Approches d'optimisation de parcours dans un hypermarché

Ismahène Hadj Khalifa[1], Abdelkader El Kamel[2]

1. Institut Supérieur des Arts de Multimédia
Université de la Mannouba, 2010, Manouba, Tunisie
ismahenhk@yahoo.fr

2. Ecole Centrale de Lille
BP 48, 59651 Villeneuve d'Ascq, France
abdelkader.elkamel@ec-lille.fr

RÉSUMÉ. Dans cet article, nous nous intéressons au problème d'optimisation de parcours dans un hypermarché. Nous proposons, dans ce cadre, une méthodologie pour calculer les distances entre articles de l'hypermarché pris deux à deux pour estimer le chemin le plus court pour trouver les articles existants dans la liste de courses.

ABSTRACT. In this paper, we concentrate on the hypermarket itinerary optimization. We propose a methodology to calculate the distance between each two items in the hypermarket before implementing the optimization algorithm, to estimate the shortest path to pick up the items existing in the shopping list.

MOTS-CLÉS : système de navigation indoor, optimisation de parcours, algorithme de Christofides modifié, algorithme du Plus Proche Voisin adapté, méthode de recherche tabou avec ensembles.

KEYWORDS: indoor navigation system, path optimization Christofides algorithm modified, Nearest Neighbor algorithm adapted, tabou search method with sets.

DOI:10.3166/JESA.49.31-53 © Lavoisier 2016

1. Introduction

Cet article fait partie d'une étude dont l'objectif est de concevoir un système de navigation intelligent et générique pour aider les personnes à se déplacer dans les grandes surfaces et ce dans la logique de développement de l'intelligence ambiante et du u-commerce (commerce ubiquitaire) préconisé par le Pôle de Compétitivité des Industries du Commerce (PICOM). Il s'agit d'un système de géolocalisation multi-canal qui assistera l'utilisateur tout au long de son parcours, l'aidera à préparer ses achats, choisir ses produits, les localiser dans le magasin et obtenir des conseils

et des suggestions. Cependant, ce système ne peut fonctionner sans un calculateur permettant de déterminer le chemin à parcourir pour récupérer les articles existants dans la liste de courses.

Après une étude de conception du système, nous nous concentrons sur l'optimisation de l'itinéraire. Pour ce faire, nous commençons, dans la section 2, par une revue de la littérature sur les méthodes d'optimisation de l'itinéraire en temps réel. Ensuite, nous présentons, dans la section 3, une méthodologie pour résoudre le problème d'optimisation de parcours dans un hypermarché. Nous proposons une méthode pour la détermination des distances entre des articles de l'hypermarché pris deux à deux. Dans la section 4, nous présentons une formulation mathématique du problème. Ensuite, nous exposons les différentes méthodes d'optimisation que nous avons adaptées à notre problème. Nous présentons, dans la section 5, les tests et les résultats. Enfin, une proposition de solutions de re-calcul du chemin en cas de changement de direction fera l'objet de la section 6.

2. État de l'art sur les méthodes d'optimisation d'itinéraire en temps réel

Dans la littérature, le problème d'optimisation de parcours en temps réel est évoqué, essentiellement, dans le domaine du transport de marchandises en utilisant un ou plusieurs modes de transport, et en particulier le problème de tournée de véhicules dynamique, pour la gestion des imprévus, des perturbations et des cas d'urgence. En outre, les applications du GPS (*Global Positioning System*) sont fondées sur le principe d'optimisation de l'itinéraire. Dans la suite, nous exposons quelques méthodes de résolution du problème de recherche du chemin le plus court pour le problème de tournée des véhicules dynamique ainsi que le guidage des personnes par GPS.

2.1. Problème de tournée de véhicules dynamique

Le problème de tournée de véhicules dynamique ou DVRP (*Dynamic Vehicle Routing Problem*) est un des problèmes de tournées de véhicules où toutes les demandes pour tous les clients ne sont pas connues d'avance. Ainsi, on peut avoir, par exemple, l'apparition d'un nouveau client en cours de journée. Dans ce cas, le décideur doit changer la planification des véhicules en réponse aux nouvelles demandes qui arrivent au cours du temps (Pillac *et al.*, 2013 ; Kilby *et al*., 1998 ; Gendreau *et al*., 1998).

Bianchi (2000) ainsi que Larsen (2001) ont présenté plusieurs méthodes de résolution du problème DTRP (*Dynamic Traveling Repairman Problem*), un cas du DVRP. Parmi ces méthodes, nous citons quelques heuristiques simples utilisées : *First Come First Serve* (FCFS), *Stochastic Queue Median* (SQM), *Nearest Neighbor* (NN), *PARTitioning policy* (PART), *Traveling Salesman Problem strategy* (TSP).

Certaines heuristiques simples ont été appliquées sur de petites instances de DTRP et deviennent instables dès que le nombre de demandes dynamiques augmente.

Par ailleurs, une heuristique d'insertion a été présentée par (Roy *et al.*, 1984). C'est une heuristique qui peut être très rapide et donc utilisable pour des problèmes de nature dynamique tels que les problèmes de DVRP. Cette méthode re-optimise simplement les tournées de véhicules quand les nouvelles données deviennent disponibles. Elle cherche ainsi à insérer le nouveau client dans la meilleure position des tournées courantes. L'heuristique d'insertion semble intéressante pour insérer les nouveaux articles entrés par l'utilisateur dans la liste de courses et recalculer le nouveau chemin.

2.2. Problème de guidage des utilisateurs par GPS

Pour des raisons commerciales, peu de travaux ont été publiés sur les méthodes d'optimisation d'itinéraires dans un GPS. Cependant, des travaux ont été réalisés pour la conception d'un système de navigation GPS pour le guidage de l'utilisateur dans son déplacement en ville (Pérez-Ponce *et al.*, 2004 ; Abboud *et al*., 2004). Les auteurs de ces deux travaux ont choisi l'algorithme de Dijkstra pour l'optimisation de la distance entre deux points.

Pérez-Ponce *et al.* (2004) ont conçu un système de guidage des personnes ayant une déficience visuelle pour se déplacer en ville sans avoir besoin d'une aide extérieure. Le système est composé d'un GPS, d'un PDA (*Personal Digital Assistan*t) et d'un kit main-libre Bluetooth. Le GPS est utilisé pour recueillir les informations sur la position de l'utilisateur (latitude, longitude). L'application utilise l'algorithme de Dijkstra pour le calcul du chemin le plus court et le moins risqué. Il compare la trajectoire que fait l'utilisateur avec celle indiquée par le système pour le prévenir en cas de dépassement. Quand il arrive au coin de la rue, une alerte est déclenchée pour l'avertir.

Par ailleurs, Abboud *et al.* (2004) ont créé un système de navigation GPS en temps réel. Il détermine la position de l'utilisateur sur la carte de Beirut en utilisant un récepteur GPS. Cette carte est une carte GIS (*Geographic Information System*) qui contient des données spatiales et des attributs. Ces attributs contiennent des informations sur les données spatiales telles que le nom de la route, le nœud de départ et le nœud d'arrivée, la longueur de la route, la durée de parcours de la route, le nom de la rue… Le chemin le plus court est défini comme étant le chemin ayant le coût le plus faible. Le coût peut être soit la somme des distances entre les rues soit la somme des temps nécessaires pour parcourir chaque rue. Il existe plusieurs algorithmes utilisés pour le calcul du chemin le plus court tels que : l'algorithme de Dijkstra, l'algorithme de Bellman Ford, l'algorithme de tri topologique et l'algorithme A Star (A*). Le choix de l'algorithme dépend de la taille de la carte. L'algorithme de Bellman Ford peut prendre beaucoup plus de temps pour trouver la solution mais ses résultats peuvent être similaires ou même meilleures que celles utilisant l'algorithme Dijkstra. Quant à l'algorithme de tri topologique, il peut trouver le chemin le plus court mais il peut ne pas être le meilleur pour cette étude. L'algorithme A* peut trouver des résultats similaires à ceux de l'algorithme Dijkstra avec moins de mémoire mais son implémentation est beaucoup plus difficile pour trouver le chemin le plus court. Pour toutes ces raisons, Abboud *et al.* (2004) ont

retenu l'algorithme Dijkstra pour l'implémentation de l'application surtout qu'il est le plus utilisé dans les systèmes GIS pour le calcul de la distance entre deux points. Le temps de réponse dépend du nombre de nœuds entre le point de départ et le point d'arrivée. L'utilisateur entre la destination et l'algorithme Dijkstra calcule le chemin le plus court. Cette application guide l'utilisateur en lui indiquant les directions et recalcule l'itinéraire en temps réel à chaque fois où il change de direction.

Cependant, l'algorithme de Dijkstra ainsi que l'algorithme A* permettent de calculer le chemin le plus court entre deux nœuds uniquement. C'est la raison pour laquelle on ne peut pas les utiliser directement pour résoudre le problème d'optimisation de parcours dans un hypermarché puisque l'algorithme doit optimiser le chemin passant par tous les articles existants dans la liste de courses.

3. Proposition d'une méthode d'optimisation d'itinéraire dans un hypermarché

Nous rappelons le principe de notre système de navigation dans les hypermarchés : l'utilisateur entre sa liste d'articles et le système calcule le chemin le plus court. En cas d'ajout ou de suppression d'un article ou encore si l'utilisateur change de direction, le nouveau itinéraire sera recalculé dynamiquement. Pour cela, nous avons besoin d'une base de données facile à mettre à jour.

Nous avons proposé dans (Hadj Khalifa *et al.*, 2010a ; 2010b) une méthodologie pour résoudre ce problème d'optimisation. Les principales étapes de cette méthodologie sont :

1. Création de la base de données du magasin.
2. Détermination des distances entre les articles deux à deux et leur insertion dans la base de données
   i. Division du magasin en zones selon la disposition des gondoles
   ii. Attribution de repères dans chaque zone
   iii. Proposition d'une méthode de calcul des distances entre les articles deux à deux
   iv. Insertion des distances dans la base de données
3. Tests des méthodes d'optimisation pour déterminer le chemin le plus court.
4. Choix de la méthode la plus appropriée à notre problème.
5. Proposition d'une méthode pour recalculer le chemin le plus court en cas de changement de direction.

Nous détaillons, dans la suite, la méthodologie de résolution du problème d'optimisation de parcours dans un hypermarché et ses différentes étapes.

3.1. Création de la base de données

La complexité du problème d'optimisation réside dans son aspect dynamique (ajout ou suppression d'articles dans la liste, changement de direction, très grand nombre d'articles dans l'hypermarché qui peuvent changer de place d'une période à

une autre…). Pour cela, nous avons créé une base de données qui détaille les différents éléments d'un magasin ce qui facilitera sa mise à jour plus tard.

Nous rappelons qu'un magasin contient des gondoles et chaque gondole peut appartenir à un ou deux rayons. Chaque rayon est divisé en sous-rayons qui contiennent différents types de produits. Afin de faciliter le remplissage de la base de données, nous avons défini les valeurs entrées suivantes :

Informations sur les gondoles : la longueur de la gondole, sa largeur, la distance entre les gondoles, la zone et le/les rayon(s) auxquels elle appartient, le nombre de parties qu'elle contient et qui dépend du maillage choisi.

Informations sur les articles : la référence de l'article, le nom du sous-rayon, le nom du type de produits ainsi que le nom du sous-type de produits auxquels il appartient.

À partir de ces informations, nous avons effectué des programmes qui permettent de remplir le reste des champs. Dans la sous-section suivante, nous nous intéressons à la détermination des distances entre des articles pris deux à deux.

3.2. Détermination des distances entre les articles deux à deux

Avant de chercher le chemin le plus court à parcourir pour trouver les articles existants dans la liste de courses, il faut déterminer les distances entre articles pris deux à deux. Compte tenu du grand nombre d'articles et pour faciliter le calcul des distances, nous proposons de diviser le magasin retenu en zones selon la disposition des gondoles dans le magasin comme le montre la figure 1. De plus, nous considérons les types de produits et non les articles. En d'autres termes, nous allons rassembler les articles voisins appartenant au même type de produits et calculer seulement la distance entre les différents types de produits. Au cas où un type de produit contient plusieurs articles distants, on considérera le sous-type de produits.

Puis nous mettons des repères sur chaque gondole. L'attribution des repères dépend de la longueur de la gondole, la disposition des produits et le maillage choisi. Compte tenu du grand nombre d'articles existants dans le magasin, nous avons travaillé sur la zone 1 et la zone 2 pour tester nos algorithmes. La figure 2 représente ces deux zones avec des repères. Nous avons choisi de diviser le repère en deux parties afin de faciliter le calcul des distances :

– une partie qui représente la référence de l'allée où se trouve l'article. Cette partie est appelée RefPartie dans la base de données (exemples de référence de l'allée : 100, 72, 69BIS…)

– une partie qui peut prendre des valeurs sous forme de lettres différentes de chaque côté de la gondole. Cette partie est appelée CodePartieDansGondole dans la base de données.

Les types de produits sont rassemblés et chaque ensemble de types de produits est attribué à un repère sur le plan (CodeReperePlan).

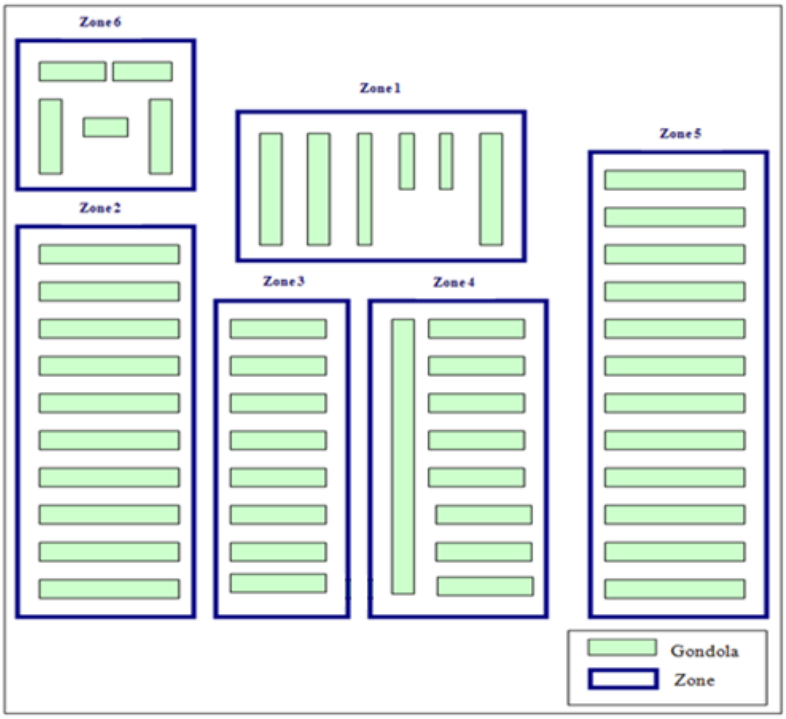

Figure 1. Division du magasin en zones

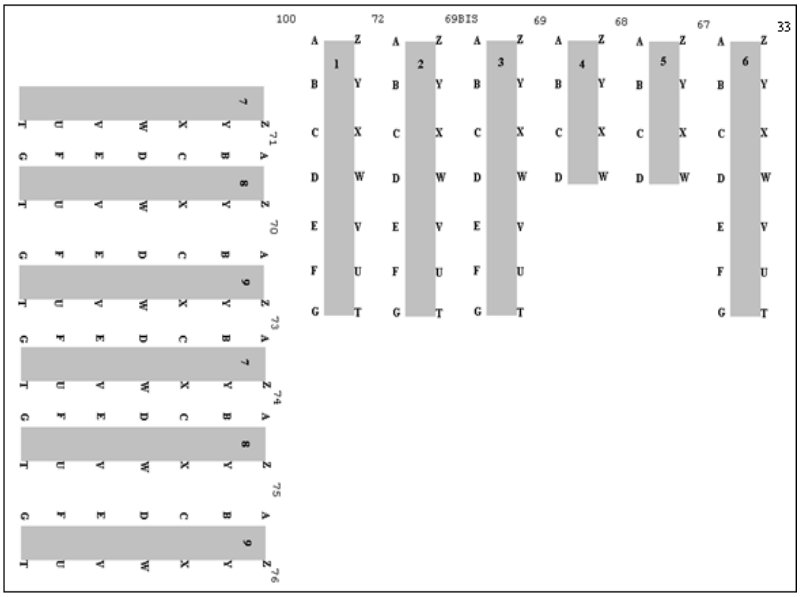

Figure 2. Attribution de repères sur les gondoles

Dans la suite, nous déterminons les distances entre deux articles de la même zone ensuite entre deux articles existants dans deux zones différentes.

3.2.1. Détermination des distances entre deux articles appartenant à la même zone

```
Gondole1 : code de la gondole qui contient l'article A1
Gondole2 : code de la gondole qui contient l'article A2
cod1 : CodePartieDansGondole de l'article A1
cod2 : CodePartieDansGondole de l'article A2
ref1 : RefPartie de l'article A1
ref2 : RefPartie de l'article A2
LongG1 : longueur de la gondole 1
LongG2 : longueur de la gondole 2
LargG1 : largeur de la gondole 1
LargG2 : largeur de la gondole 2
NbrPartG1 : nombre de parties dans la gondole 1
NbrPartG2 : nombre de parties dans la gondole 2
DistGond[G1,G2] : distance entre la gondole 1 et la gondole 2
EH1 : Extrémité en haut de la gondole 1 du côté de A1
EB1 : Extrémité en bas de la gondole 1 du côté de A1
dist[A1,EH1]: distance entre A1 et EH1
dist[A1,EB1]: distance entre A1 et EB1

Si (Gondole1≠Gondole2) alors
  Si (ref1≠ref2)
    Si (A1 se trouve à gauche de Gondole1) et (A2 se trouve à droite de
       Gondole2)
        Distance1=(LongG1/NbrPartG1)*dist[A1,EH1]+(LargG1/2)+(LargG2/2)+
                  DistGond[G1,G2]+(LongG2/NbrPartG2)* dist[A2,EH2]

        Distance2=(LongG1/NbrPartG1)*dist[A1,EB1]+(LargG1/2)+(LargG2/2)+
                  DistGond[G1,G2]+(LongG2/NbrPartG2)* dist[A2,EB2]

         Si (Distance1-Distance2<=0) alors
             Distance = Distance1
         Sinon
             Distance = Distance2
```

Figure 3. Extrait de l'algorithme de calcul des distances

La distance entre deux articles de la même zone est fonction de la longueur et de la largeur de la gondole à laquelle appartient chaque article, la distance entre ces gondoles, le nombre de parties de chaque gondole… Ainsi, notre algorithme a besoin de ces variables qu'il récupère de la base de données. Cependant, il existe plusieurs cas à traiter. Ces cas dépendent essentiellement de l'emplacement de :

– la gondole contenant l'article 1 (Gondole1) par rapport à la gondole contenant l'article 2 (Gondole2) ;

– la RefPartie de l'article 1 (A1) par rapport à la RefPartie de l'article 2 (A2) ;

– le CodePartieDansGondole de A1 et A2 par rapport à l'extrémité des gondoles sur lesquelles ils se trouvent.

La figure 3 illustre un exemple de calcul de distances dans le cas où les deux articles n'appartiennent pas à la même gondole et ne se trouvent pas sur la même allée. Dans ce cas, on doit différencier 4 éventualités selon l'existence du repère sur la partie à droite ou la partie à gauche de sa gondole. La figure 3 traite une de ces éventualités dans le cas où le repère de l'article 1 se trouve dans la partie à gauche de la gondole 1 et le repère de l'article 2 se trouve dans la partie à droite de la gondole 2. Comme il peut exister deux chemins pour aller du repère 1vers le repère 2, on doit choisir la distance du chemin le plus court.

3.2.2. Détermination des distances entre deux articles appartenant à deux zones différentes

Pour déterminer la distance entre les articles appartenant à deux zones différentes, nous avons eu l'idée de représenter les repères selon des coordonnées cartésiennes et nous avons défini un point à l'entrée du magasin comme le point d'origine (0,0). Cette méthode facilite le calcul des distances pour ce cas puisqu'on ajoute la distance euclidienne entre les extrémités de chaque zone.

Soient deux articles A1 etA2 appartenant respectivement à la zone 1 et à la zone 2. Soit E1 l'extrémité de la zone 1 appartenant à l'allée commune entre la zone 1 et la zone 2 et E2 l'extrémité de la zone 2 appartenant à l'allée commune entre la zone 1 et la zone 2.

La distance d[A1, A2] entre A1 et A2 est égale à :

$$d[A1, A2] = d[A1, E1] + d[E1, E2] + d[E2, A2]$$

avec d[E1, E2] la distance euclidienne calculée à partir des coordonnées de E1 et E2.

Comme le montre la figure 4, pour aller à l'allée commune, il faut passer par l'extrémité EH1, en haut, ou par l'extrémité EB1, repère en bas.

La distance d[A1, A2] est égale au Minimum(d1, d2) telle que :

$$d1 = d[A1, EH1] + d[EH1, E2] + d[E2, A2]$$

$$d2 = d[A1, EB1] + d[EB1, E2] + d[E2, A2]$$

L'extrémité E2 appartient à la même gondole et à la même allée que A1.

Enfin, nous insérons toutes ces distances dans la base de données, pour les utiliser plus tard dans la partie optimisation. Par ailleurs, nous déterminons le chemin entre deux articles. Ce chemin représente l'ensemble des nœuds par lesquels passe l'utilisateur pour aller d'un article à un autre.

Après avoir déterminé et inséré les distances dans la base de données, nous nous concentrons dans la section suivante sur les méthodes d'optimisation de parcours dans un hypermarché.

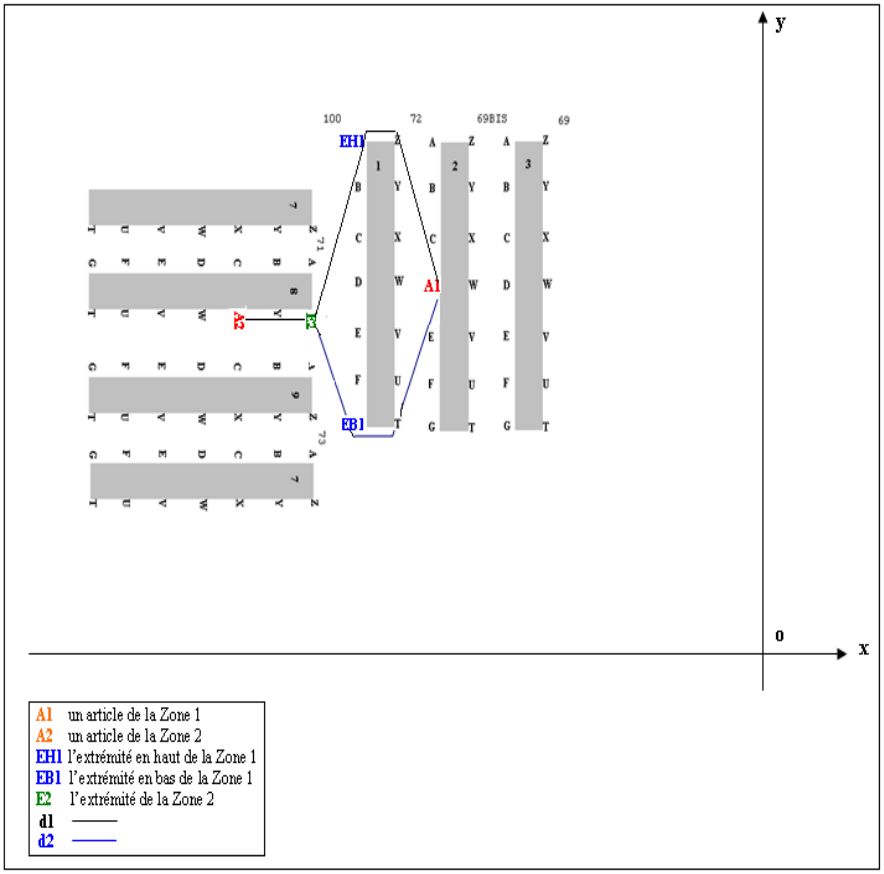

Figure 4. Détermination des distances entre deux articles appartenant à deux zones différentes

4. Approches d'optimisation de l'itinéraire dans un hypermarché

L'objectif de notre problème est de guider les clients dans un magasin en leur montrant le chemin pour trouver tous les articles ajoutés à leur liste de courses en minimisant la distance à parcourir. Pour ce faire, nous proposons, tout d'abord, une formulation mathématique du problème. Ensuite nous exposons les différentes méthodes d'optimisation que nous avons adaptées à notre problème. Enfin, nous présentons les tests et les résultats obtenus.

4.1. Présentation du problème

À l'entrée du magasin, l'utilisateur constitue sa liste d'articles soit en tapant leurs noms, soit en les récupérant à partir du catalogue en cours. Ensuite, notre application

récupère, de la base de données, sa position actuelle qui représente le nœud de départ. Notre problème consiste à trouver le chemin le plus court à parcourir à partir de la position actuelle de l'utilisateur et passant, une seule fois, par tous les articles existants dans la liste de courses.

Tableau 1. Analogie entre le problème du voyageur de commerce et notre problème

Paramètres	Problème de voyageur de commerce	Notre problème
Ensemble des nœuds	Les villes	Les articles entrés par l'utilisateur
Contraintes	- Le nœud de départ est le même nœud d'arrivée - Passer par toutes les villes une et une seule fois	- Le nœud de départ n'est pas le même nœud d'arrivée - Le point initial dépend de la position donnée par le système de localisation - Passer par tous les articles une et une seule fois
Critères	Distance, temps, coût…	Distance, temps, passage par les rayons…
Solution	Circuit hamiltonien	Chemin hamiltonien

Dans la littérature, ce problème d'optimisation est similaire au problème de voyageur de commerce (PVC) qui consiste à trouver la plus courte tournée permettant de visiter n villes et de revenir au point de départ en ne visitant chaque ville qu'une seule fois (Johnson et *al.*, 2002 ; Lacomme et *al.*, 2003). Étant donné que le nœud de départ (position actuelle de l'utilisateur) de notre problème n'est pas le même nœud d'arrivée (caisse) contrairement au PVC, le résultat de notre problème sera un chemin hamiltonien alors que celui du PVC est un circuit hamiltonien. Le tableau 1 représente une comparaison entre les deux problèmes.

4.2. Formulation mathématique du problème

Notre objectif est de guider les clients dans un magasin afin qu'ils trouvent tous les articles ajoutés à leur liste de courses en minimisant la distance à parcourir. Nous présentons, dans la suite, une formulation mathématique du problème par analogie avec le PVC. Notre problème est caractérisé par les paramètres suivants :

– N : Nombre des nœuds représentant les articles existants dans la liste de courses du client,

– Ni : Article au nœud i,

– c : Le nœud final qui représente la caisse

– Dij : distance entre le nœud i et le nœud j, si Dij = ∞ alors le chemin entre i et j n'existe pas,

– i = 0..N : indice des nœuds prédécesseurs,

– j = 0..N : indice des nœuds successeurs.

– Il y a un seul client qui cherche les articles existants dans sa liste de courses,

– Un nœud n'est visité par le client qu'une et une seule fois,

– Il y a un seul nœud de départ qui représente la position actuelle du client,

– Le client commence le trajet à partir de sa position actuelle et se dirige vers la caisse à la fin.

$$Xij = \begin{cases} 1 & \text{Si le client part du nœud i et se dirige vers le nœud j} \\ 0 & \text{Sinon} \end{cases}$$

La fonction à optimiser est la suivante :

$$\text{Minimiser F} = \sum_{i \in N} \sum_{j \in N} D_{ij} X_{ij} \tag{1}$$

Sous les contraintes :

$$\sum_{j \in N} X_{0j} = 1 \tag{2}$$

$$\sum_{i \in N} X_{ic} = 1 \tag{3}$$

$$\sum_{i \in N} X_{iu} + \sum_{j \in N} X_{uj} = 2 \qquad \forall u \in N \tag{4}$$

L'équation (2) assure que chaque sommet n'est visité qu'une seule fois par le client.

L'équation (3) assure que client se dirige vers la caisse une fois.

L'équation (4) assure la continuité d'une tournée par le client : le sommet visité doit impérativement être quitté.

4.3. Proposition de méthodes de résolution du problème

Nous considérons dans notre problème que le nombre d'articles ajoutés par le client à sa liste de courses ne dépasse pas 100 alors notre problème est similaire au PVC avec 100 villes. Ainsi, le problème est considéré, par analogie, comme un problème NP difficile (Lacomme *et al.*, 2003). Plusieurs recherches ont été faites pour résoudre ce problème. Parmi ces méthodes, nous trouvons les méthodes exactes

et les méthodes approchées. Nous retenons l'algorithme de *Branch and Bound*, l'algorithme de Christofides et l'algorithme du Plus Proche Voisin (PPV). Nous décrivons, dans la suite, ces méthodes pour résoudre le problème d'optimisation de parcours dans un hypermarché en les adaptant au problème et nous proposons une amélioration des résultats avec une recherche tabou.

4.3.1. Algorithme de Branch and Bound

Cet algorithme, appelé aussi l'algorithme de séparation et évaluation progressive, se base sur l'énumération et l'évaluation des solutions possibles. Il construit une arborescence et évalue, pour chacune des branches, la possibilité de trouver la solution optimale. Ensuite, seuls les sommets qui peuvent mener à une solution intéressante sont examinés pour éviter de parcourir entièrement l'arbre des solutions (Applegate *et al.*, 2007 ; Lacomme *et al.*, 2003).

Nous avons testé cet algorithme en utilisant AMPL comme langage de modélisation et CPLEX comme solveur. L'algorithme de *Branch and Bound* a été implémenté par analogie au problème de TSP. Il est vrai que cette méthode donne des solutions optimales mais dès que le nombre des articles devient grand, le temps de calcul devient important. C'est pour cette raison que la méthode de B*ranch and Bound* n'a pas été retenue.

4.3.2. Algorithme de Christofides

L'algorithme de Christofides (1976) est un algorithme basé sur l'Arbre Recouvrant de Poids Minimal (ARPM) (*Minimum Spanning Tree*, MST). Il existe plusieurs algorithmes pour déterminer cet arbre dont on peut citer les plus utilisés (Lacomme *et al.*, 2003 ; Held *et al.*, 1969) :

– l'algorithme de Prim qui consiste à choisir, arbitrairement, un sommet et à faire croître un arbre à partir de ce sommet. Chaque augmentation se fait de la manière la plus économique possible ;

– l'algorithme de Kruskal qui consiste à ranger, tout d'abord, par ordre de poids croissant les arêtes d'un graphe, puis à retirer une à une les arêtes selon cet ordre et à les ajouter à l'ARPM cherché tant que cet ajout ne fait pas apparaître un cycle dans l'ARPM. Le principe de cet algorithme est le suivant :

1. Construire L'ARPM à partir de l'ensemble des nœuds
2. Déterminer un couplage de coût minimum sur les sommets de degré impair et l'ajouter à l'ARPM.
3. Déterminer un cycle eulérien à partir du graphe obtenu
4. Raccourcir la tournée si elle passe plus d'une fois par un même sommet.

Nous avons utilisé l'algorithme Prim pour déterminer l'ARPM. A partir d'un sommet initial représentant la position actuelle de l'utilisateur, cet algorithme fait croître un arbre de manière que la somme des poids des arcs soit minimale.

Par ailleurs, puisque notre solution est un chemin hamiltonien et non pas un circuit hamiltonien, nous proposons une modification à la méthode de Christofides :

nous supprimons, de la solution, le dernier arc relié au nœud initial pour obtenir un chemin et nous relions le dernier nœud au nœud de la caisse la plus proche.

4.3.3. Algorithme du Plus Proche Voisin

Cet algorithme consiste à construire un cycle en faisant croître une chaîne. On part d'un sommet arbitraire à partir duquel on va au sommet voisin le plus proche, puis de celui-là à son plus proche voisin non visité jusqu'à ce que tous les sommets soient parcourus et on revient au départ. Cependant, les solutions qu'il fournit peuvent être mauvaises. En effet, cette procédure commence généralement par faire de très bons choix en sélectionnant des arêtes de poids faible, mais, vers la fin, la chaîne doit ensuite aller visiter des sommets qui restent, et des distances importantes peuvent alors être rajoutées à la chaîne.

Nous avons testé l'algorithme du Plus Proche Voisin (PPV) ainsi que l'algorithme de Christofides modifié. Nous avons obtenu des solutions réalisables mais pas toujours optimales et parfois même on peut revenir à un rayon déjà visité. Pour améliorer ces solutions et les adapter à la structure d'un magasin, nous avons ajouté à la méthode du PPV, une règle de passage par tous les articles d'un même rayon avant de passer aux autres.

4.3.4. Algorithme du Plus Proche Voisin Adaptée

La méthode du Plus Proche Voisin Adaptée est basée sur l'algorithme du PPV. Nous avons ajouté à ce dernier une règle de passage par tous les articles d'un même rayon avant de passer aux autres. De plus, nous avons proposé de laisser les produits fragiles ainsi que les surgelés (cas des magasins agroalimentaires) à la fin du parcours pour ne pas les abîmer. Cette solution peut être parfois un peu plus longue que les autres mais elle paraît plus cohérente pour l'utilisateur et répond plus à ses besoins. De plus, à la fin, le client se dirige vers la caisse la plus proche du dernier article visité.

Cependant, la méthode du PPV adaptée (PPVA) se base sur une méthode qui fait généralement de très bons choix en sélectionnant d'abord des arêtes de poids faible, mais doit être modifiée pour visiter les sommets restants. Ainsi, des distances importantes peuvent être rajoutées au chemin. Pour cela, nous proposons d'améliorer la solution en appliquant la méthode de recherche tabou.

4.3.5. Algorithme de recherche tabou

La méthode de recherche tabou consiste à examiner les solutions voisines de la solution courante en utilisant un ou plusieurs mouvements déterminant la prochaine solution. Ensuite, l'algorithme enregistre la meilleure solution parmi les voisins. Cette méthode est caractérisée par la présence d'une liste tabou qui permet d'éviter au cours des itérations de retomber sur un minimum local (Gendreau, 1994 ; Golver, 1990 ; Golver et *al.*, 1993).

Il convient de décrire plus précisément les différentes composantes de la méthode. Ces paramètres portent sur le choix de la solution initiale, la structure du

voisinage, la restriction tabou et le critère d'aspiration. Nous présentons, dans la suite, la mise en application de la méthode de recherche tabou adaptée pour la résolution du problème d'optimisation du parcours dans un hypermarché.

Solution initiale

Dans la phase d'initialisation, nous retenons la solution construite grâce à l'algorithme du PPVA. Cette solution respecte les règles de passage par tous les articles existants au même rayon et laisse les produits fragiles à la fin du parcours. Puis le processus itératif de recherche tabou des nouvelles solutions admissibles commence, jusqu'à ce qu'un critère d'arrêt soit satisfait.

Structure du voisinage

Il est clair que l'exploration d'un voisinage de grande dimension à chaque nouvelle itération du processus augmente à la fois les chances de converger vers l'optimum global et la complexité de la recherche. Afin d'obtenir un équilibre entre qualité et rapidité de la recherche, deux mouvements d'échange sont retenus dans la création du voisinage de la solution courante.

– la permutation des deux éléments i et i+k avec k allant de 2 à N-2 et N nombre de nœuds ;

– la permutation des deux éléments i et N-i-1

Dans le cas où k = 1 cela revient à l'inversion de deux éléments successifs dans la solution initiale. La figure 5 représente le premier mouvement dans le cas où k = 1 et la figure 6 représente le deuxième mouvement.

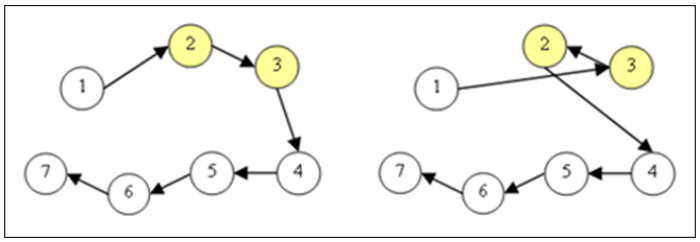

Figure 5. Mouvement d'échange entre deux éléments successifs

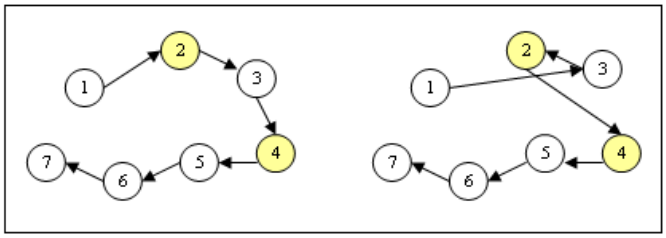

Figure 6. Mouvement d'échange entre l'élément i et l'élément N-i-1

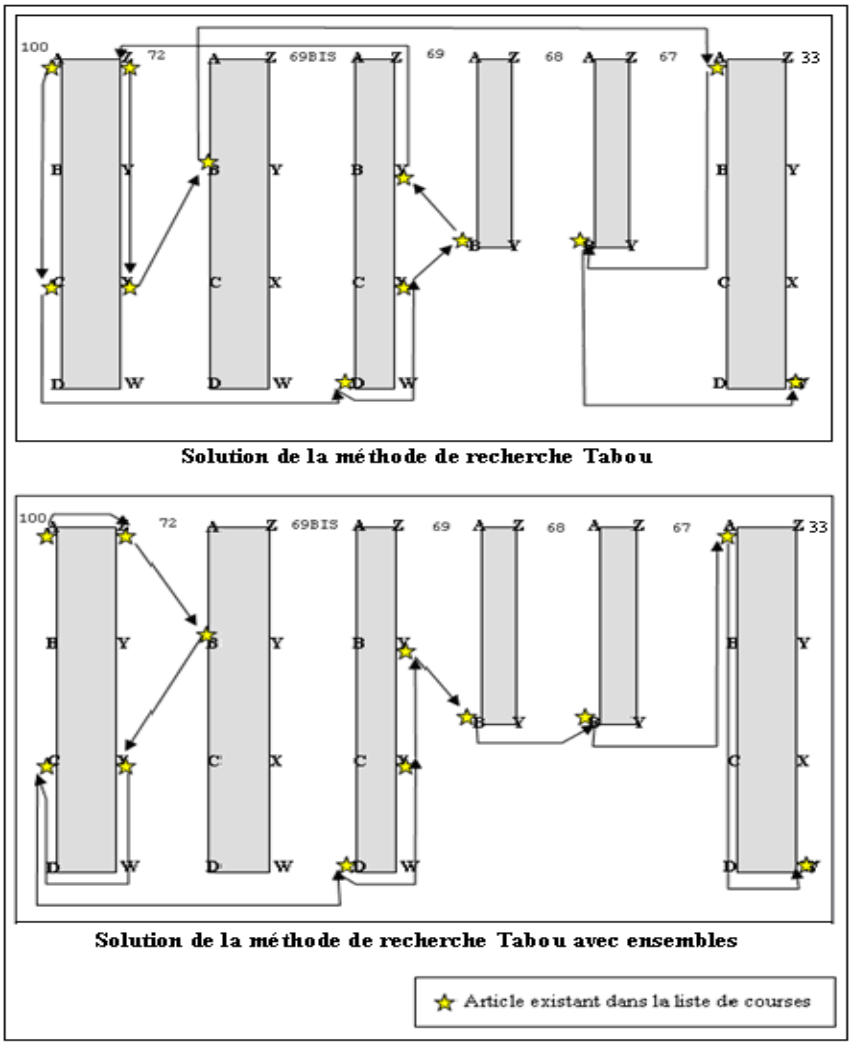

Figure 7. Amélioration de la solution Tabou après définition des ensembles

Définition des ensembles

Notons que l'utilisation de ces mouvements fournit un voisinage très peuplé, qui peut contenir des solutions non réalisables qui ne respectent pas la règle de passage aux articles appartenant à la même allée et ensuite passer aux autres. Dans le but de diminuer le temps de calcul et d'éviter ce type de solutions, nous proposons de vérifier cette règle avant de faire un mouvement d'échange. Cela permettra à l'algorithme d'avoir des solutions réalisables mais diminuera l'intérêt des mouvements et par la suite on risque de tomber rapidement dans un minimum local.

Pour cela, nous proposons de constituer des ensembles de nœuds. Chaque ensemble constitue un élément qui subit un mouvement. Cet ensemble contient des nœuds appartenant à la même allée et qui peuvent être permutés ou inversés entre eux. Chaque nœud appartenant à la tête de gondole constitue un singleton à part. Ainsi, les mouvements seront effectués entre ensembles, d'une part, et entre les nœuds appartenant à un même ensemble d'autre part.

La figure 7 illustre un exemple de parcours résolu en utilisant la méthode de recherche Tabou. Le coût de cette solution est égal à 132. Nous pouvons remarquer l'amélioration de cette solution en permutant les ensembles suivants: {Z72}, {X72, B72}, {D69BIS}, {B69}, {X69, Y69}, {B68}, {C101}, {A67}. Le nouveau coût de la solution devient 105. A100 représente le point de départ et W33 le point d'arrivée.

Restriction tabou

La restriction tabou que nous avons choisie d'utiliser est compatible avec la construction du voisinage que nous avons retenue. En effet, lorsque l'exécution d'un mouvement est réalisée à l'itération t, un sommet ne peut plus être soumis à un autre mouvement jusqu'à l'expiration de sa tenure. Ainsi les sommets retenus pour composer la restriction tabou et enregistrés dans la liste tabou sont considérés comme attributs tabou d'une solution.

Critère d'aspiration

Rappelons que le critère d'aspiration permet d'éliminer le statut tabou d'une solution. Pour notre algorithme, nous considérons que si une solution tabou est la meilleure rencontrée depuis le début de la recherche, elle devient la solution initiale suivante dans le processus itératif de recherche. L'algorithme 1 de recherche tabou est adapté au problème d'optimisation de parcours dans un hypermarché.

Algorithme 1. Algorithme de recherche tabou adapté

1. Définir un critère d'arrêt de la recherche
2. Prendre pour solution initiale (S_0=s) la solution donnée par l'algorithme du PPVA
3. Initialiser la liste tabou T (vide)
4. Tant que le critère d'arrêt n'est pas satisfait :
   - Mettre à jour de la liste Tabou (s'il y des solutions à libérer)
   - Prendre la meilleure solution S0 comme solution courante
   - Chercher le voisin de la solution courante en appliquant les 2 mouvements d'échange
   - Choisir le mouvement le plus intéressant (celui qui donne le moindre coût)
   - Ajouter les solutions tabou à la liste tabou T
   - Evaluer l'ensemble des solutions et définir la meilleure solution s' (avec $F(s') \leq F(s)$) comme nouvelle solution courante
5. Retourner la meilleure solution

5. Étude expérimentale et résultats

Dans cette section, nous présentons les tests que nous avons effectués afin de bien choisir la tenure et le nombre des itérations nécessaires. Nous avons généré deux types de problèmes classés selon le critère de l'appartenance aux zones du magasin. Nous avons traité des problèmes pouvant avoir 5, 10, 15, 20, 30, 40, 50, 70 et 100 articles à récupérer. Les caractéristiques de ces problèmes sont les suivantes :

Type P1 : les sommets appartiennent à la même zone.

Type P2 : les sommets appartiennent à des zones différentes

Le choix de P1 et P2 est lié à l'étude de l'impact de la taille du problème (maillage) et de la dispersion des repères sur la solution.

Avec :

N : le nombre des articles à récupérer ;

NI : le nombre des itérations de la recherche tabou ;

LT : la tenure ;

CT : le coût de la solution obtenue par la recherche taboue en mètres ;

TT : le temps de calcul de la solution tabou en millisecondes ;

CPPVA : le coût de la solution obtenue par PPVA en mètres ;

TPPVA : le temps de calcul de la solution PPVA en millisecondes.

Tableau 2. Résultats de la méthode de recherche tabou adapté et la méthode du plus proche voisin adaptée pour P1

P1 : Articles appartenant à la même zone						
N	**RTA**				**PPVA**	
	NI moyen	**LT moyenne**	**CT (mètres)**	**TT (ms)**	**CPPVA (mètres)**	**TPPVA (ms)**
5	9	4	77	21	85	15
10	10	5	103	65	111	15
15	36	27	121	226	135	16
20	47	28	130	270	159	16
30	42	24	140	488	182	31
40	35	20	164	682	194	46
50	30	16	187	773	209	57
70	10	6	232	840	236	69

Tableau 3. Résultats de la méthode de recherche tabou adapté et la méthode du plus proche voisin adaptée pour P2

P2 : Articles appartenant à deux zones différentes						
N	**RTA**				**PPVA**	
	NI moyen	**LT moyenne**	**CT (mètres)**	**TT (ms)**	**CPPVA (mètres)**	**TPPVA (ms)**
5	6	4	83	22	99	15
10	9	4	152	71	256,5	15
15	30	17	165	196	215,5	16
20	34	20	190	257	276,416	16
30	39	23	195	487	263,75	31
40	43	24	196	619	247	37
50	48	27	268	746	298,75	42
70	56	31	263	892	299	50
100	72	34	362	1021	406,5	56

Les tableaux 2 et 3 illustrent les résultats donnés par la méthode de recherche tabou adaptée (RTA) et la méthode du PPVA pour les problèmes P1 et P2.

Les tests de P1 sont effectués sur la zone 1, contenant 72 nœuds, et les tests de P2 sur la zone 1 et zone 2 qui contiennent au total 149 nœuds. Les tests ont été effectués sur un ordinateur portable avec un processeur Intel Core i5 et 4 Go de mémoire vive. Par ailleurs, nous avons utilisé des threads pour exécuter des mouvements simultanément pour diminuer le temps de calcul.

Nous remarquons que les résultats donnés par l'algorithme de recherche tabou adapté sont satisfaisants puisque le temps de calcul du chemin pour récupérer 100 articles est environ une seconde.

Pour déterminer le nombre des itérations (NI) ainsi que la valeur de la tenure (LT), nous avons effectué plusieurs tests pour chaque instance afin de trouver les valeurs qui trouvent la bonne solution avec un temps raisonnable. Si cette tenure est trop petite, le risque d'avoir un comportement cyclique du processus de recherche est grand. Si LT est trop grande, il y a un risque d'avoir de nombreux mouvements non évalués, rendant la recherche de l'optimum global excessivement coûteuse en temps de calcul (Gendreau, 1994 ; Glover, 1990). Les valeurs de NI et LT qui se trouvent dans les tableaux 2 et 3 représentent les valeurs moyennes calculées sur la base des tests effectués pour chaque instance. Dans les sous-sections suivantes, nous interprétons les résultats trouvés.

5.1. Impact de la dispersion sur la solution

Les figures 8 et 9 représentent l'amélioration apportée par la méthode de Recherche Tabou Adaptée. Nous remarquons que pour des instances (70) de taille proche du nombre total des nœuds (72), la solution du PPVA est très proche voire dans la plupart des cas égale à la solution tabou. Cependant, cette instance est très peu réaliste car il est très peu probable que l'utilisateur ajoute tous les produits d'une zone d'où l'intérêt de l'utilisation de la méthode de recherche tabou.

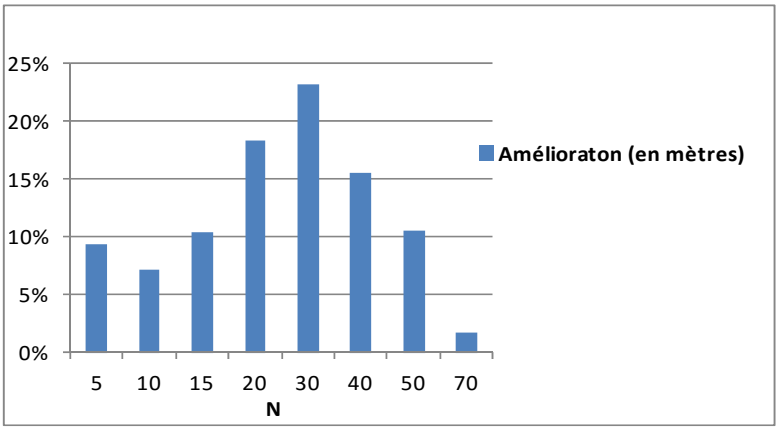

Figure 8. Amélioration de la solution pour P1

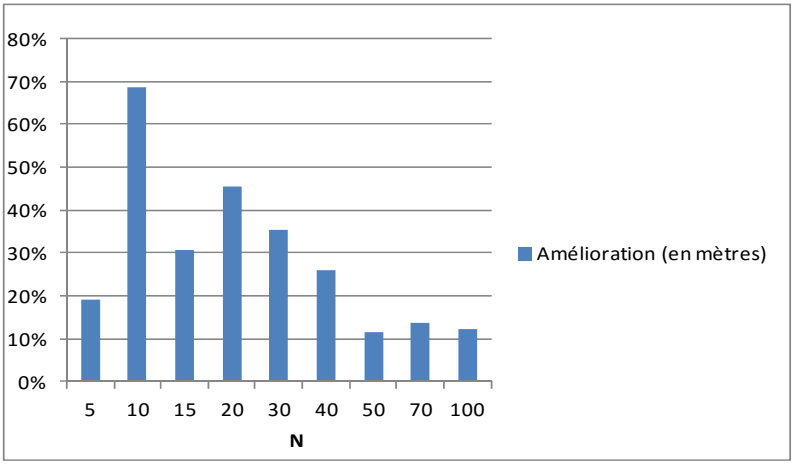

Figure 9. Amélioration de la solution pour P2

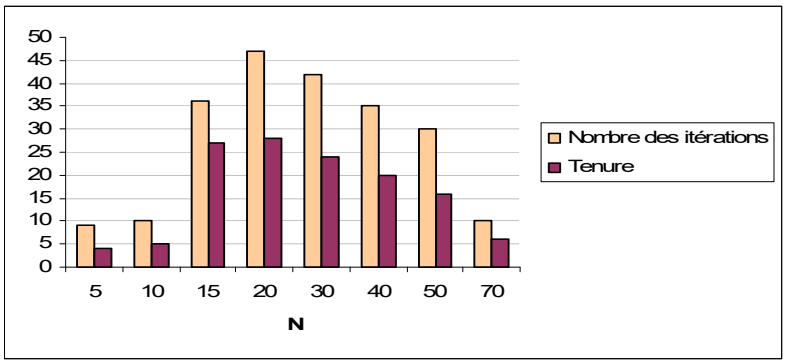

Figure 10. Convergence de la solution pour P1

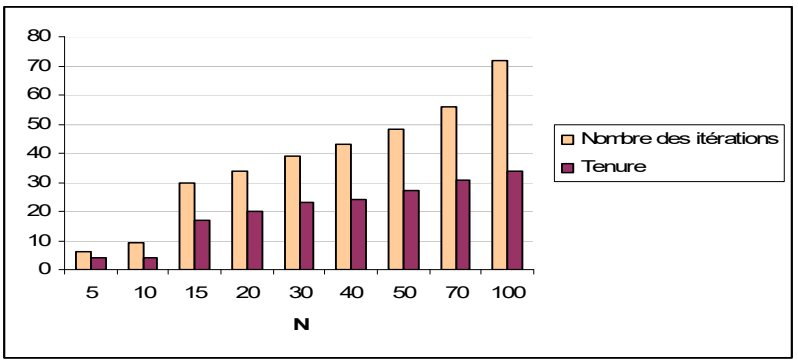

Figure 11. Convergence de la solution pour P2

5.2. Impact du maillage sur la solution

Le choix du maillage à mettre en place dépend essentiellement de la précision recherchée et du budget de l'entreprise. Dans notre cas, nous avons proposé une distance qui varie entre 2 et 3 mètres selon les dimensions des produits et la longueur de la gondole. Avec ce maillage, nous avons défini 149 repères sur les zones de test de la zone 1 et la zone 2. Plus le maillage est fin plus la taille du problème augmente. D'après les figures 9 et 10, nous remarquons que l'amélioration de la solution apportée par la méthode tabou est plus importante dans le cas où les articles appartiennent à deux zones différentes (taille du problème plus grande). Ainsi, l'application de la méthode de recherche tabou est de plus en plus intéressante quand la taille du problème est importante. D'ailleurs, au départ, nous avons appliqué un maillage plus grossier avec des distances allant de 4 à 6 mètres sur les mêmes zones de test. Les améliorations apportées par la méthode tabou étaient moins importantes.

5.3. Étude de la convergence de la solution Tabou

D'après les figures 10 et 11, nous remarquons que, pour le problème P1, ou encore dans le cas de la même zone, l'algorithme converge plus rapidement quand les instances sont de petite taille (< 10) et quand les instances ou encore le nombre des articles à récupérer s'approche du nombre total des repères de la zone. Dans le cas de la même zone, c'est à partir de 70 articles. Pour les autres instances, la convergence est moins rapide. Étant donné que le temps de calcul reste raisonnable, nous pouvons affirmer que les solutions de la méthode de recherche tabou sont de bonne qualité. Par ailleurs, à travers les tests que nous avons effectués pour chaque instance, la convergence dépend également de l'emplacement des nœuds par rapport au point initial et au point final. Ainsi, il suffit de changer un nœud de la solution de départ et la solution peut converger moins ou plus vite selon son emplacement.

6. Contributions et conclusion

Dans cet article, nous avons proposé des approches d'optimisation de parcours dans un hypermarché. Nous avons choisi le critère de la plus courte distance pour la détermination du chemin. Avant de chercher le chemin le plus court à parcourir pour trouver les articles existants dans la liste de courses, nous avons proposé une méthode pour la détermination des distances entre les articles de l'hypermarché deux à deux. Pour faciliter le calcul de ces distances, nous avons divisé le magasin en zones selon la disposition des gondoles et regroupé les articles dans des repères sur les gondoles. Après une modélisation mathématique du problème, nous avons testé l'algorithme de *Branch and Bound* et l'algorithme de Christofides ainsi que l'algorithme du Plus Proche Voisin (PPV) pour déterminer le chemin le plus court à parcourir. Ces deux dernières méthodes ont donné des résultats avec un inconvénient de revenir à un rayon déjà visité. Pour cela, nous avons proposé d'ajouter, à l'algorithme au PPV, une règle de passage par tous les articles d'un même rayon avant de passer aux autres pour améliorer les résultats. De plus, nous avons proposé de laisser les produits fragiles ainsi que les produits surgelés (cas des magasins agroalimentaires) à la fin du parcours pour ne pas les abîmer. Cependant, la méthode du PPV Adaptée se base sur une méthode qui commence généralement par faire de très bons choix en sélectionnant des arêtes de poids faible, mais, vers la fin, les sommets qui restent peuvent engendrer des distances importantes. Pour cela, nous avons amélioré la solution en appliquant la méthode de recherche tabou. Cette méthode a été testée pour deux types de problèmes P1 et P2 classés selon le critère de l'appartenance aux zones du magasin et a donné de bons résultats avec un temps raisonnable pour des articles allant jusqu'à 100.

Dans ce travail de recherche, tous nos tests ont porté sur la zone 1 et la zone 2 seulement. Cependant, nous envisageons d'élargir notre domaine d'application dans des travaux futurs. Pour le calcul des distances entre des articles appartenant à deux zones différentes, il faut penser à construire un algorithme de calcul plus générique se basant sur les méthodes de recherche du chemin le plus court entre deux points telles que la méthode Dijkstra, la méthode A*... De plus, nous testerons les

méthodes proposées avec une application mobile adaptée aux besoins des personnes handicapées. Par ailleurs, nous considérons les cas de changement de la liste de courses ou encore le changement de direction au cours du parcours.

Bibliographie

Applegate D, Bixby RE, Chvatal V, Cook W. (2007). *The Traveling Salesman Problem: a Computational Study,* Princeton University Press, Princeton.

Abboud M., Abou Jaoude L.M., Kerbage Z. (2004). *Real Time GPS Navigation System*, http://webfea-lb.fea.aub.edu.lb/proceedings/2004/SRC-ECE-27.pdf

Avella P., Boccia M., Sforza A. (2004). Solving a fuel delivery problem by heuristic and exact approaches, *European Journal of Operational Research*, vol. 152, n° 1, p. 170-179.

Bianchi L. (2000). *Notes on Dynamic Vehicle Routing - The State of the Art*. Technical Report, IDSIA-05-01.

Christofides N. (1976). *Worst-case analysis of a new heuristic for the travelling salesman problem*. Report 388, Graduate School of Industrial Administration, CM.

Gendreau M., Guertin F., Potvin J.-Y., Séguin R (1998). *Neighborhood search heuristics for a dynamic vehicle dispatching problem with pick-ups and deliveries*. Technical Report, CRT-98-10, Centre de Recherche sur les Transports, Université de Montréal.

Gendreau M., Hertz A., Laporte G. (1994). A tabu search heuristic for the vehicle routing problem, *Management Science*, 40, p. 1276-1290.

Glover F. (1989). Tabu Search—Part I, *ORSA Journal on Computing 1*, p. 190-206.

Glover F. (1990). Tabu Search—Part II, *ORSA Journal on Computing 2*, p. 4-32.

Glover F.,Taillard E., Werra D. (1993). A user's guide to tabu search, *Annals of Operations Research*, 41, p. 3-28.

Hadj Khalifa I., El Kamel A., Barfety B. (2010). Optimisation de parcours en temps réel dans un hypermarché, *ROADEF 2010*, Toulouse, France.

Hadj Khalifa I. El Kamel A., Barfety B. (2010). Real time indoor intelligent navigation system inside hypermarkets, *Large Scale Systems: Theory and Applications, LSS2010*, Lille, France.

Held M., Karp R.M (1969). The Traveling Salesman Problem and Minimal Spanning Trees, *Operations Research* 18, p. 113.

Housroum H., Goncalves G., Dupas R., Hsu T. (2004). An hybrid GA approach for solving the Dynamic Vehicle Routing Problem with Time Windows, *Francoro IV*, Fribourg, Suisse.

Johnson D.S., McGeoch L.A. (2002). Experimental Analysis of Heuristics for the STSP, *The Traveling Salesman Problem and its Variations*, Gutin and Punnen (eds), Kluwer Academic Publishers, p. 369-443.

Kilby P., P. Prosser, et P. Shaw (1998). Dynamic VRPs: A Study of Scenarios, *CSIRO*, Canberra ACT 2601, Australia.

Lacomme P., Prins C., Sevaux M. (2003). *Algorithmes de graphes*, Eyrolles 2e édition.

Laporte E G. (1992). The traveling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59, p. 231-247

Larsen A (2001). *The Dynamic Vehicle Routing Problem.* Thèse de doctorat, IMM, Technical University of Denmar.

Larsen A., Madsen O.B.G., Solomon M. (2002). Partially Dynamic Vehicle Routing – Models and Algorithms, *Journal of the Operational Research Society*, p. 637-646.

Montamenni R., Gambardella L.M., Rizzoli A.E., Donati A.V. (2002). A new algorithm for a Dynamic Vehicle Routing Problem based on Ant Colony System, *IDSIA*, Switzerland.

Pérez-Ponce H., P.R. Hernandez-Rodriguez (2004). Navigation system for visual impaired persons based on satellital location, *Proceedings of the 26th Annual International Conference of the IEEE EMBS San Francisco*, CA, USA.

Pillac V., Gendreau M., Guéret C., Medaglia A.L. (2013). A review of dynamic vehicle routing problems. *European Journal of Operational Resear,* vol. 225, p. 1-11

Ravi R., Goemans M.X. (1996). The constrained minimum spanning tree problem, *Proceedings of the Scandinavian Workshop on Algorithm Theory*, Lecture Notes on Computer Science, vol. 1097, p. 66-75

Roy S.,J.-M. Rousseau, G. Lapalme, Ferland J. A. (1984). *Routing and scheduling of transportation services for disabled: summary report*. Technical report, Centre de recherche sur les transports, Université de Montréal.

Article soumis le : 13/03/2012

Accepté le : 26/11/2015