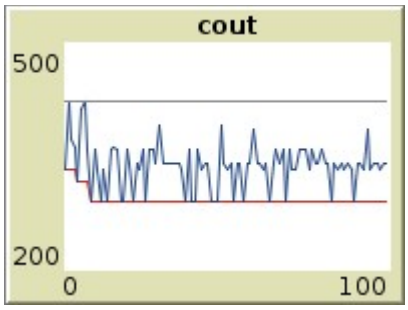


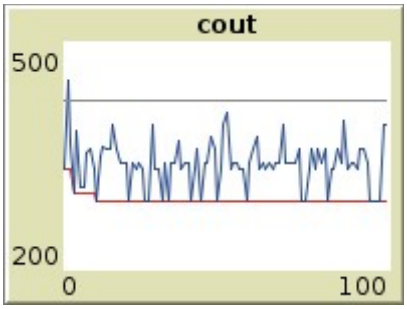


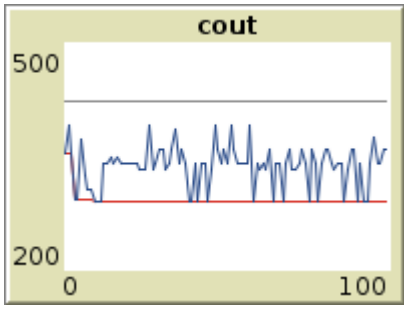
On retrouve dans les trois cas un comportement similaires : les agents trouvent assez rapidement la meilleur solution après quelques améliorations, les circuits empruntés sont cependant variables, ce qui permet de garantire qu'un nombre satisfaisant de chemins ont été testés.

Simulations de Ant-Q:

On utilise ici les mêmes paramètres que précedemment mais en appliquant la formule de Q-learning lors de la mise à jour locale de phéromone avec $\gamma=0.5$.







Ces simulations semblent montrer une convergence un peu plus rapide qu'avec l'algortithme ACS classique.

Conclusion

Ce projet nous a permis de découvrir une méthode particulière de résolution d'un problème NP-difficile très classique. Cet algorithme utilise une méthode pseudo-aléatoire faisant appel à des agents ne partageant que des informations très simples et qui pourtant parviennent collectivement à faire émerger une solution proche de l'optimal.

Netlogo se revèle également particulièrement adapté pour l'étude d'intelligence artificielles distribuées de part sa simplicité et son expressivité, mais également par son interface graphique qui permet de mettre en place très rapidement des simulations précises.

Cette méthode inspirée de la nature peut de plus être mise en place dans des conditions réelles de fait de sa relative simplicité et de son caractère dynamique qui lui permet de s'adapter au variations de paramètres.