Stage KOMOTO

Documents de recherches

par Sébastien Maire

Explication du contexte

Dans le cadre de la préparation de commandes on dispose d’un stock de marchandises. Ces marchandises sont rangées dans des allées et référencées par un système de codes-barres.

Il existe différentes méthodes pour la préparation de commandes, mais la méthode que nous traitons ici, se découpe en 3 étapes :

1. Le prélèvement des articles dans le stock

2. Le tri

3. Le colisage

1. Le prélèvement

Dans notre cas, pour préparer les commandes, le préparateur ne prélève pas seulement les pièces de sa commande. On constitue des lots de commandes (prélèvement par groupe).

En effet la constitution de lots de commandes diminue la distance à parcourir.

Prenons pour exemple un entrepôt dont le parcours total des allées représente 5km. Nous devons préparer 3 commandes A, B et C ayant comme distance respective à parcourir pour le picking 3km, 4km et 2km. Si on prélève ces commandes séparément, la distance totale parcourue sera de 9km, alors que si on constitue un lot de ces 3 commandes, on effectuera au maximum 5km (le parcours total du stock). Ainsi on économise 4km de marche.

1. Le tri

Une fois le prélèvement effectué, il est acheminé au poste de tri.

Le tri peut se faire avec ou sans aide informatique.

Ici on procède à une ‘ventilation’.

Quand le préparateur à prélevé le lot de commandes, un 2ème préparateur s’occupe de trier les commandes avec une assistance informatique. Il dispose d’une étagère où chaque compartiment correspond à une commande et est équipé d’une lumière. Le préparateur prend les articles un à un et les lit avec un scanner qui déclenche la lumière de la commande correspondante. Chaque case est également équipée d’un scanner afin de vérifier que l’article déposé est le bon, ce qui réduit le risque d’erreurs lors de la préparation de commandes.

1. Le colisage

Quand les commandes sont triées, elles sont envoyées au poste de colisage.

La façon de grouper les commandes influence la distance à parcourir pour leur prélèvement.

Reprenons l’exemple du stock dont le parcours total représente 5 km, nous avons cette fois 4 commandes A, B, C et D avec respectivement 3km, 4km, 2km, et 4km de distance à parcourir pour chacune.

Si nous constituons des lots de 2 commandes, nous pouvons créer 6 lots de commandes différents.

AB, CD

AC, BD

AD, BC

Oui, mais plutôt 3 possibilités de partitions en 2 lots.

Je n’ai pas eu le temps de me pencher sur le calcul, je verrais ça ce week-end.

Supposons que la commande A contient les même articles que la commande B avec quelques articles en moins. En groupant les 2, leur distance ne s’additionnera pas puisqu’on passera par le même chemin de picking.

* Sujet du stage

On utilise ici la méthode du ‘pick then pack’, on effectue d’abord le prélèvement des articles par groupe, on les emmène ensuite au poste de ventilation pour le tri et enfin on les emballe.

Pour accélérer la préparation de commande, je dois chercher un moyen de diminuer la distance parcourue pour le prélèvement des pièces dans le stock.

Pour cela, le but est de trouver comment constituer des lots de commandes pour que la distance totale de prélèvement soit minimale.

* Plan de travail

1. **Etape de recherche** : rechercher les différents algorithmes qui pourraient fournir une solution à la problématique.
2. **Prototypage, (phase de test)** : Si les recherches se sont avérées concluantes, fabrication d'un prototype afin de juger l'efficacité d'un algorithme.
3. **Mise en œuvre**: Si le prototypage confirme l'efficacité d'un algorithme, intégration dans le projet kheops en C#, puis portage sur CUDA.

Recherches

* Algorithmes exhaustifs

Pour les algorithmes exhaustifs, le nombre de combinaisons possibles augmente de façon exponentielle en fonction du nombre de données ; pour 50 commandes que nous séparons en 10 lots de 5 commandes, il y a 502395101553638076363011606091869094039864705870235479376 combinaisons possibles. Nous travaillons ici sur des échelles plus grandes, entre 500 et 2000 (voire beaucoup plus) commandes.

J’ai utilisé le même calcul ici que plus haut qui me donnait 15 possibilités. Le calcul est aussi erroné.

* Réseaux de neurones

Parmi les réseaux de neurones, se trouve une catégorie appelée SOM (Self Organizing Map)

Ces réseaux de neurones fonctionnent sur un apprentissage dit non-supervisé, ce qui permet au réseau de classer de manière homogène les informations (ici créer des lots de commandes).

Contrairement à l’apprentissage supervisé ou l’on ‘apprends’ par des exemples (comparaison entre le résultat donné et le résultat souhaité), l’apprentissage non-supervisé ne nécessite pas que l’on connaisse le résultat voulu.

Nous considérons les articles d’une commande comme les composantes d’un vecteur.

1. On affecte des valeurs aux neurones de sortie. Cette affectation peut être pseudo aléatoire et dépend du problème traité.
2. On prend en compte aléatoirement les commandes à grouper.
3. Chacune des valeurs de la commande prise en compte est envoyée dans le réseau de neurones.
4. Apprentissage non-supervisé, on regarde quel neurone a le ‘mieux’ réagit au stimulus. Le ‘mieux’ correspond au résultat obtenu grâce à une fonction d’évaluation qui nous donne une valeur, que l’on compare ensuite aux autres sorties pour déterminer laquelle choisir.
5. Une fois la sortie choisi, on modifie son état (affectation d’une commande à une sortie). On peut aussi modifier les sorties voisines, pour faire converger le réseau (regrouper les commandes).
6. On recommence à 2 jusqu’à une condition d’arrêt (temps, précision…)

Plusieurs réseaux de neurones utilisent ce procédé. La seule chose qui les différencie est en fait la forme du réseau.

**Récapitulatif des caractéristiques de différents réseaux de neurones.**





Ces résultats ont étés obtenus par ce programme <http://www.demogng.de/> qui permet de tester les différents réseaux.

**Definitions :**

CHL\* : Competitive Hebbian Learning, algorithme qui fait se relier les neurones les uns avec les autres.

Neurones interagissant : Les neurones sont connectés dès le départ et s’influencent.

Neurones indépendants : Ils ne sont jamais reliés, donc la modification des poids d’un neurone n’influence pas le reste.

Neurones sociaux : Ils peuvent se lier et se délier.

Neurones s’accouplant : Des liens se créent mais ils restent en place.

Ces différentes caractéristiques influencent la vitesse de convergence du réseau ainsi que sa précision.

OK. Il faudra ensuite imaginer et expliquer comment ces différentes variantes pourraient être utilisées concrètement dans notre contexte.

Dans notre contexte, les neurones interagissant limitent la vitesse de convergence. Car si on augmente le poids d’un neurone de sortie, les autres neurones autour vont augmenter aussi, hors ceci modifiera le résultat de l’injection des autres commandes. Comme les neurones sont connectés en permanence, ce lien rend l’algorithme rigide et ne garantit pas un résultat minimal par rapport à d’autres réseaux.

Au-delà de ça, j’ai encore du mal à voir dans notre contexte, comment ces différents réseaux influencent le résultat et la vitesse pour l’obtenir.

* Algorithmes génétique

Le but des algorithmes génétiques est de se rapprocher au maximum d’une solution par l’évaluation des caractéristiques d’individus (solutions aléatoires ou pseudo-aléatoires).

1. On créé une population (liste d’individus) qui correspond à des solutions potentielles.
2. On évalue chaque individu (score).
3. En fonction de l’évaluation, on récupère les meilleurs caractéristiques que veux-tu dire par « meilleures caractéristiques » ? de chacun pour faire un croisement.
4. On applique des mutations sur les caractéristiques de chacun, sur les bons points comme les mauvais, la mutation peut être contrôlée et non totalement aléatoire.
5. On reprend au 2 jusqu’à une condition d’arrêt (temps, précision…)

Dans notre cas, il faut définir ce qu’est une ‘bonne’ caractéristique. La simple distance à parcourir pour un lot, n’assure pas que la distance totale pour tous les lots sera minimale.

En lisant cette dernière phrase, je ne suis pas sûr que tu aies compris.

Chaque individu « code » (représente) un élément de l’espace des solutions (c’est-à-dire, pour notre sujet, un choix pour l’ensemble des lots de commandes).

Exemple : 12 commandes, on souhaite faire 4 lots de 3 commandes. Les commandes sont A, B, C, …, L.

* Un individu I, pourra être BDFEGHLKACJI
* Pour évaluer cet individu, on peut par exemple faire la somme S des distances de chaque lot BDF, EGH, LKA, CJI
* Sur l’ensemble de la population, tu as donc { (Ii, Si) }

En règle générale, des croisements sont possibles. Un croisement (dans sa forme la plus simple car il y a d’autres façons de faire), c’est prendre deux individus, choisir une position dans le gène et intervertir ce qui se trouve à droite et à gauche de cette position pour obtenir 2 nouveaux individus :

* Individu 1 : ABCDEFGHIJKL
* Individu 2 : MNOPQRSTUVWX
* Position : 5
* Individus résultants :
  + ABCDERSTUVWX
  + MNOPQFGHIJKL

La question que tu dois te poser dans notre contexte : est-ce qu’un croisement est possible ? Intéressant ?

Croisement possible, il donne des resultat dans le domaine des solutions.

La mutation consiste à modifier une partie du génome de l’individu. Par exemple,

* passer de ABCDEFGHIJKL à AGCDEFBHIJKL
* ou bien de ABCDEFGHIJKL à ABCXEFGHIJKL
* (cela, en fonction du problème traité…)

La mutation doit faire s’intervertir 2 commandes qui ne sont pas dans le même lot pour qu’elle présente un interet. Il ne faut pas que la mutation sorte du cadre d’application (X est une commande qui n’existe pas)

Après, je te laisse réfléchir…

J’entendais par bonne caractéristique, la position dans le gène qui fait que l’individu obtient un ‘score’ plus proche de ce que l’on souhaite, seulement ici, la modification d’un gène influence forcément les autres. A partir de ça je ne vois pas comment choisir la position dans le gène de manière à ce que le croisement donne un résultat plus intéressant. Si on fait les mutations et le choix des gènes à croiser aléatoirement, on ne convergera pas forcément vers le résultat minimal.

* Le data clustering

Le data clustering ou partitionnement de données est une méthode d’analyse de données. Elle vise à regrouper les données en sous-ensembles qui partagent les mêmes caractéristiques.

K-means :

On partitionne aléatoirement ou non l’espace. Chaque partition dispose d’un noyau (peut être une moyenne ou quelque chose de plus complexe).

Le but est de regrouper dans k ensembles, n observations. Chaque observation est associée avec la partition ayant la moyenne (noyau) la plus proche.

Dans notre cas, créer au départ les k ensembles sans connaitre la forme est difficile. Il serait donc utile d’utiliser un autre algorithme permettant de déterminer ces noyaux. Le FL AME clustering par exemple, désigne des CSO (Cluster Supporting Objects) par la création d’un graph de voisinage qui permet de connaître la densité en chaque point.

FLAME Clustering :

Le FLAME Clustering est aussi utilisable tel quel. Les CSO donnent naissances a des clusters en se développant aux éléments les plus proches (les plus proches, mais pas forcément au sens de la distance). Dans notre contexte on pourrait représenter chaque commande par un point et grouper dans l’ordre par les autres commandes possédant les caractéristiques les plus proches. Pour que cet algorithme soit efficace, il faut établir correctement le graph de voisinage.

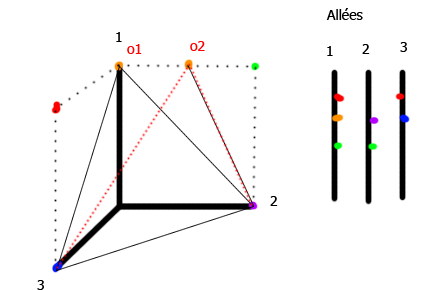
Si l’operateur passe dans une allée, il va forcément la parcourir dans sa totalité. Dans notre cas, une commande pourrait être un point à n-dimensions ou chaque dimension représenterait une allée. Ainsi on met un 1 dans le vecteur si on passe dans l’allée et 0 sinon.

Par contre si on a : 3 commandes A, B et C contenants chacune 1 article, la distance sera la même entre chacune des commandes peu importe la position de l’allée. Ceci se règle facilement si on ordonne les allées (les composantes du vecteur) en fonction de leur proximité et qu’entre les 1 et les 0, on fasse une interpolation (un peu comme une corde posée sur le dessus). L’interpolation peut être linéaire ou non.

Soit la commande A avec un entrepôt de 16 allées : A [0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,0 ,0]

Sur l’axe horizontal, on a le numéro de l’allée. En vertical, 1 ou 0 en fonction du passage ou non dans l’allée. La courbe bleu est celle contenant uniquement des 0 et des 1, celle en rouge contient une interpolation linéaire.

Cette interpolation permet de régler le problème des 3 commandes ayant uniquement 1 article chacune mais dont la distance qui les sépare n’est pas égale pour les 3. Grâce à ça, on est capable de grouper les commandes non seulement en fonction des allées parcourues, mais aussi si on passe devant pour aller à une autre allée. Ce qui veut dire que le groupement se fait de manière moins rigide. Ainsi on constitue des lots de manière plus intelligente en prenant en compte la distance à parcourir pour qu’elle soit minimum.



Ici en orange, violet et bleu, les commandes A, B et C. o1 représente la commande avec les coordonées en 1 et 0, o2 avec l’interpolation. On constate que la distance entre la commande orange et la commande violette est plus courte qu’avec la commande bleue.

Une fois les coordonnées du point (la commande) calculées, on peut calculer la distance euclidienne avec les autres points pour établir le graph de proximité.

J’ai trouvé une implémentation en C du flame clustering que j’ai aussitôt testé. Sur 500 points en 2 dimensions, le résultat était instantané, sur 5000 il a pris moins d’une minute. Dans notre cas, on aura autant de dimensions que d’allées donc le calcul de la distance entre chaque point sera plus long. Ce calcul est cependant parallélisable.

Si on a 1000 commandes à traiter, calculer toutes les distances reviendra à en calculer 499500 (dénombrement du nombre de combinaisons de 2 éléments parmi 1000). Ceci étant rapide à calculer pour les ordinateurs actuels. Ensuite un simple algorithme de tri sur les branches du graph permet d’obtenir les voisins les plus proches très rapidement.

* Algorithme par séparation et évaluation.

Un algorithme par séparation et évaluation consiste à diviser le problème principal dont on cherche à trouver une solution, en plusieurs petits problèmes. Ainsi ont divise l’ensemble S des solutions possibles en plusieurs sous-ensembles de solutions qui recouvrent la totalité de l’ensemble S. Il suffit ensuite de trouver une solution à chaque ‘petit’ problème.

Ce type d’algorithme s’applique particulièrement aux problèmes d’optimisation combinatoire.

Dans notre contexte, je ne vois pas de moyen de diviser le problème, étant donné que la composition d’un lot de commande influence forcément les autres lots.

Je n’abandonne cependant pas l’idée, je retravaillerais dessus cette semaine pour voir si je peux en tirer quelque chose.