Optimisation du remplissage de conteneurs

Projet informatique 2015

Table des matières

1 Classes utilisées

Classe	Attributs	Méthodes
Article : classe qui modélise un article	numero (int) hauteur (int) largeur (int) longueur (int) poids (double) orientation (String) materiau (String) tourne (boolean) client (Client)	Constructeur getNumero getPoids getVolume getMateriau getHauteur getLargeur getLongueur getClient getTourne estMetal toString codeClient
ArticleHauteurComparator : classe qui modélise la comparaison des hauteurs de deux articles	aucun attribut	Constructeur compare
ArticleLargeurComparator : classe qui modélise la comparaison des largeurs de deux articles	aucun attribut	Constructeur compare
Client : classe qui modélise un client	code (String) articlesDemandes (LinkedList<Articles>) nombreArticleNonPlaces (int)	Constructeur getCode getNombreArticleNonPlaces taux incrementNbArticleNonPlaces
Conteneur : classe qui modélise un conteneur	code (String) hauteur (int) largeur (int) longueur (int) poidsMax (double)	Constructeur getHauteur getLargeur getLongueur getPoidsMax getVolume toString
ConteneurRempli : classe qui modélise un conteneur rempli	conteneur (Conteneur) listePilePositionnee (LinkedList<PilePositionnee>) poids (double) posiX (int) posiY (int) nextPosiY (int) tauxDeRemplissage (double)	Constructeur getConteneur getTauxDeRemplissage getVolumeRemplissage write getCodesClientsConteneurRempli addPile

Classe	Attributs	Méthodes
Demande : classe modélisant la demande client	articlesDemandes (Articles[]) alpha (double) beta (double) gamma (double) cpmax (Conteneur) chmax (Conteneur)	Constructeur getAlpha getBeta getGamma getCpmax getChmax poidsArticles volumeArticles poidsArticles getArticleI getTailleTabArticleDemande getTailleArticleLePlusLong
Executable	aucun attribut	main
Ligne : classe modélisant une ligne d'articles	ListeArticle (LinkedList<Article>) largeur (int) = L, suivant y dans un ROND longueur (int) = l, suivant x dans un ROND hauteur (int) orientation (boolean) poids (double)	Constructeur getLargeur isSizeOK getLongueur getLargeurMin getHauteur getOrientation getPoids codesClients toString isBetaOK addArticle getLargeurMax getLongueurMin getLongueurMax
ListeClients : classe modélisant une liste de clients	ListeClients LinkedList<Client>	Constructeur addArticleToUnused canStay toString
Palette : classe modélisant une palette	ListeLigne LinkedList<Ligne> : ligne sur x, colonne sur y largeur (int) longueur (int) hauteur (int) poids (double) orientation (boolean)	Constructeur getLargeur getLongueur getHauteur getPoids getOrientation codesClients write toString isBetaOK isSizeOK isOrientationOK addLigne getLargeurMin getLargeurMax getLongueurMin getLongueurMax
PaletteLargeurPuisLongueurCompare : classe modélisant le rangement des piles par largeur puis longueur	aucun attribut	Constructeur compare
PalettePoidsComparator : classe modélisant le rangement des palettes par poids	aucun attribut	Constructeur compare

Classe	Attributs	Méthodes
Pile : classe modélisant une pile	ListePalette (LinkedList<Palette>) densite (double) largeur (int) longueur (int) hauteur (int) poids (double)	Constructeur (LinkedList<Palette> lpa) Constructeur (Palette pa, double beta) getLargeur getLongueur getVolume getPoids write codesClients estCompatible isDensityOK isBetaOK isHeightOK addPalette getLargeurMin getLargeurMax getLongueurMin getLongueurMax
PileLargeurComparator : classe modélisant le rangement des piles par largeur	aucun attribut	Constructeur compare
PilePositionnee : classe qui modélise une pile positionnée	pile (Pile) position (Point)	Constructeur getPile getPosition toString
Point	Classe professorale	non détaillée
Probleme : classe qui modélise un problème	demande (Demande) clients (ListeClients) nom (String) listeCR (LinkedList<ConteneurRempli>)	Constructeur triArticleM triArticleNM creationLignesNM creationPalettesNM creationPilesNM creationLignesM creationPalettesM creationPilesNM triParLargeur creationConteneurRempli longueurPourConteneursRemplisOptimaux addArticleToUnused getPlusPetitVolume conteneurStay getVolumeConteneursRemplis getVolumePileConteneurAQuai resoudre write tauxDeRemplissageMoyen
Test	aucun attribut	main

2 Optimisation : méthodes et stratégies

Pour résoudre ce problème informatique, nous nous sommes dans un premier temps penchés sur le modèle objet afin que celui-ci soit le plus rigoureux et le plus adaptable possible pour la suite. En effet nous nous sommes dit qu'une fois le problème bien modélisé, le résoudre serait bien plus simple et que si notre modèle objet était bien fait, les changements permettant l'optimisation seraient bien plus facile.

2.1 Résolution triviale

Nous avons travaillé sur la résolution en elle même dans un second temps mais un traitant le cas trivial d'un article par ligne, une ligne par palette, une palette par pile et une pile par conteneur. Cependant, afin de faire un code qui nous serait le plus utile possible dans les étapes suivantes d'optimisation, nous avons directement implémenté tous les tests permettant de savoir si les coefficients α, β et γ étaient respectés. Pour cela nous avons écrit des méthodes "add(Nom de l'objet)" vérifiant tous ces critères dans chacune des classes parentes, renvoyant le booléen correspondant et ajoutant l'article si possible (par exemple la méthode "addArticle" dans la classe "Ligne"), ainsi que la méthode "CanBeAdded" dans la classe "Probleme" permettant de savoir si une pile peut être ou non ajoutée à un conteneur.

2.2 Optimisation

Une fois cette solution simple fonctionnelle, il a été facile de modifier le code pour optimiser au mieux le placement des articles.

Première Idée : Laisser à terre le conteneur le moins remplis si cela est possible.

Pour réaliser cette première idée nous avons décidé d'écrire une fonction "canStay" déterminant si un conteneur rempli peut rester à terre ou non. Pour cela nous avons tout simplement compté les articles contenus dans ce conteneur (par client) puis, en les comparant au nombre d'articles demandés par les clients, nous avons calculé un ratio que nous avons comparé à γ.

Ensuite il a suffit de trouver le conteneur le moins rempli dans la liste de conteneur remplis et d'appliquer cette fonction.

Seconde Idée : Placer plusieurs piles dans un conteneur.

Pour réaliser cette idée, nous avons du changer un peu notre modèle objet en ajoutant dans la classe ConteneurRempli les attributs "posiX" (prochaine abscisse disponible), "posiY" (prochaine ordonée disponible) et "nextPosiY" (prochaine ordonnée dans le cas ou la pile ne tient pas dans le sens des x) permettant de savoir où placer la pile suivant dans le conteneur. De plus à chaque fois qu'on place une ligne, on recommence depuis le début des conteneurs remplis afin de voir si cette pile ne peut pas tenir dans un conteneur déjà rempli pour le compléter.

Troisième Idée : Faire des lignes, des palettes et des piles contenant le plus d'articles possible.

Pour cela nous avons commencé par trier notre liste d'article en séparant les articles métalliques des articles non métalliques et dans le cas des non métalliques nous avons ensuite fait un tri par hauteur et par largeur pour faire des lignes de taille maximale.

Dans le cas des articles métalliques, on procède de même mais avec seulement un article par ligne et une ligne par couche.

Quatrième Idée : Trouver la taille optimale des lignes pour un meilleur remplissage.

Pour cela nous avons créé une méthode calculant le pourcentage moyen de remplissage des conteneurs remplis. Il a ensuite fallu imposer une longueur maximale aux lignes et écrire une méthode qui fait varier cette longueur, résout le problème avec chacune de ces longueurs et choisit celle qui convient le mieux à l'instance à traiter et l'applique.

Cinquième Idée : (non implémentée) Choisir le conteneur à remplir. En effet jusque la nous utilisions le conteneur le plus grand afin de nous assurer que tous les articles allaient tenir, mais on pourrait adapter la taille du conteneur à la taille des piles afin de ne pas perdre de place.

Sixième Idée : (non implémentée) Gérer la fait que la dernière couche d'une pile n'est pas contrainte de suivre la règle du coefficient β. Cela pourrait permettre de placer les petits articles qui ne permettent pas de faire des piles.

3 Résultats numériques sur les instances fournies

Instance	Volume des conteneurs envoyés	Volume des piles restant à quai
A	207.000	0.000
B	267.290	0.000
C	467.758	0.000
D	1013.576	0.000
E	568.910	0.000
F	2539.258	24.307
Exemple	78.960	0.000

4 Problèmes rencontrés

- Rédaction de la javadoc :
  - opération longue et difficilement automatisable (copié-collés non applicables)
  - travail en binôme (accord sur le placement des ponctuations difficile à respecter systématiquement)
- Mise en place et synchronisation du Workspace commun (via DropBox)
  - travail simultané sur une classe impossible (conflit des sauvegardes)
  - mais cela a quand même grandement simplifié le travail en équipe car on pouvait coder à deux en simultané tant qu'on ne travaillait pas sur la même classe.
- Superposition du travail de deux projets : rendu du GIP qui impliquait une mobilisation des ressources non négligeables sur les premiers jours du projet informatique
- Choix de l'optimisation :
  - notre codage permet une optimisation progressive - donc compatible avec les différentes dates de rendu des livrables
  - mais il devrait être retravaillé pour atteindre une optimisation beaucoup plus pousée que celle obtenue

5 Conclusion générale sur le projet

Nous avons retiré beaucoup de plaisir à travailler ensemble, d'autant plus que nous étions relativement complémentaires dans nos compétences. Nous avons constaté moins de tensions dans notre groupe que dans d'autres groupes où les capacités des membres se valaient.

Ce genre de travail nous a permis de réaliser que, dans un travail de groupe, on peut être amené à effectuer des tâches que ne nous n'apprécions pas spécialement. Cependant, les effectuer en laissant les autres faire un travail dans lequel ils sont efficace implique la réussite au mieux du projet. Ceci met en évidence le caractère essentiel - de notre point de vue - de la complémentarité des compétences d'une équipe.

Enfin, travailler avec un binôme avec lequel on s'entend bien est la clé de la conclusion efficace d'un projet. Ceci facilite les échanges et surtout les critiques.