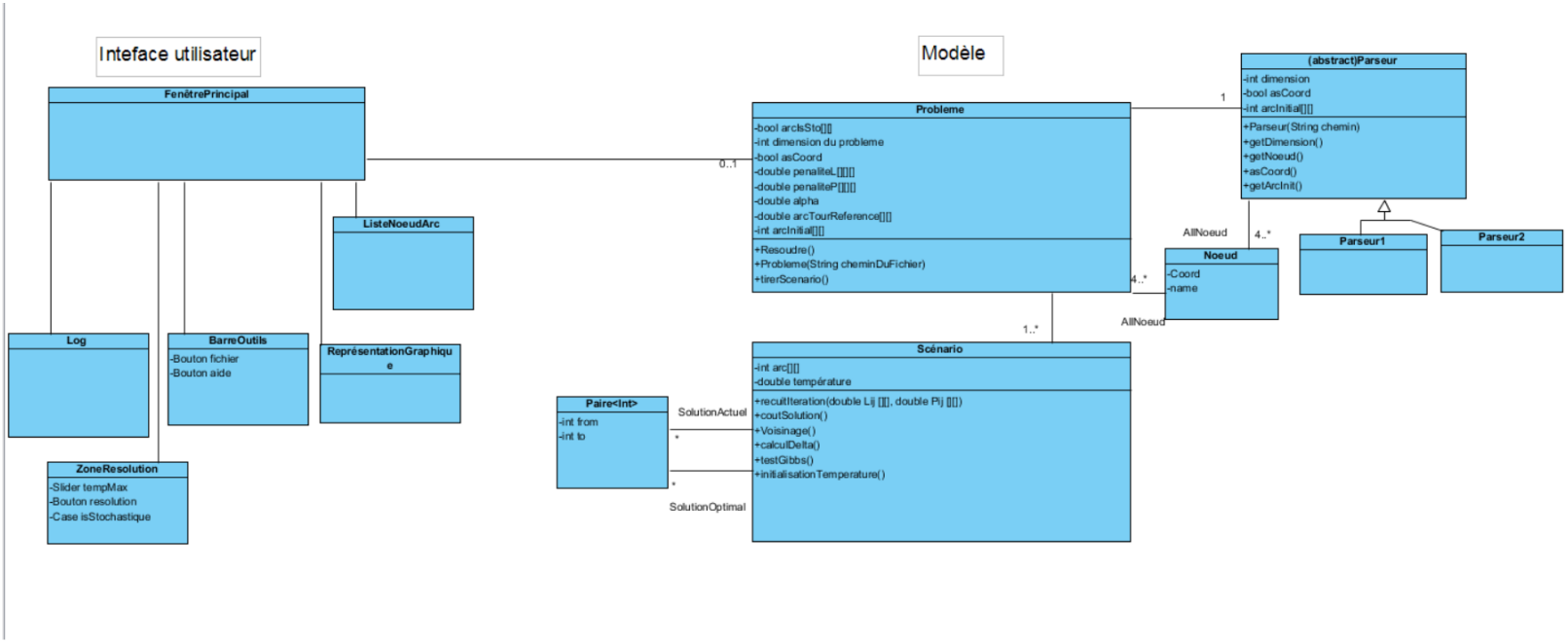
Aymeric Duplaquet
Florian Talour
Thomas Lenepveu

Projet Programmation Stochastique :

Le problème du voyageur de commerce stochastique


Inteface utilisateur
Modèle
FenêtrePrincipal
ListeNoeudArc
Log
BarreOutils
-Bouton fichier
-Bouton aide
RepréssentationGraphiqu
e
ZoneResolution
-Slider tempMax
-Bouton resolution
-Case isStochastique
0..1
Probleme
-bool arclsSto[][]
-int dimension du probleme
-bool asCoord
-double penaliteL[][][]
-double penaliteP[][][]
-double alpha
-double arcTourReference[][]
-int arcInitial[][]
+Resoudre()
+Probleme(String cheminDuFichier)
+tirerScenario()
1
(abstract)Parseur
-int dimension
-bool asCoord
-int arcInitial[][]
+Parseur(String chemin)
+getDimension()
+getNoeud()
+asCoord()
+getArcInit()
Parseur1
Parseur2
AllNoeud
4..*
Noeud
-Coord
-name
4..*
AllNoeud
1..*
Scénario
-int arc[][]
-double température
+recuitIteration(double Lij [][], double Pij [][])
+coutSolution()
+Voisinage()
+calculDelta()
+testGibbs()
+initialisationTemperature()
Paire<Int>
-int from
-int to
SolutionActuel
*
*
SolutionOptimal

Partie modèle :

Class Probleme

La classe problème, regroupe l'ensemble des scénarios du problème actuel.
La classe est instanciée par le biais de l'appel du constructeur Probleme(String chemin) retournant le problème chargé si le chemin de fichier a pu être lu et parsé en un problème de voyageur de commerce. Dans le cas contraire la fonction lance un exception.

Champs :

int dimension : Représente le nombre de noeuds que possède le problème. Existence d'un getter().
bool asCoord : Représente si les coordonnées des noeud lus ont un sens, ou si leur représentation graphique doit faire l'objet d'un algorithme spécifique. Possède un getter()
ArrayList<Noeud> allNoeud : représente les informations sur chaque noeud. Possède un getter();
bool arcIsSto[][] : un tableau en deux dimensions (de taille dimension * dimension). arcIsSto[i][j] représente si l'arrête entre la ville i et j est stochastique. Possède un getter() qui renvoie une copie du tableau.
int arcInitial[][] : un tableau en deux dimensions (de taille dimension * dimension).
arcInitial[i][j] représente le poids de l'arête lu dans le jeu de données. Possède un getter qui renvoie une copie du tableau

int nbScenario : Nombre de scénarios actuellement instanciés par le problème.
ArrayListe<Scenario> scenarios : Liste de l'ensemble des scénarios.
double alpha : représente la constante α utilisé dans le PHA.
int nbMaxIteration représentate le nombre maximal d'itérations.
double penaliteL[][][] un tableau de dimension : nbScenario*dimension*dimension
penaliteL[k][i][j] représente la pénalité λ pour l'arête entre la ville i et j pour le scénario k.
double penaliteP[][][] un tableau de dimension : nbScenario*dimension*dimension
penaliteP[k][i][j] représente la pénalité ρ pour l'arête entre la ville i et j pour le scénario k.
double arcTourReference[][] : tableau de dimension: dimension*dimension
arcTourReference[i][j]: représente la prise de l'arête entre la ville i et j dans le tour de référence.

Le problème étant symétrique, pour les tableau arcIsSto, arcInitial, arcTourReference.
Seul les case i , j tel que j < i sont utilisés.

Pour les tableaux de pénalités, la forme est similaire sur la dimension 2 et 3.

I\j	0	1	2	3
0	X	Utilisé	Utilisé	Utilisé
1	X	X	Utilisé	Utilisé
2	X	X	X	Utilisé
3	X	X	X	X

Méthodes :

Probleme(String chemin) : Constructeur de la classe Probleme.
Le paramètre chemin correspond au chemin d'accès du fichier .tsp du problème que l'on souhaite résoudre.

Le constructeur se charge de :
D'instancier un parseur, qui lira le problème. Puis d'initialiser, avec des appels successifs sur les différentes fonctions du parseur, les champs :
-dimension
-allNoeud
-asCoord
-Le tableau ArcInitial

Ensuite, le constructeur crée et initialise le tableau arcIsSto, en tirant pour chaque case utile du tableau si l'arête est stochastique ou non. (80% de chance qu'elle soit stochastique).

Il initialise la liste scenario.
Le constructeur crée ensuite une instance de scénario, en utilisant les données initiales,(sans tirer l'aléatoire des arêtes donc) et le place dans la liste scenario.
Initialement le nombre de scénario est de 1.

Enfin, il initialise un tour de référence arbitraire. Pour cela, il place toutes les cases du tableau arcTourReference[i][j] à 0 sauf les cases telles que : (j=i+1 et (i=0 j=dimension-1)) qu'il place à 1. Cela correspond au tour: Ville 1-> ville2 -> ville 3 -> …. -> ville n -> ville . Qui existe car le graphe est dense, et est hamiltonien.

void tirerScenario() : Tire un nouveau scenario et l'ajoute à la liste de scenario.
Pour cela, la méthode effectue une copie du tableau arcInitial, puis, applique l'aléatoire sur les arêtes stochastiques.
La méthode instancie ensuite une instance de scénario avec ses arcs spécifiques, l'ajoute à la liste et rajoute 1 au nombre de scénarios.
Pas de valeur de retour.

void résoudre(int tempMaxSeconde):
C'est la fonction qui va appliquer les PHA, l'algorithme est détaillé dans le document technique.

Initialise alpha à 1.2
initialise nbMaxIteration à 150
Effectue 10 recuits sur l'instance initiale de scénario et évalue le temps nécessaire moyen pour effectuer un recuit = T en seconde.
Calcul nbScenarioVoulu = tempMaxSeconde / T / nbMaxIteration
Tire des scénarios (par tirerScenario) tant que nbScenario< nbScenarioVoulu

Appelle le recuit (sans pénalité) sur tous les scénarios et fait la moyenne des solutions en les stockant dans le tableau du tour de référence.
Initialise les tableaux des pénalités et introduit les pénalités de la première itération.

Tant que nbIteration < nbMaxIteration
Appelle le recuit sur tous les scénarios et fait la moyenne des solutions en les stockant dans le tableau du tour de référence.
Mettre à jour les pénalités
nbIteration ++
fin tant que

Obtient le tour de référence final. (il n'est pas renvoyé).

Class Scénarios:

Champs:

int arc[][]: représente les arcs que contient le scénario
ArrayList<Paire<int>> solutionActuel: représente la solution actuelle calculée au cours de l'algorithme de recuit qui représente le début et la fin des arcs que contient le scenario.
ArrayList<Paire<int>> solutionOptimale: représente la meilleure solution obtenue par l'algorithme de recuit dans une liste de paires qui représente le début et la fin des arcs que contient le scénario.
double température: représente la température actuelle du recuit.

Methodes:

Le constructeur recevra l'ensemble des arcs et la dimension du problème et initialisera arc[][].

int initTempérature() calcule la température initiale:

On commence avec une température égale à la moyenne du poids des arcs, on applique ensuite le voisinage pour échanger des arcs et on regarde si on accepte ou non les voisins générés. Après 20 voisins calculés, on regarde si le pourcentage d'acceptation est supérieur à 80%: si oui on accepte la température actuelle sinon on double la température et on vérifie à nouveau si la température est la bonne.

void recruitIteration(double Lij[][], double Pij [][]) est la fonction de recuit simulé, elle fait appel aux fonctions voisinage(), coutSolution(), calculDelta, et testGibbs() pour calculer la solution du recuit. (voir document technique pour les détails)

double coutSolution() calcule le coût de la solution en sommant tous les coûts des arcs qui font partie de la solution.

Paire<Paire<int>> voisinage() renvoie les deux arcs sélectionnés pour être échangés par la technique du 2-opt et vérifie que les 2 arcs sélectionnés ne sont pas adjacents.

double calculDelta() donne la différence objective entre la solution actuelle et la solutionOptimale.

bool testGibbs renvoie un booléen qui, selon la distribution de Gibbs-Boltzman, permettra d'accepter ou non le voisinage calculé.

testGibbs est vrai dans deux cas:

- si calculDelta() est inférieur à 0
- sinon si calculDelta() est positif alors en tirant un nombre aléatoire p entre 0 et 1, si exp(-calculDelta()/t)<p

Class Noeud

Simple classe pour stocker les éléments constituants d'un noeud.

Champs:

Point2D Coordonnée : représentant les coordonnées du noeud sur un repère en deux dimensions cartésiennes. Possède un getter()
String name : nom du noeud. Possède un getter()

Méthode :

Noeud(Point2D Coordonnées, String name) : constructeur d'un noeud. Utilisé normalement uniquement par le parseur lors de la création du problème.

Class Paire< T >

Simple classe pour stocker une paire d'un type T.
Utilisé principalement pour stocker des paire<int>, utilisé sous forme de liste pour représenter une solution lors du recuit simulé.
Utilisé pour le retour de voisinage, sous forme de paire<paire<int>>, représentant une paire d'arcs.

Champs :

T first
T second
Les champs sont publics.

Méthode :

Paire<T>(T f,T s) constructeur d'une paire pour un type T.

Class Parseur

Classe abstraite imposant les fonctions que doit implémenter chaque parseur.

Champs

int dimension : représente la dimension du problème lu dans le jeu de données.
bool asCoord : boolean signalant si le jeu de données donne des coordonnées pour chaque noeud.
ArrayList<Noeud> AllNoeud : l'ensemble des noeuds du problème.
int arcInitial[][] : L'ensemble des poids des arcs entre les noeuds.

Méthodes

Parseur(String chemin) : Forme du constructeur commun à tous les parseur: n'est pas implémenté pour la classe abstraite Parseur.
Doit normalement, en lissant le fichier passé en paramètre :
Initialiser dimension,asCoord
Initialiser la liste de noeuds et la remplir avec les coordonnées si possible et un nom.
Initialiser le tableau d'arc[][], le remplir en fonction des données, éventuellement donc effectuer des calculs pour remplir ce tableau. Attention, les poids sont des entiers, l'heuristique choisie est d'arrondir à l'entier le plus proche. Pour 3.5 par exemple, arrondir à 4.
Le constructeur lance une exception si des problèmes surviennent lors de la lecture du jeu de données.

getDimension() : retourne la dimension lue dans le jeu de données
getNoeud() : retourne la liste des noeuds du problème
asCoord() : retourne le booléan asCoord
getArcInit() : retourne le tableau du poids des arcs

Partie interface utilisateur :

Class FenetrePrincipale

Classe dérivée de JFrame, organisant l'ensemble de l'interface utilisateur.

Champs:

BarreOutils barreOutils : la partie de la fenêtre pour la barre d'outils.
ListeNoeud listeNoeud : la partie de la fenêtre pour la liste de noeuds
TextArea log : la partie de la fenêtre pour les log
Canvas representationGraphique : la partie de la fenêtre pour la représentation graphique
JComponent zoneResolution : la partie de la fenêtre pour les éléments relatifs à la résolution.

Probleme problemeChargé : représente le problème actuellement chargé.

Méthode :

FenetrePrincipale() : constructeur de l'instance de FenetrePrincipale
Crée et organise dans un layout les différentes parties de la fenêtre principale.

OuvrirProbleme(String fichier) : appelé par le contrôleur du bouton "Fichier", lance l'ouverture du problème dont le chemin de fichier est donné, et si l'ouverture est réussie, signale à l'objet RepréstentationGraphique et listeNoeudArc de se mettre à jour avec les nouvelle données.

Résolution(bool isSto,int tempMax) : appelé par le contrôleur du bouton "Résolution", lance la résolution du problème actuellement chargé, avec les paramètre isSto et tempMax.

static Main() : entrée du programme, lance le constructeur de FenetrePrincipale.

Class BarreOutils :

Dérive de JToolBar

Méthodes :

BarreOutils() constructeur, créé l'objet en intégrant les deux boutons "fichier" et "aide". Ajoute les contrôleurs adaptés à ces boutons.
Le contrôleur de fichier ouvre un JFileChooser, pour la sélection du fichier de données, il signale ensuite à FenetrePrincipal qu'un nouveau problème doit être chargé.
Aide affiche une fenêtre explicitant les différentes fonctionnalités du programme.

Class ListeNoeudArc :

Classe dérivé de JTextArea , affiche les différent noeuds du programme

Champs :
ArrayList<Noeud> AllNoeud : représente l'ensemble des noeuds du problème affiché.
int arc[][] : représente les différents arcs du problème.
Méthodes :
ListeNoeudArc() : constructeur: crée l'objet avec notamment un JScrollPane pour faire défiler le texte.
Affiche(ArrayList<Noeud> , int arc[][]) : met à jour l'ensemble des informations de l'objet.

Class RepresentationGraphique

Classe dérivé de l'objet Canvas. Dessine la représentation graphique de l'objet.

Champs:
ArrayList<Noeud> AllNoeud : représente tous les noeuds
int arc[][] : représente tous les arcs.
bool isSto[][] : représente si un arc doit être affiché de manière stochastique.
bool solution[][] : représente la solution actuellement représentée par le graphe.

Méhtode:
RepresentationGraphique() : constructeur de l'objet.
Affiche(ArrayList<Noeud> , int arc[][],bool isSto[][],bool solution[][]) : met à jour les données de l'objet et applique les changements sur le graphique.

Class ZoneDeResolution

objet dérivant de Component.

méthodes:
ZoneDeResolution() constructeur de l'objet. Crée également les objets internes pour la case isStochastique, le slider de temps maximal de résolution, et le bouton résolution.
Il lie également un contrôleur sur le bouton résolution pour obtenir les informations des deux autres objets internes et signaler à la fenêtre principale qu'une résolution à besoin d'être lancée.